Methods of enhancing the MOO CEM algorithm

Veronique Tränkle

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Industrial) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof JF Bekker

March 2021

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2021

Abstract

Optimisation refers to solving a problem as accurately as possible while making the most effective use of the available resources. While some problems may have a single best solution, problems with more than one objective often do not have a single optimal solution. These types of problems are known as multi-objective problems. Improving the performance of multi-objective optimisation algorithms, both in terms of accuracy and execution time, is an active topic of research, with great emphasis being placed on computational efficiency.

The MOO CEM algorithm was developed as an alternative algorithm to solve multi-objective optimisation problems accurately and efficiently. However, the algorithm has a number of limitations. This study explores methods to improve and enhance the existing MOO CEM algorithm. These areas of improvement include: improving the sampling method of the algorithm and enhancing the algorithm by adding functionality to solve constrained problems and problems with more than two objective functions.

Following a thorough literature study of appropriate techniques, two methods of sampling improvement were identified: the Beta distribution and the use of covariance of the decision variables. In terms of solving constrained problems, two methods were evaluated: the elimination method and the dynamic penalty method. The ENS-SS algorithm was selected as the ranking and selection method, enabling the algorithm to sort (and thereby solve) problems with more than three objectives.

The proposed improved and enhanced algorithms were tested individually on a number of benchmark problems. Pareto-compliant performance indicators were used to evaluate the performance of the algorithms and, where possible, statistical tests were conducted to compare the performances to that of the original MOO CEM algorithm. It was observed that the use of the Beta distribution improved the performance of the algorithm, while using covariance did not. Adding the constraint and multi-dimensional ranking functionality yielded positive results.

Following the outcome of the tests, a final algorithm was proposed, incorporating the successful elements of the study. The final algorithm is relatively simple to implement and improves and expands the functionality of the original MOO CEM algorithm.

Opsomming

Optimering verwys na die oplossing van ’n probleem op die mees akkurate manier, terwyl die beskikbare hulpbronne so effektief moontlik benut word. Vir sommige probleme mag daar dalk slegs een beste oplossing wees, maar probleme met meer as een doelwit het dikwels nie een optimale oplossing nie. Hierdie tipe probleme word meerdoelige optimeringsprobleme genoem. Om die doeltreffendheid van meerdoelige optimeringsalgoritmes ten opsigte van akkuraatheid en uitvoeringstyd te verbeter, is ’n aktiewe studieveld waarin groot klem geplaas word op berekeningsdoeltreffendheid.

Die MOO CEM-algoritme is as alternatiewe algoritme ontwikkel om meerdoelige optimeringsprobleme op akkurater en effektiewer maniere op te los. Hierdie algoritme het egter ’n aantal beperkings. Dié studie ondersoek metodes om die bestaande MOO CEM-algoritme te verbeter en uit te brei. Die areas van verbetering sluit die volgende in: verbetering van die steekproefnemingsmetodes en uitbreiding van die algoritme met behulp van die byvoeging van funksionaliteit sodat probleme met beperkings en probleme met meer as twee doelwitte ook opgelos kan word.

Ná ’n deeglike literatuurstudie van toepaslike tegnieke, is twee metodes ter verbetering van steekproefneming geïdentifiseer: die Beta-verdeling en die gebruik van kovariansie van die besluitnemingsveranderlikes. Rakende die oplossing van probleme met beperkings is twee metodes geëvalueer: die eliminasie- en die dinamiese boetemetode. Die ENS-SS-algoritme is gekies as die rangordening-en-seleksiemetode, wat die algoritme in staat stel om probleme met meer as twee doelwitte op te los.

Die voorgestelde verbeterde en uitgebreide algoritmes is individueel op 'n aantal maatstafprobleme, getoets. Gehalte-aanwysers wat aan Pareto-standaarde voldoen, is gebruik om die doeltreffendheid van die algoritmes te evalueer en statistiese toetse is, waar moontlik, uitgevoer om die doeltreffendheid daarvan met dié van die oorspronklike MOO CEM-algoritme te vergelyk. Daar is bevind dat die gebruik van die Beta-verdeling die doeltreffendheid van die algoritme verbeter het. Daarteenoor het die gebruik van kovariansie nie die doeltreffendheid verbeter nie. Die byvoeging van beperkings- en multidimensionele rangordefunksionaliteit het positiewe resultate gelewer.

Na aanleiding van die toetsresultate is 'n finale algoritme wat die suksesvolle implementering van die elemente van die studie insluit, voorgestel. Die finale algoritme is relatief eenvoudig om te implementeer en verbeter die funksionaliteit van die oorspronklike MOO CEM-algoritme.

Acknowledgements

Throughout the writing of this thesis I received a great deal of support form the following individuals to whom I am eternally grateful:

- My husband, Nicholas Tränkle, for his unwavering love and support.
- My study leader, Professor James Bekker, for his guidance and encouragement.
- My parents for the input and motivation they offered.
- Anne Erikson, for proofreading and making valuable suggestions.
- The leadership of Deloitte's Cognitive Advantage team for granting me the time I required to complete this research.
- God, for giving me the patience and determination to complete this study.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ACO	Ant Colony Optimisation
BAP	Buffer Allocation Problem
BOS	Best Order Sort
BS	Binary Search Strategy
CE	Cross-entropy
CEM	Cross-entropy Method
CMOEA	Constrained Multi-objective Evolutionary Algorithm
CV	Pareto front convergence indicator
EA	Evolutionary Algorithms
ENS	Efficient Non-dominated Sort
GA	Genetic Algorithms
GD	Generation Distance indicator
HA	Hyperarea
HV	Hypervolume
IGD	Inverted Generational Distance

LP	Linear Programming
ME	Maximum Pareto front Error indicator
MNDS	Merge Non-dominated Sorting Algorithm for Many-Objective Optimization
MO-CMA-ES	Multi-objective Covariance Matrix Adaptation Evolution Strategy
MOO	Multi-objective Optimisation
MOO CEM	Multi-objective Optimisation using the Cross-entropy Method
MOP	Multi-objective Optimisation Problem
MPFE	Maximum Pareto Front Error
NPGA	Niched-Pareto Genetic Algorithm
NSDE	Non-dominated Sorting Differential Evolution
NSGA	Non-dominated Sorting Genetic Algorithm
ONVG	Overall Non-dominated Vector Generation
OS	Overall Pareto Spread
PAES	Pareto Archived Evolution Strategy
PSO	Particle Swarm Optimisation
SOO	Single Objective Optimisation
SOP	Single Objective Optimisation Problem
SP	Pareto front spacing indicator
SPAD	Seven Points Average Distance
SPEA	Strength Pareto Evolutionary Algorithm

SS	Sequential Search Strategy
TS	Tabu Search
UD	Uniform Distribution
VEGA	Vector Evaluated Genetic Algorithm

Greek Symbols

α	Beta distribution parameter
β	Beta distribution parameter
δ	Tolerance value
ϵ	Common termination threshold
Γ	Gamma function
λ	Number of new candidate solutions samples
μ	Mean of a distribution
ϕ	Truncated normal distribution
ρ	Rank value of multi-objective solution vector
ρ_E	MOO CEM algorithm ranking threshold
σ	Standard deviation of a distribution
Σ	Covariance matrix
τ_{ij}	MOO CEM histogram frequency count, decision variable i, class j

Roman Symbols

c_{succ}	Learning rate
c_j	$j-$th constraint

c^j_{max}	Maximum possible violation of the $j-$th constraint
d_i	Distance measure
$E(x)$	Expected value of x
f_i	Objective function
$\tilde{f}_i$	Normalised objective function
f^i_{max}	Maximum value of objective function f_i
f^i_{min}	Minimum value of objective function f_i
g	Equality constraint
h	Inequality constraint
H_0	Null hypothesis
I	Identity matrix
I_h	Hypervolume indicator
j	Total number of equalities and inequalities
k	Number of objective functions
n	Number of decision variables
N	Population size for population-based algorithms
p_i	Adaptive penalty
p_{succ}	Success rate
r_f	Ratio of feasible solutions to the population size
s	Reorder level
S	Reorder quantity
$v(x)$	Constraint violation
$\mathcal{N}$	Multi-variate normal distribution

Chapter 1

Introduction

This chapter serves as an introduction to the research presented in this thesis. Background pertaining to this research is briefly discussed, followed by the formal statement of the research assignment. Subsequently, the scope of the research, objectives, methodology, deliverables and contributions are addressed. Finally, the structure of the document is outlined.

1.1 Project background

In recent times, increasing emphasis is being placed on optimisation. Due to time and financial restrictions, problems must be solved and decisions made as quickly as possible with as little resources as possible. Banks are reliant on real-time predictive models when clients apply for loans, credit card fraud must be identified as it occurs, self-driving cars must identify objects and react to them without delay and pacemakers must monitor heartbeat and respond immediately in order to prevent tragedy. These problems can be considered optimisation problems.

When solving these problems, a number of goals (or objectives) and restrictions must be taken into account. When a single best solution does not exist, a good solution must be selected from a number of good solutions. Given time and monetary restrictions, not all solutions to a problem can be considered. A good decision must be made in the shortest possible time, making the most effective use of the available resources.

Two types of optimisation problems exist: single objective optimisation (SOO) and multi-objective optimisation (MOO). Single objective optimisation refers to a problem with one single requirement. The goal of single objective optimisation is to find the single best solution to the problem (or objective function). On the other hand, a multi-objective optimisation problem is a problem which requires taking into account more than one requirement (or multiple objective functions) and making trade-offs between conflicting requirements. In many cases, no single optimal solution exists for this type of problem, but rather a set of good solutions termed the *Pareto-optimal* solution set. By considering the solutions in the Pareto set, an informed decision can be made when selecting the 'best' solution (Savic, 2002).

A MOO problem (MOP) can be mathematically formulated as (Fan *et al.*, 2019)

Minimise or Maximise

$$\mathbf{f}(\mathbf{x}) = f_i(x_1, x_2, \ldots, x_n),\ i = 1, \ldots, k \tag{1.1}$$

subject to

$$g_j(x) >= 0,\ j = 1, \ldots, q \tag{1.2}$$

and

$$h_l(x) = 0,\ l = 1, \ldots, p \tag{1.3}$$

with

$$x \in \mathbb{R}^n,$$

where f_i represents the k objective functions to be minimised or maximised, $g_j(x) >= 0$ are the q inequality constraints and $h_l(x) = 0$ are the p equality constraints.

MOO is an active field of research. Many SOO algorithms have been extended to solve MOO problems using a variety of techniques. One such algorithm is the *multi-objective optimisation using the cross-entropy method* (MOO CEM) algorithm developed by Bekker (2012). In his dissertation, Bekker applies the cross-entropy method to multi-objective optimisation of dynamic stochastic systems to develop the MOO CEM algorithm. Initially, the algorithm is tested on a number of deterministic and continuous benchmark problems, on which the algorithm performs satisfactorily. The MOO CEM algorithm is then applied to more stringent dynamic, stochastic problems, including the buffer allocation problem (BAP) and variants thereof, as well as a number of practical problems. The performance of the algorithm is first tested using four Pareto non-compliant quality indicators. Bekker then continues to test the performance of the MOO CEM algorithm against two commercial packages: Matlab® MOO GA and OptQuest®. It was found that MOO CEM outperforms the Matlab® MOO GA algorithm. Compared to OptQuest®, the MOO CEM algorithm performed better for certain problems, while the performance of OptQuest® was superior for others. The comparison between the performance of the MOO CEM algorithm and OptQuest® is, thus, inconclusive.

In his dissertation, Bekker (2012) lists the assumptions made in the design, as well as the limitations of the algorithm. Some of these assumptions may influence the performance of the algorithm.

One such assumption is that the decision sets are independent. However, this may not necessarily be the case. A correlation could exist between the decision sets, which influences the way in which sampling is done. Bekker (2012) suggests that further research be done on the improvement of the search efficiency of the MOO CEM algorithm. He specifically recommends that a covariance structure similar to the structure used in the multi-objective covariance matrix adaptation evolution strategy (MO-CMA-ES) by Igel *et al.* (2007) be considered. By simultaneously considering both sets as well as the covariance between sets, sampling could potentially be improved.

Another assumption made in the MOO CEM algorithm design is that the truncated normal distribution is the most suitable distribution to be used for sampling. The problem with using a truncated normal distribution is the potential loss of good samples, and therefore, solutions. In order to solve this, a different type of distribution could be used instead of the truncated normal distribution. This would include samples on the extremes of the distribution

(Burkardt, 2014), which could improve sampling and, ultimately, the quality of the solutions contained in the Pareto-optimal set.

A limitation of the MOO CEM algorithm is the inability to accommodate problems with side-constraints. These types of problems differ from the standard multiple objective optimisation problems in that they contain additional complexity in the form of the limitations placed on decision variables (Woldesenbet *et al.*, 2007). In addition to standard constraints (*e.g.* the limitation of x_i such that $g_i \geq 0$), (1.2) and (1.3) could include complex relationships between decision variables, interference among constraints and interrelationships between constraints and objective functions (Woldesenbet *et al.*, 2007).

A second limitation of the MOO CEM algorithm is that it cannot solve problems with more than two objective functions. This is due to the ranking method used, i.e. the Goldberg method. By using a different ranking method, the algorithm could be extended to solve problems in N-dimensional space.

Considering the assumptions and limitations highlighted above, it is hypothesised that the MOO CEM algorithm could be enhanced. This research explores the possibility of developing a more efficient MOO CEM algorithm with an extended scope which will allow it to solve additional classes of problems. The research hypothesis, based on these assumptions and limitations, is presented in the next section.

1.2 Research assignment

MOO is an active field of research. Many researchers are attempting to improve simulation optimisation in terms of computational burden and time (and by implication, cost). In his dissertation, Bekker (2012) successfully applies the cross-entropy method to multi-objective optimisation of dynamic stochastic systems to reduce computational burden. In his thesis, Bekker (2012) suggests that in future work the MOO CEM algorithm could be enhanced. He suggests that the algorithm could be improved by considering covariance: "the correlation of solution sets should be investigated to improve search efficiency of the MOO CEM algorithm. A covariance structure similar to that of the MO-CMA-ES reported by Igel *et al.* (2007) may be considered". Bekker (2012) also advises that the capabilities of the MOO CEM algorithm should be extended to cater for problems with more than two objectives: "All the objectives in this research were limited to two objectives, but the MOO CEM algorithm should be assessed with respect to problems of higher objective dimensions".

The aim of this research is to enhance the MOO CEM algorithm by improving its efficiency and extending the functionality of the algorithm. By doing so, the search efficiency of the algorithm could be improved and the scope of the algorithm could be extended to solve additional types of problems.

The research assignment can thus be summarised as

Enhance the MOO CEM algorithm by improving the sampling method and extend the functionality of the algorithm to solve a wider range of problem types.

1.3 Research scope

The research will focus only on MOO and MOO CEM, not SOO. The two methods of improving the algorithm suggested by Bekker (2012), the MOO CEM algorithm developer, in his dissertation will be considered, as well as two additional methods of enhancement and improvement. These four methods are described in detail in the next section.

In order to quantify the improvement of the algorithm, the Pareto-compliant performance indicators produced by the new algorithm will be compared to that of the original MOO CEM algorithm for a set of benchmark problems. To determine whether the functionality of the algorithm has been extended successfully, Pareto-compliant performance indicators will be utilised.

The enhanced algorithm will not be compared to existing MOO algorithms other than MOO CEM, as the original MOO CEM algorithm was compared to other algorithms in Bekker (2012)'s dissertation.

1.4 Research objectives

Considering the research assignment, the research project has the following objectives:

1. *Determine* whether the sampling method of the MOO CEM algorithm can be improved by using the Beta distribution, rather than a truncated normal distribution. If the Beta distribution can be used to improve the sampling method, *develop* an improved MOO CEM algorithm which utilises the Beta distribution and *compare* the new algorithm to the original MOO CEM algorithm for a number of benchmark problems.

2. *Determine* if the sampling method of the MOO CEM algorithm can be improved by considering the covariance of the solution sets (in a similar manner to MO-CMA-ES). If covariance can be used to improve the sampling method, *develop* an improved MOO CEM algorithm which utilises covariance and *compare* the new algorithm to the original MOO CEM algorithm for a number of benchmark problems.

3. *Extend* the capability of the existing MOO CEM algorithm to include functionality to solve side-constrained problems and *compare* the results of the new algorithm to the true Pareto set for a number of benchmark problems.

4. *Extend* the scope of the current MOO CEM algorithm to include functionality to solve problems with more than two objectives and *compare* the results of the new algorithm to the true Pareto set for a number of benchmark problems.

1.5 Problem-solving methodology

In order to meet the objectives outlined in the previous section, this research project will follow a modified engineering design process depicted in Figure 1.1 (**?**). The engineering design process is used to structure ideas and refine solutions to engineering problems. With slight modification, this process can be applied to algorithm development in this research project.

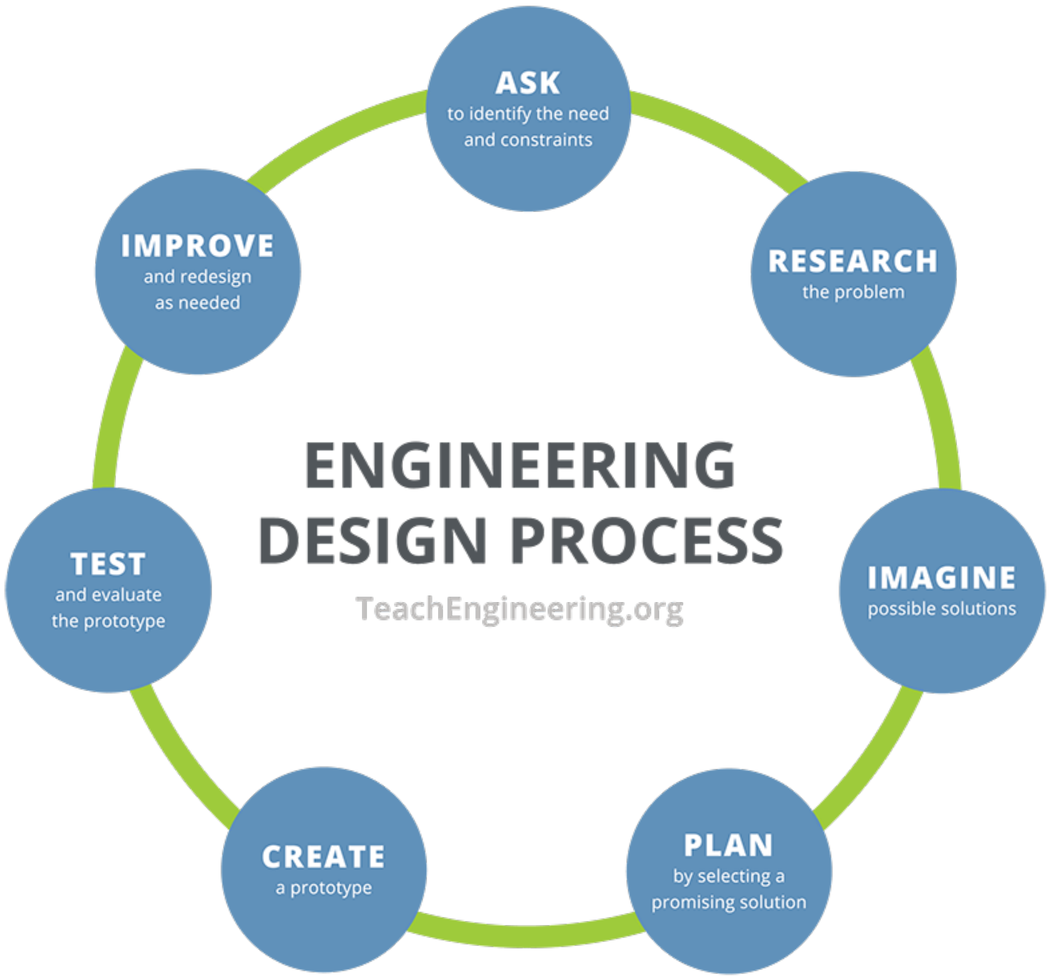

ASK
to identify the need and constraints
RESEARCH
the problem
IMAGINE
possible solutions
PLAN
by selecting a promising solution
CREATE
a prototype
TEST
and evaluate the prototype
IMPROVE
and redesign as needed
ENGINEERING DESIGN PROCESS
TeachEngineering.org

Figure 1.1: The Engineering Design Process (**?**)

The first step of the engineering design process is **Ask**. This step requires defining the research problem in detail, as well as the requirements and limitations. This research project aims to enhance the MOO CEM algorithm, limited to the four enhancement methods listed in the previous section.

The second step in the process is **Research**. MOO and specifically the MOO CEM algorithm will be researched extensively in Chapter 2. The four enhancement methods will also be investigated.

Next, one must **Imagine** the possible solutions. This refers specifically to the four enhancements which will be applied to the MOO CEM algorithm and methods of doing so highlighted in the **Research** step.

Plan is the fourth step in the process. Upon identifying possible methods of creating the enhancements to the algorithm, the most promising methods must be selected.

The fifth step is **Create**. This refers to the implementation of the selected methods in the existing MOO CEM algorithm. Following the review of the MOO CEM algorithm and possible improvement and enhancement methods, four algorithms corresponding to these methods are proposed in Chapter 3.

Next, the developed algorithm must be evaluated in the **Test** step. For the two sampling improvement enhancements, the results of the new algorithm will be compared to the results generated by the original MOO CEM algorithm for a set of benchmark problems. For the two enhancement methods which extend the scope of the algorithm, the algorithm will be tested on standard problems and compared to the true Pareto set. The specifications of the tests and test results are presented in Chapter 4.

Improve is the last step in the process. If the new algorithm does not show significant improvement over the existing algorithm, the reason for this should be investigated and improved where possible.

This process will be followed for each of the four enhancement methods. The engineering design process is iterative and can be conducted multiple times until satisfactory improvement is observed.

1.6 Deliverables envisaged

The following items will be delivered as part of this research project:

1. A recommendation will be made on whether the Beta distribution can be used to improve sampling of the MOO CEM algorithm. If the sampling method can in fact be improved using the Beta distribution, a new optimisation algorithm will have been developed and a comparison between the new algorithm and the original MOO CEM algorithm will have been

done to determine the difference in performance for a number of benchmark problems.

2. A recommendation will be made on whether the covariance of solution sets can be used to improve sampling of the MOO CEM algorithm. If sampling can in fact be improved using covariance, a new optimisation algorithm will have been developed and a comparison between the new algorithm and the original MOO CEM algorithm will have been done to determine the difference in performance for a number of benchmark problems.

3. An enhanced MOO CEM algorithm with the extended capability to cater for problems with side-constraints will have been developed. A comparison between the elite set produced by the algorithm and the true Pareto set will also have been conducted.

4. An enhanced MOO CEM algorithm with the extended capability to cater for problems with more than two objective functions will have been developed. A comparison between the result set produced by the algorithm and the true Pareto set will also have been conducted.

1.7 Contributions

The primary value of this research lies in the improvement of the performance and enhancement of the capabilities of the MOO CEM algorithm. This could benefit the academic community and those using the current MOO CEM algorithm to solve MOO problems.

If it is found that the performance of the MOO CEM algorithm can be improved, the quality of the solutions generated could be increased. Additional benefits could include the reduction in required computational resources, run-time and, therefore, cost. By enhancing the capabilities of the algorithm, researchers will be enabled to solve problems which could not previously be solved by the MOO CEM algorithm. These types of problems include problems with side-constraints and problems with more than two objective functions.

1.8 Structure of the document

The structure of this thesis document is as follows:

Chapter 2 will present a high level literature review on MOO. It will focus specifically on how the cross-entropy method has been applied to improve MOO problems and the MOO CEM algorithm. It will also explore the four methods of enhancement listed in Section 1.4 and consider different techniques of implementation. The MOO CEM component of this chapter is centred around previous research completed by Bekker (2012).

Upon completion of the literature study, four enhanced MOO CEM algorithms will be presented in Chapter 3.

In Chapter 4 the algorithms developed in Chapter 3 will be applied to a number of benchmark problems. The performance of the algorithm will be analysed. For the two sampling improvement methods, performance of the algorithm will be compared to that of the original MOO CEM algorithm when applied to the benchmark problems. For the two enhancement methods, a comparison between the algorithm results and the true Pareto set will be presented. The results in this chapter will be used to determine which of the sampling methods improved sampling of the MOO CEM algorithm and whether the functionality of the MOO CEM algorithm was extended successfully.

Considering the results produced in Chapter 4, Chapter 5 will propose a final algorithm, incorporating the successful enhancement methods.

Chapter 6 will serve as the conclusion to the research project. It will review the viability of using the four suggested methods to improve and enhance the MOO CEM algorithm.

1.9 Summary: Chapter 1

This chapter has served as an introduction to the outline research problem. It described the objectives that will be met in order to achieve the overall goal of the project: to enhance the MOO CEM algorithm through four specific methods.

The results of this research could benefit the research community and those using the current MOO CEM algorithm in terms of the quality of the solution sets, required computational power, time and cost, and will give users the ability to apply the algorithm to additional classes of problems.

A literature study of MOO, specifically focused on MOO CEM, the four enhancement methods and benchmark problems, is presented next.

Chapter 2

Literature study

In the previous chapter, the concept of optimisation was introduced, and the scope and objectives of this research project were defined. The problem-solving methodology followed was described and the structure of this document was outlined. This chapter serves as an introduction to optimisation, with specific focus on multi-objective optimisation (MOO) and multi-objective optimisation using the cross-entropy method (MOO CEM) algorithm. It also explores the four suggested types of enhancements to the MOO CEM algorithm as outlined in Section 1.4: the Beta distribution, covariance, constraint handling and non-dominated sorting. Finally, some of the methods of algorithm evaluation and standard benchmark problems which will be used to evaluate the proposed algorithms are listed.

2.1 Multi-objective optimisation and MOO CEM

In this section an introduction to optimisation and the MOO CEM algorithm are presented.

2.1.1 Introduction to optimisation

Optimisation is the art of solving a problem in such a way that the best solution is selected using the available resources in the most effective manner possible (Astolfi, 2006). It is a problem which dates back thousands of years - from selecting a cave which provides protection while offering a good view of the immediate surroundings, to the optimal number of livestock to keep, given the available land and resources.

The first formal optimisation techniques were developed by Sir Isaac Newton (1660) and Gottfried Wilhelm von Leibniz (1670) with their study of differential equations (Theodossiou *et al.*, 2014). In 1939 Linear Programming (LP), the concept on which combinatorial optimisation is built, was formulated by Leonid Kantorovich. Then in 1947 both the Simplex Method and the Theory of Duality were published by George Dantzig and Johan von Neumann respectively (Wang, 2018). Based upon these theories, many optimisation techniques were developed, including gradient-based methods (such as the Golden Section search, the Fibonacci search, Karush-Kuhn-Tucker Conditions and Newton's method), and Evolutionary Algorithms (EA) (such as the Genetic Algorithm (GA), Particle Swarm Optimisation (PSO), Tabu Search (TS) and Ant Colony Optimisation (ACO)) (Venter, 2010).

The optimisation of a problem with a single goal (or objective) is termed single objective optimisation (SOO). Identifying the best solution to a SOO problem (SOP) is generally simpler than identifying the best solution to a MOO problem, as one best solution normally exists for SOO problems, while this is often not the case for MOO problems.

2.1.2 Multi-objective optimisation

Real-world scenarios often require multiple goals to be met; goals which are frequently contradictory (Savic, 2002). For example, when designing a vehicle, the material cost must be minimised, while reliability and safety are maximised. These types of problems are known as multi-objective optimisation problems (MOP). When solving an MOP, generally, no single best solution exists. Instead, a set of good solutions are selected, known as the Pareto-optimal solution set (Quiza Sardiñas *et al.*, 2006).

Figure 2.1 shows a sample dataset evaluated at two objective functions (f_1 and f_2). In this instance, the MOP has two objective functions and both functions are to be minimised. The red square data points indicate the Pareto front (while other data points which do not form part of the Pareto front are indicated in blue):

the best solutions when both objective functions are evaluated simultaneously. These solutions are known as the Pareto-optimal solution set according to the principle of Pareto Dominance. Pareto Dominance is defined as

$$\begin{array}{ll} \mathbf{a} \succ \mathbf{b} \ (\mathbf{a} \text{ dominates } \mathbf{b}) & \textit{iff } \mathbf{f}(\mathbf{a}) > \mathbf{f}(\mathbf{b}) \\ \mathbf{a} \succeq \mathbf{b} \ (\mathbf{a} \text{ weakly dominates } \mathbf{b}) & \textit{iff } \mathbf{f}(\mathbf{a}) \geq \mathbf{f}(\mathbf{b}) \\ \mathbf{a} \sim \mathbf{b} \ (\mathbf{a} \text{ is indifferent to } \mathbf{b}) & \textit{iff } \mathbf{f}(\mathbf{a}) \ngeq \mathbf{f}(\mathbf{b}) \wedge \mathbf{f}(\mathbf{b}) \ngeq \mathbf{f}(\mathbf{a}) \end{array}$$

where **a** and **b** are two decision vectors (Zitzler, 1999).

If decision vector a is found to dominate b, a is a non-dominated solution vector. This implies that a is optimal and cannot be improved in one objective without worsening the solution in at least one other objective. These solutions are denoted as Pareto optimal (Zitzler, 1999).

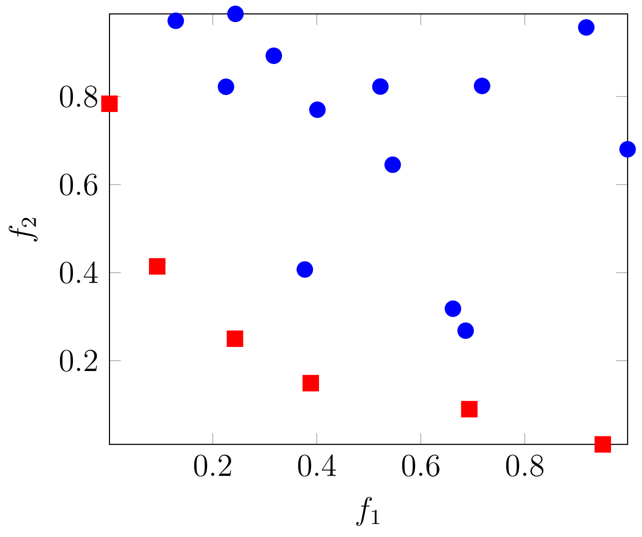


Figure 2.1: Pareto front (red squares) for a MOO problem where two objective functions (f_1 and f_2) are to be minimised

An MOP is defined by (1.1) - (1.3). A MOO algorithm finds a set of good solutions (Pareto solution set) to the k objective functions (by minimising or maximising the functions) represented by (1.1) in terms of the n decision variables, while complying with the q inequality constraints in (1.2) and p equality constraints in (1.3). The n decision variables are defined on a decision space with lower and upper bounds.

An unconstrained problem is defined using only objective functions and has no equality or inequality constraints. Constrained problems are subject to equality and/or inequality constraints and may include side-constraints. Side-constraints add an additional level of complexity to constrained and unconstrained problems. Side-constraints are normally considered separately from equality and inequality constraints (Venter, 2010). Side-constraints will be discussed in more detail in Section 2.4.

Since the 1950s a variety of approaches have been developed to solve MOPs. Today many mathematical programming techniques exist which can be used to solve MOPs. However, these techniques are limited by, amongst other things, the shape of the Pareto front, and the differentiability of the objective functions and constraints. Another disadvantage of some of these techniques is that most algorithms generate a single solution after each run, requiring several runs with different starting points to generate an acceptable Pareto-optimal solution set (Coello, 2006).

Due to the limitations of mathematical programming, other approaches and techniques have been explored to solve MOPs - one of the most popular being evolutionary algorithms (EA). Evolutionary algorithms are not influenced by the shape of the Pareto front, or the differentiability of the objective functions. EAs also have the ability to consider an entire population simultaneously, rather than single solutions, which allows for numerous Pareto-optimal solutions to be generated after each run. Some popular EA algorithms include the Vector Evaluated Genetic Algorithm (VEGA), developed by David Schaffer in the 1980s, the Non-dominated Sorting Genetic Algorithm (NSGA) developed by Srinivas and Deb, NSGA-II and NSGA-III, Niched-Pareto Genetic Algorithm (NPGA), Multi-Objective Genetic Algorithm (MOGA), Strength Pareto Evolutionary Algorithm (SPEA) and SPEA2 and Pareto Archived Evolution Strategy (PAES) (Coello, 2006).

Another recently developed MOO algorithm leverages the benefits of the cross-entropy method and applies it to the field of multi-objective optimisation. The

MOO CEM algorithm was developed by Bekker (2012). It was applied to a number of benchmark problems, as well as a range of industry-specific problems.

2.1.3 Cross-entropy method

The cross-entropy method (CEM) was developed by Rubinstein & Kroese (2004) and is based on the *Kullback-Leibler* or *cross-entropy* (CE) *distance*. This adaptive importance sampling algorithm was first used to estimate the probability of rare events. It was then extended to combinatorial optimisation problems by utilising the cross-entropy divergence as a measure of closeness between two distributions. In essence, a very small probability (rare-event) exists of finding an optimal solution through naive, random sampling. By using the cross-entropy method, the distributions from which the points are sampled can be adjusted, so the probability of the rare event occurring is increased. Over time, the sampling distribution converges to a distribution concentrated around optimal (or near-optimal) solutions (Rubinstein & Kroese, 2004).

The cross-entropy method is iterative, with each iteration consisting of two distinct parts (Rubinstein & Kroese, 2004):

1. Generate a random data sample according to a specified mechanism.
2. Update the parameters of the random mechanism based on the data to produce a better sample in the next generation.

Based on these two iterative steps, Algorithm 1 shows the main cross-entropy algorithm for continuous optimisation.

Algorithm 1 Cross-entropy Algorithm for continuous optimisation

```
1: Choose some $\hat{\mathbf{v}}_0$ for the density $h(\cdot;\mathbf{v})$. Set $t = 1$.
2: Generate a sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ from the density $h(\cdot;\hat{\mathbf{v}}_{t-1})$ and compute the
   $(1-\varrho)$-quantile $\hat{\gamma}_t$ of the performances according to (2.1).
3: Use the sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ and solve the stochastic program in (2.2). This
   solution is $\mathbf{v}_t$.
4: Smooth the vector $\mathbf{v}_t$ using the expression in (2.3).
5: If, for some $t \geq \delta$, say $\delta = 5$, $\hat{\gamma}_t = \hat{\gamma}_{t-1} = \cdots = \hat{\gamma}_\delta$ then stop, otherwise set
   $t \leftarrow t + 1$ and return to Step 2.
```

$\mathbf{v}$ represents a reference parameter vector and γ is the cross-entropy optimisation rare-event threshold value. $\mathbf{X}$ is a random vector $(X1, \ldots, Xn)$ and γ_t is updated adaptively according to

$$\hat{\gamma}_t = f_{([(1-\varrho)N])}, \tag{2.1}$$

where ϱ is the user-specified rare-event threshold value and is typically chosen to be 10^{-2}.

$\mathbf{v}_t$ is calculated by solving the stochastic program

$$\max_v \hat{D}(v) = \max_v \frac{1}{N} \sum_{i=1}^{N} I_{\{f(\mathbf{X} \geq \hat{\gamma}_t)\}} \ln h(\mathbf{X}_i; v). \tag{2.2}$$

The parameter vector $\mathbf{v}$ is smoothed using the smoothing function

$$\hat{\mathbf{v}}_t = \omega \tilde{\mathbf{v}}_t + (1-\omega)\hat{\mathbf{v}}_{t-1}. \tag{2.3}$$

where ω is a smoothing constant in the range $0 - 1$ (typically $0.6 - 0.9$).

2.1.4 Multi-objective optimisation using the cross-entropy method

In his dissertation, Bekker (2012) starts by applying the cross-entropy method to four continuous SOO benchmark problems, namely De Jong's first function, the Rosenbrock function, the Shekel function and the Rastrigin function. Promising results were achieved, with the largest number of iterations to solve each of the four problems being a mere 4 000 iterations.

Given the results of the algorithm, Bekker (2012) then extended the application of the cross-entropy method to multi-objective optimisation to develop the MOO CEM algorithm described by Algorithm 2.

Algorithm 2 MOO CEM Algorithm

```
 1: Set $Elite = \varnothing, t = 1, k = 1$.
 2: Initialise variable vectors $\mathbf{X}_i = \varnothing, 1 \leq i \leq D$, and compute initial objective
    values.
 3: For each decision variable $x_i, 1 \leq i \leq D$ initialise a histogram class vec-
    tor $\mathbf{C}_i = \{c_{i1}, c_{i2}, \ldots, c_{i(r+2)}, c_{i(r+2)+1}\}$ and histogram frequency vector $\mathbf{R}_i =$
    $\{\tau_{i1}, \tau_{i2}, \ldots, \tau_{i(r+1)}, \tau_{i(r+2)}\}$.
 4: Set $i = 1$.
 5: Set $\kappa = 0$.
 6: Increment $\kappa$.
 7: for each frequency element $\tau_{i\kappa}$ in $\mathbf{R}_i$ do
 8:     Generate a class-based $\tilde{\mathbf{v}}'$ in the range $[c_{i\kappa}\, c_{i(\kappa+1)})$
 9:     Generate a subsample $Y$ according to pdf $\phi_i(\mathbf{x}_i\, \tilde{\mathbf{v}}')$
10:     with $\mathbf{x}_i \epsilon [c_{i\kappa}, c_{i(\kappa+1)})$ and $|\mathbf{Y}| = \tau_{i\kappa}, 1 \leq \kappa \leq r + 2$ .
11:     Append $\mathbf{Y}$ to $\mathbf{X}_i$.
12: end for
13: If $\kappa < r + 2$ , return to Step 6.
14: Invert the histogram counts with probability $p_h$.
15: Increment $i$.
16: If $i \leq D$, return to Step 5.
17: Compute the $NK$ objective function values using $\mathbf{X}_i, 1 \leq i \leq D$
18: Rank the objective function values using the Pareto ranking of Algorithm 3
    with a relaxed $\rho_E = 2$ to obtain an updated elite vector Elite.
19: Form new histogram class vectors $\mathbf{C}_i$ and histogram frequency vectors $\mathbf{R}_i$
    based on Elite , $1 \leq i \leq D$.
20: Use the values in Elite and compute $\tilde{\mathbf{v}}'$ for all $i, 1 \leq i \leq D$.
21: Smooth the vectors $\tilde{\mathbf{v}}'$ for all $i, 1 \leq i \leq D$, using (2.3).
22: If all $\sigma_{it} > \epsilon_c$ or less than the allowable number of evaluations have been
    done, increment $t$ and reiterate from Step 4.
23: Rank the elite vector Elite using the Pareto ranking of Algorithm 3 with
    $\rho_E = 1$.
24: Increment $k$.
25: If $k$ is smaller than the allowable number of loops, return to Step 2.
26: Rank the elite vector Elite using the Pareto ranking of Bekker Algorithm 3
    with $\rho_E = 0$ to obtain the final elite vector.
```

where ϵ_c is a common threshold (a small number) implying that if a value does not change by an amount greater than this threshold ϵ_c, the algorithm has reached steady-state and further iterations would most likely not improve the solution. D represents the number of decision variables and K is the number of objectives. N is the number of solutions.

The MOO CEM algorithm uses the Goldberg (1989) Pareto ranking algorithm (outlined in Algorithm 3) as a means of finding the set of best solutions, or Pareto set, shown in Figure 2.1. The implementation of the Goldberg (1989) method is discussed in more detail in Section 2.5.

Algorithm 3 Pareto ranking algorithm (minimisation) implemented in MOO CEM

```
 1: Input: working matrix $\mathbf{W}$ with $N$ rows and $D+K+1$ columns, and user-
    selected threshold $\rho_E$.
 2: $j \leftarrow D+1$.
 3: Sort the working matrix $\mathbf{W}$ with the values in column $j$ in *descending* order.
 4: $r_p \leftarrow 1$.
 5: $r_q \leftarrow 1$.
 6: If $\mathbf{W}(r_p, j+1) \geq \mathbf{W}(r_q+1, j+1)$, increment the rank value $\rho_{r_p}$ in $\mathbf{W}(r_p, D+K+1)$.
 7: $r_q \leftarrow r_q + 1$.
 8: If $\mathbf{W}(r_p, D+K+1) < \rho_E$ and $r_q < N$, return to Step 6.
 9: $r_p \leftarrow r_p + 1$.
10: If $r_p < N$, return to Step 5.
11: $j \leftarrow j + 1$
12: If $j < D+K-1$, return to Step 3, otherwise return the rows in $\mathbf{W}$ with rank
    value not exceeding $\rho_E$ as the weakly or non-dominated vector **Elite**.
```

ρ_E represents the number of the Pareto set with 0 being the set of optimal solutions, and therefore, the first Pareto front. $\rho_E = 1$ represents the second Pareto front, with the set of second-best solutions.

The MOO CEM algorithm was then tested on a number of benchmark problems (which are discussed in detail in Section 2.6). The maximum number of evaluations was limited to 10 000, which is far less than the number of evaluations required by other MOO algorithms, such as the algorithm developed by Zitzler (1999) which required 25 000 evaluations. The results showed that the MOO CEM algorithm could achieve proximity to the true Pareto front, while maintaining diversity.

Considering the positive results, the algorithm was applied to various real-world problems: an inventory problem, the buffer allocation problem (BAP), an extrusion equipment design problem and a CO gas management problem. The performance of the algorithm was then compared to Matlab®'s commercial MOO genetic algorithm (GA) which is based on the NSGA-II algorithm. Four quality indicators were used: Pareto front spacing indicator (SP), Generation Distance indicator (GD), Maximum Pareto front error indicator (ME) and Pareto front convergence indicator (CV). The hyperarea and epsilon indicator were calculated and a two-tailed t-test was performed to determine if there was a statistically significant difference between the results produced by the MOO CEM algorithm and those of the MOO GA algorithm. The algorithm which produces the larger hyperarea and smaller epsilon indicator is considered the best algorithm. For 7 of the 8 test problems, the MOO CEM proved to be superior to the MOO GA algorithm (one test problem produced inconclusive results). The MOO CEM algorithm also required fewer evaluations than the MOO GA algorithm. Finally, the MOO CEM algorithm was compared to the commercial OptQuest®for two test problems: a BAP and an inventory problem. The MOO CEM algorithm produced better results for the inventory problem, but the OptQuest®algorithm achieved better results for the BAP, concluding that additional tests would be required to compare the algorithms, as the results were inconclusive (Bekker, 2012).

As recommendations for future work, Bekker (2012) suggests, amongst others, the following areas:

1. Considering a different distribution to the truncated normal distribution to improve search efficiency or sampling.
2. Investigating the correlation of solution sets to improve search efficiency.
3. Applying the MOO CEM algorithm to constrained problems.
4. Applying the MOO CEM algorithm to problems with more than two objectives.

The MOO CEM algorithm uses truncated normal distributions (an example of a truncated normal distribution is shown in Figure 2.2) to sample solutions. This type of distribution runs the risk of omitting good solutions located on the edges of the distribution. It is hypothesised that a different type of distribution, which would include the solutions located at the edges, would improve the search efficiency of the algorithm as well as the quality of the final solution set. Appropriate distributions are considered in the subsequent section.

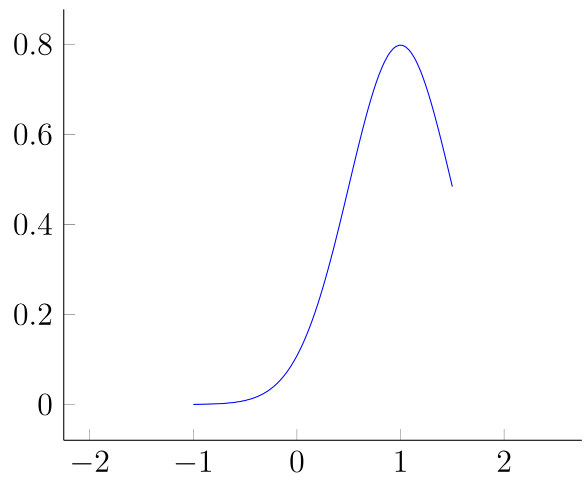


Figure 2.2: A truncated normal distribution

The MOO CEM algorithm makes the assumption that no relationship exists between the solution sets of the decision variables. If it is found that a correlation does in fact exist between them, this property could be leveraging to improve the search efficiency of the algorithm. Bekker (2012) refers specifically to the MO-CMA-ES developed by Igel *et al.* (2007). The concept of covariance and the work by Igel *et al.* (2007) are analysed in Section 2.3.

Currently, the MOO CEM algorithm is not able to solve problems with side-constraints. By adding the functionality to handle side-constraints, the algorithm could be applied to this class of problems. Methods of solving side-constraint problems are explored in Section 2.4.

In order to rank and evaluate the sampled solutions, the MOO CEM algorithm uses the Goldberg (1989) Pareto algorithm (Algorithm (3)). This algorithm was implemented in such a manner that it limits the maximum number of objectives to two. The algorithm has a high complexity ($\mathcal{O}(KN^2)$) making it extremely computationally expensive and inefficient in terms of runtime for problems with a higher number of objectives than two. Appropriate non-dominated sorting algorithms which are less computationally intensive are explored in Section 2.5.

2.2 Beta distribution

Many different probability distributions exist, some of the most widely-used include: the Gaussian (or normal) distribution and modified versions thereof, such as the truncated normal distribution used by the original MOO CEM algorithm, uniform distribution, chi-square distribution, binomial distribution, Poisson distribution and Beta distribution.

The Beta distribution is extremely versatile and can approximate many other distributions using only two parameters: α and β. These two parameters specify whether the distribution is symmetrical and whether the distribution's mode falls within the range of the distribution. The Beta distribution is continuous and defined over the interval $(0, 1)$ (Bury, 2012).

The Beta probability density function is mathematically defined as (Bury, 2012)

$$\mathrm{Beta}(\alpha, \beta) : \mathrm{prob}(x|\alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, \tag{2.4}$$

where α and β are positive and real and B is the function

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}. \tag{2.5}$$

Γ in (2.5) refers to the Γ function, which is defined as

$$\Gamma(x) = (x-1)!, \tag{2.6}$$

for any positive integer x. The Γ function is extended to any real number such that

$$\Gamma(x) = \int_0^\infty t^{x-1} \exp^{-t} \, \mathrm{d}t. \tag{2.7}$$

It is evident that the Beta distribution is relatively computationally inexpensive, if α and β are positive integers. This is as a result of the definition of the *Gamma* function. When α and β are non-integer, evaluating the *Gamma* function becomes considerably more computationally expensive.

The expected value of a Beta distributed variable is defined as

$$E(x) = \mu = \frac{\alpha}{\alpha + \beta}, \tag{2.8}$$

and the variance as

$$\mathrm{var}(x) = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta) + 1}. \tag{2.9}$$

Figure 2.3 shows the Beta distribution for different values of α and β. When α and β are both equal to 1, the distribution approximates a uniform distribution (black). When α and β are equal and $\alpha + \beta$ is large enough, the distribution approximates a normal distribution (red). When α is greater than β, the distribution is skewed to the right (purple) and *vice versa*. When α and β are between 0 and 1, the distribution resembles a U-shaped distribution where the outcomes are most likely to occur at the extremes of the range (blue). When α is less than 1 and β is greater than 1, the distribution resembles an exponential distribution (yellow). The range of the Beta distribution can be extended from (0,1) to any range by means of a multiplication factor, making it suitable to problems of any range.

2.3 Covariance

Variance is the measure of a variable's variability. Covariance is the measure of the joint variability of two random variables and provides an indication of the relationship between the variables. A positive covariance indicates that an increase in one variable will result in an increase in the other, as shown in the left-hand graph of Figure 2.4.

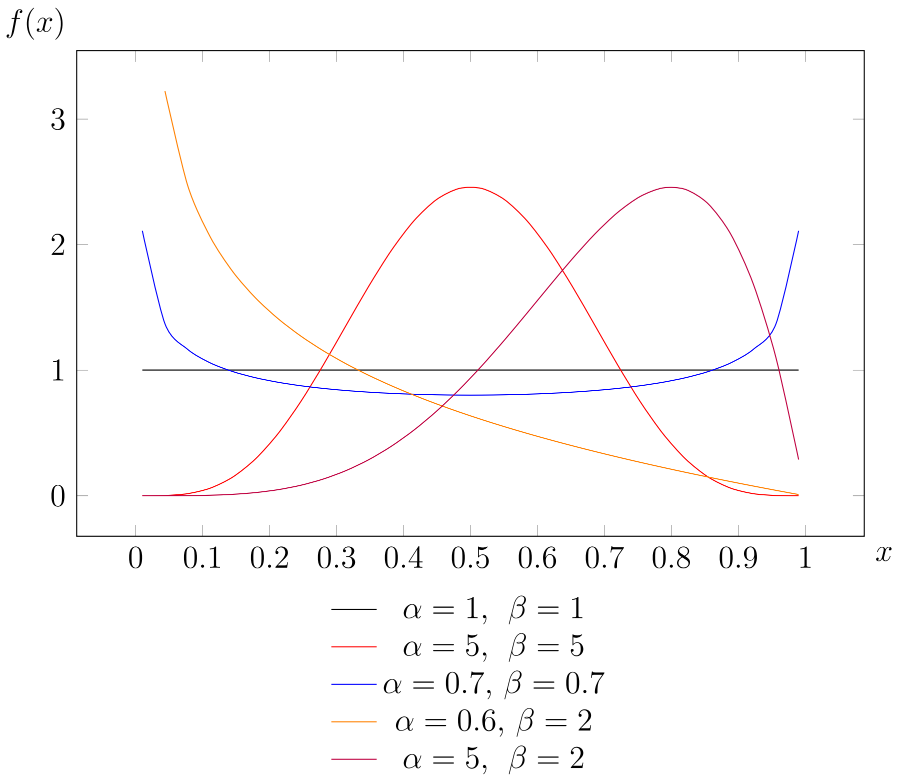


Figure 2.3: The Beta distribution for different values of α and β

Conversely, for variables with a negative covariance, one variable decreases as the other increases, depicted by the centre graph of Figure 2.4. A covariance of approximately 0 indicates that there is a negligible relationship between the variables (right-hand graph of Figure 2.4)(Rice, 2007). It is important to note that covariance does not have a standard unit, but is rather an indication of the relationship between variables.

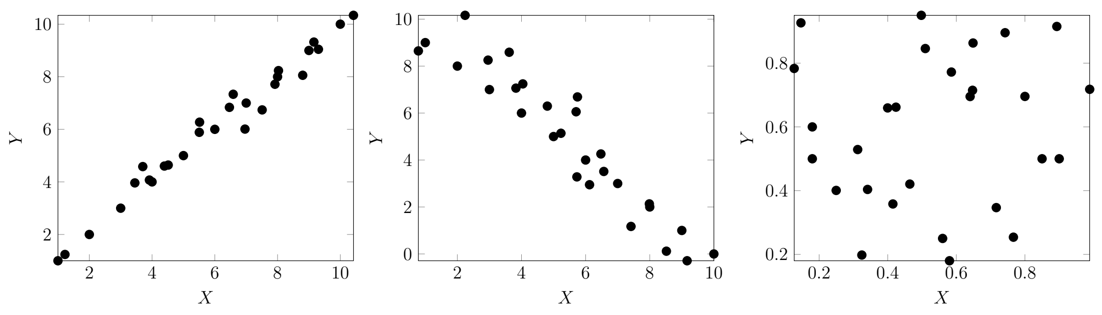


Figure 2.4: The left-hand graph shows two variables X and Y with a positive covariance, the middle graph shows variables X and Y with a negative covariance and the right-hand graph shows variables X and Y with a covariance of approximately 0

The covariance of two random variables X and Y is described by

$$\text{Cov}(X,Y) = E(X-\mu_x)(Y-\mu_y), \tag{2.10}$$

where μ_x and μ_y are the expected values (or means) of random variables X and Y respectively.

Alternatively, covariance can be expressed as

$$\begin{aligned}\text{Cov}(X,Y) &= E(XY - X\mu_y - Y\mu_x + \mu_x\mu_y)\\ &= E(XY) - E(X)\mu_y - E(Y)\mu_x + \mu_x\mu_y\\ &= E(XY) - E(X)E(Y).\end{aligned} \tag{2.11}$$

If variables X and Y are independent, $E(XY) = E(X)E(Y)$ and the covariance is 0 (Rice, 2007).

The covariance matrix is defined in terms of covariance as

$$Q = \begin{bmatrix} {\sigma_x}^2 & \sigma_{yx} \\ \sigma_{xy} & {\sigma_y}^2 \end{bmatrix} \tag{2.12}$$

where ${\sigma_x}^2$ and ${\sigma_y}^2$ are the variances of variables X and Y respectively.

Taking the relationship between variables into consideration could provide an alternative method of sampling. As an example, an adaptation of the well-known inventory problem (s, S) is considered. The inventory problem has two objectives:

maximise service level and minimise average inventory cost. The problem also has two decision variables: reorder point (s) and reorder quantity (S). The problem is modified to be dynamic and stochastic for the purposes of this example.

Figure 2.5 shows the two decision variables (in this example reorder point and reorder quantity) on the x and y axes respectively. The probability density function of each variable is shown on the z axis. The red area shows where points from the two distributions overlap (the darker, the more dense). The stochastic decision variable x (reorder point) has a normal distribution characterised by: $\mu_x = 1$ and $\sigma_x = 0.5$. The stochastic decision variable y (reorder quantity) has a normal distribution characterised by: $\mu_y = 2$ and $\sigma_y = 1$.

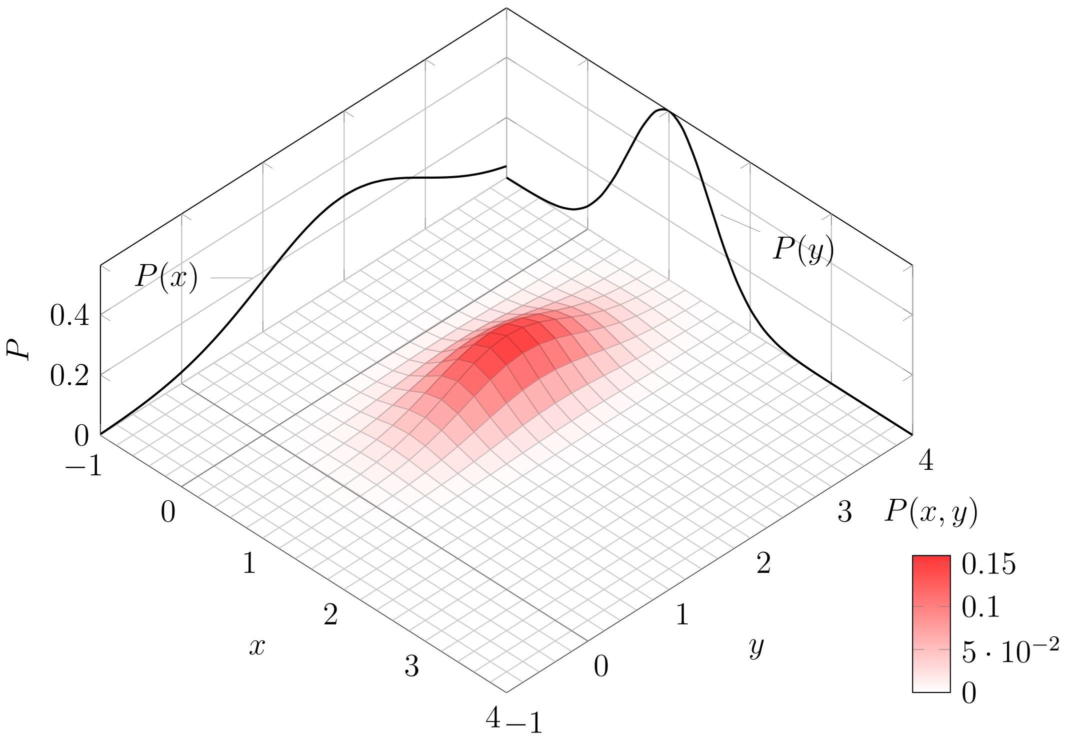


Figure 2.5: The probability density functions of two decision variables (x and y), each with their own normal distribution

Rather than considering the decision variables (reorder point and reorder quantity) individually, perhaps the relationship between the decision variables could be leveraged to decrease the computational time required to determine the

Pareto-optimal solution set, as well as create a better final Pareto solution set. This decrease could be the result of more efficient sampling.

Covariance has been applied to the field of MOO. One of the best known covariance applications to MOO is the study by Igel *et al.* (2007): Covariance Matrix Adaptation for Multi-objective Optimization. Igel *et al.* (2007) extends the covariance matrix adaptation evolution strategy (CMA-ES) to MOO and develops a new variant of the CMA-ES algorithm to solve MOO problems: MO-CMA-ES. The MO-CMA-ES algorithm exploits the properties of covariance, such as its invariance properties, in particular invariance against rotation.

The MO-CMA-ES algorithm is depicted in Algorithm 4, which relies on algorithms 5 and 6 for step size and covariance calculations.

Algorithm 4 (1+λ)-CMA-ES

```
1:  g=0, initialize $a^{(g)}_{\text{parent}}$
2:  **repeat**
3:      $a^{(g+1)}_{\text{parent}} \leftarrow a^{(g)}_{\text{parent}}$
4:      **for** k=1,...,$\lambda$ **do**
5:          $x^{(g+1)}_{k} \sim \mathcal{N}(x^{(g)}_{\text{parent}}, {\sigma^{(g)}}^2 C^{(g)})$
6:      **end for**
7:      updateStepSize$\left(a^{(g+1)}_{\text{parent}}, \frac{\lambda^{(g+1)}_{\text{succ}}}{\lambda}\right)$
8:      **if** $f(x^{(g+1)}_{1:\lambda}) \leq f(x^{(g)}_{\text{parent}})$ **then**
9:          $x^{(g+1)}_{\text{parent}} \leftarrow x^{(g+1)}_{1:\lambda}$
10:         updateCovariance$\left(a^{(g+1)}_{\text{parent}}, \frac{x^{(g+1)}_{\text{parent}} - x^{(g)}_{\text{parent}}}{\sigma^{(g)}_{\text{parent}}}\right)$
11:     **end if**
12:     $g \leftarrow g + 1$
13: **until** stopping criterion is met
```

λ refers to the number of new candidate solutions samples in each iteration. After the new candidate solutions are sampled, the step size is updated based on the success rate $p_{succ} = \frac{\lambda^{g+1}_{succ}}{\lambda}$.

Algorithm 5 updateStepSize ($a = [x, \bar{p}_{\text{succ}}, \sigma, p_c, C], p_{\text{succ}}$)

```
1: $\bar{p}_{\text{succ}} \leftarrow (1 - c_p)\bar{p}_{\text{succ}} + c_p p_{\text{succ}}$
2: $\sigma \leftarrow \sigma \cdot exp\left(\frac{1}{d} \frac{\bar{p}_{\text{succ}} - p^{\text{target}}_{\text{succ}}}{1 - p^{\text{target}}_{\text{succ}}}\right)$
```

The learning rate c_p should be set to a value in the range $0 - 1$. If the best new candidate solution is better than the parent solution, the covariance matrix is updated by means of Algorithm 6.

Algorithm 6 updateCovariance $(a = [x, \bar{p}_{\text{succ}}, \sigma, p_c, C], x_{\text{step}} \in \mathbb{R}^n)$

```
1: if $\bar{p}_{\text{succ}} < p_{\text{thresh}}$ then
2:     $p_c \leftarrow (1 - c_c)p_c + \sqrt{c_c(2 - c_c)}x_{\text{step}}$
3:     $C \leftarrow (1 - c_{\text{cov}})C + c_{\text{cov}} \cdot p_c p_c^T$
4: else
5:     $p_c \leftarrow (1 - c_c)p_c$
6:     $C \leftarrow (1 - c_{\text{cov}})C + c_{\text{cov}} \cdot (p_c p_c^T + c_c(2 - c_c)C)$
7: end if
```

The default selection, step size and covariance parameters are set according to the values in Table 2.1.

Table 2.1: MO-CMA-ES default parameters

Selection:		
$\lambda = 1$		
Step size control:		
$d = 1 + \frac{n}{2\lambda}$,	$p_{\text{succ}}^{\text{target}} = \frac{1}{5+\sqrt{\lambda/2}}$,	$c_p = \frac{p_{\text{succ}}^{\text{target}}\lambda}{2+p_{\text{succ}}^{\text{target}}\lambda}$
Covariance matrix adaptation:		
$c_c = \frac{2}{n+2}$,	$c_{\text{cov}} = \frac{2}{n^2+6}$,	$p_{\text{thresh}} = 0.44$

Igel *et al.* (2007) compare the MO-CMA-ES to two other non-dominated sorting algorithms namely: Non-dominated Sorting Genetic Algorithm II (NSGA-II) and Non-dominated Sorting Differential Evolution (NSDE) for a number of test problems. MO-CMA-ES outperforms NSGA-II for all test problems except one, and performs better than NSDE on all test problems but two (where NSDE performs worse than the other two algorithms on all other test problems). The results indicate that using the covariance matrix adaptation as a new selection mechanism, significantly improves the search efficiency of the MO-CMA-ES algorithms, making it superior to the NSGA-II and NSDE algorithms.

When compared to other non-dominated sorting algorithms, the promising results of the MO-CMA-ES algorithm could be an indication that a similar covariance matrix adaption could improve the search efficiency of other MOO algorithms, such as MOO CEM.

2.4 Constrained MOPs

At a high level, two types of MOO problems exist: constrained and unconstrained problems. (1.2) and (1.3) represent the constraints which apply to an unconstrained problem formulated by (1.1). Constraints add a dimension of complexity to an MOO problem, as they limit the feasible region of the decision variables (Coello, 2006). Additional challenges arise from the interference among constraints and the relationships among decision variables, constraints and the objective functions (Woldesenbet *et al.*, 2007).

Many different methods have been developed to solve constrained MOPs, one of the most simple methods being the removal of infeasible solutions post-processing. Once all points have been sampled, the constraints are applied and those solutions which do not fall within the feasible region are simply discarded. This method may be simple, but could result in the removal of too many points, leaving the algorithm with little direction for the next iteration and, ultimately, a poor Pareto solution set (Coello, 2006).

Other methods include (Coello, 2002):

- Penalty functions
- Special representations and operators
- Repair algorithms
- Separation of objectives and constraints
- Hybrid methods

This research paper will focus specifically on the Penalty Function method, as this is a well-researched and widely accepted constraint-handling method.

The main aim of this method is to transform constrained optimisation problems into unconstrained optimisation problems by adding (for a minimisation problem) or subtracting (for a maximisation problem) a specific amount to or from the objective function. This amount is referred to as the penalty. The size of the penalty is dependent on the degree of the constraint violation (Coello, 2002). For a minimisation problem, a good solution would have a low objective function value. However, if the solution is far from the feasible region, a large penalty would be added to the objective function, resulting in a poor solution.

The challenge with the Penalty Function method is determining the size of the penalty. If the penalty is too large, the algorithm will be directed to the feasible region only, limiting exploration. If the optimal solution were to lie on the boundary between the feasible and infeasible region, the algorithm would most likely not find the optimal solution. On the other hand, if the penalty is too small, many infeasible solutions would be considered and much time would be spent exploring the infeasible region. This is where the *minimum penalty rule* provides direction. The *minimum penalty rule* states that the penalty value should be kept as small as possible, "just above the limit below which infeasible solutions are optimal" (Coello, 2002).

Penalty functions can be constructed using three different techniques (Coello, 2002):

- penalise all infeasible solutions equally, irrespective of the distance from the feasible region,
- penalise solutions according to the 'size' of their infeasibility, or
- penalise solutions according to the cost required to transform the solution into a feasible solution.

Several studies have been focused on which of these methods prove to be the most effective. A study by Richardson *et al.* (1989) found that penalties which take into consideration the size of the infeasibility show better performance than those which only consider the number of violated constraints.

Woldesenbet *et al.* (2007) proposed a constraint-handling multi-objective evolutionary optimisation algorithm which uses the Penalty Method. The algorithm extends the single objective constraint-handling evolutionary algorithm developed by Tessema & Yen (2006) to multi-objective optimisation problems. The algorithm developed by Woldesenbet *et al.* (2007), the Constrained Multi-objective Evolutionary Algorithm (CMOEA), calculates the penalty according to the size of the solution's constraint violation, defined as

$$F_i(x) = d_i(x) + p_i(x). \tag{2.13}$$

The penalty has two components: a distance measure $d_i(x)$ and an adaptive penalty $p_i(x)$. The distance measure is calculated using Algorithm 7, based on

$$d_i(x) = \begin{cases} v(x), & \text{if } r_f = 0 \\ \sqrt{\tilde{f}_i(x)^2 + v(x)^2}, & \text{otherwise} \end{cases} \tag{2.14}$$

where

$$r_f = \frac{\text{number of feasible solutions in the current population}}{\text{population size}}. \tag{2.15}$$

r_f represents the percentage of feasible solutions in the current population. If there are no feasible solutions in the current population, the distance value is the constraint violation $v(x)$. However, if some feasible solutions exist, the normalised objective function and constraint violation are used to calculate the distance. If a solution is feasible, the distance is simply the normalised objective function.

The constraint violation $v(x)$ is the normalised violation of each constraint and is calculated according to

$$v(x) = \frac{1}{m}\sum_{j=1}^{m} \frac{c_j(x)}{c^j_{\max}}, \tag{2.16}$$

where

$$c_j(x) = \begin{cases} \max(0, g_j(x)) & j = 1, \ldots, q \\ \max(0, h_l(x) - \delta) & l = 1, \ldots, p \end{cases}$$

and

$$c_{\max}^j = \max_x c_j(x).$$

If a solution violates a constraint, c_j is the value (or size) of the $j-$th equality (or $l-$th inequality) constraint violation. If the constraint is not violated, c_j is simply 0. $c_{\max}^j$ is the maximum possible violation of the $j-$th (or $l-$th) constraint, which allows for the normalisation according to (2.16). $g_j(x)$ and $h_l(x)$ are the equality and inequality constraints as per (1.2) and (1.3) in the MOO problem definition. The equality constraints are converted to inequality constraints by the addition of the tolerance value δ.

CMOEA requires each solution x for each objective i to be normalised according to (2.17), where $\tilde{f}_i(x)$ is the normalised objective value of i and $f_{\min}^i$ and $f_{\max}^i$ refer to the minimum and maximum values of objective i respectively.

$$\tilde{f}_i(x) = \frac{f_i(x) - f_{\min}^i}{f_{\max}^i - f_{\min}^i}, \tag{2.17}$$

where

$$f_{\min}^i = \min_x f_i(x) \tag{2.18}$$

and

$$f_{\max}^i = \max_x f_i(x). \tag{2.19}$$

The penalty value $p_i(x)$ is calculated according to Algorithm 8, based on

$$p_i(x) = (1 - r_f)X_i(x) + r_f Y_i(x), \tag{2.20}$$

where

$$X_i(x) = \begin{cases} 0, & \text{if } r_f = 0 \\ v(x), & \text{otherwise} \end{cases}$$

and

$$Y_i(x) = \begin{cases} 0, & \text{if x is a feasible solution} \\ \tilde{f}_i(x), & \text{if x is an infeasible solution.} \end{cases}$$

The penalty value $p_i(x)$ consists of two penalty values: X_i, which is based on the objective value, and Y_i, which is based on the constraint violation.

Algorithm 7 Distance Measure of the constraint-handling multi-objective evolutionary optimisation algorithm

```
1:  if $r_f = 0$ then
2:      for $i = 1$ to $number\_of\_objectives$ do
3:          for $k = 1$ to $Population\_Size$ do
4:              $d_i(x_k) \leftarrow v(x_k)$
5:          end for
6:      end for
7:  else
8:      for  $i = 1$ to $number\_of\_objectives$ do
9:          for $k = 1$ to $Population\_Size$ do
10:             $\tilde{f}_i(x_k) \leftarrow \frac{f_i(x_k) - f^i_{\min}}{f^i_{\max} - f^i_{\min}}$
11:             $d_i(x_k) \leftarrow \sqrt{\tilde{f}_i(x_k)^2 + v(x_k)^2}$
12:         end for
13:     end for
14: end if
```

Algorithm 8 Penalty Value of the constraint-handling multi-objective evolutionary optimisation algorithm

```
1:  for $i = 1$ to $number\_of\_objectives$ do
2:      for  $k = 1$ to $Population\_Size$ do
3:          if $r_f = 0$ then
4:              $X_i(x_k) \leftarrow 0$
5:          else
6:              $X_i(x_k) \leftarrow v(x_k)$
7:          end if
8:          if $v(x_k) = 0$ then
9:              $Y_i(x_k) \leftarrow 0$
10:         else
11:             $Y_i(x_k) \leftarrow \tilde{f}_i(x_k)$
12:         end if
13:         $p_i(x_k) \leftarrow (1 - r_f)X_i(x_k + r_f Y_i(x_k))$
14:     end for
15: end for
```

Woldesenbet *et al.* (2007) compared the CMOEA algorithm to the NSGA-II and Ray-Tai-Seow algorithms for 14 different benchmark problems, using hypervolume as the performance indicator. Results indicated that the CMOEA algorithm performs better, providing a well-distributed, consistent Pareto front for all test problems.

2.5 Non-dominated sorting algorithms

In the field of MOO, finding the Pareto-optimal front in the shortest amount of time is crucial. Many sorting algorithms have been developed, one of the most well-known algorithms being the Goldberg (1989) Pareto-ranking algorithm.

As mentioned in Subsection 2.1.4, the MOO CEM algorithm uses the Goldberg (1989) algorithm as a means of identifying the Pareto-optimal solution set. The Goldberg (1989) method is implemented in such a manner that only problems with two objective functions can be solved, according to Algorithm 3. This method has a complexity of $O(KN^2)$, where N represents the population size. It would be useful to extend this functionality such that problems with more than two objective functions could be solved in a more efficient manner.

Since the Goldberg (1989) algorithm was published in 1989, a number of faster sorting algorithms have been developed, including: Deductive Sort (McClymont & Keedwell, 2012), Corner Sort (Wang & Yao, 2014), Efficient Non-Dominated Sort (ENS) (Zhang *et al.*, 2015), Best Order Sort (BOS) (Roy *et al.*, 2016) and Merge Non-Dominated Sorting Algorithm for Many-Objective Optimization (MNDS) (Moreno *et al.*, 2020). The best and worst case time complexities of each algorithm are stated in Table 2.2.

Figure 2.6 shows the runtime per algorithm for different non-dominated sorting algorithms as per the experiments performed by Moreno *et al.* (2020). BOS and MNDS are the most recently developed algorithms and have proven to be extremely effective in higher dimensions. ENS, on the other hand, performs better at lower dimensions (fewer objectives than 5). Since one of the objectives of this research paper is to add the functionality to solve problems with more than two objectives to the MOO CEM algorithm, ENS has been selected as a suitable non-dominated sorting algorithm for this use case and will be discussed in detail.

Table 2.2: Best and worst case time complexities of selected ranking algorithms

Algorithm	Best Case Complexity	Worst Case Complexity
Goldberg	$\mathcal{O}(KN^2)$	$\mathcal{O}(KN^2)$
Deductive Sort	$\mathcal{O}(KN\sqrt{N})$	$\mathcal{O}(KN^2)$
Corner Sort	$\mathcal{O}(KN\sqrt{N})$	$\mathcal{O}(KN^2)$
Efficient Non-dominated Sort (sequential search strategy)	$\mathcal{O}(KN\sqrt{N})$	$\mathcal{O}(KN^2)$
Efficient Non-dominated Sort (binary search strategy)	$\mathcal{O}(KN\log N)$	$\mathcal{O}(KN^2)$
Best Order Sort	$\mathcal{O}(KN\log N)$	$\mathcal{O}(KN\log N + KN^2)$
Merge Non-Dominated Sorting Algorithm for Many-Objective Optimization	$\mathcal{O}(N\log N)$	$\mathcal{O}(KN^2)$

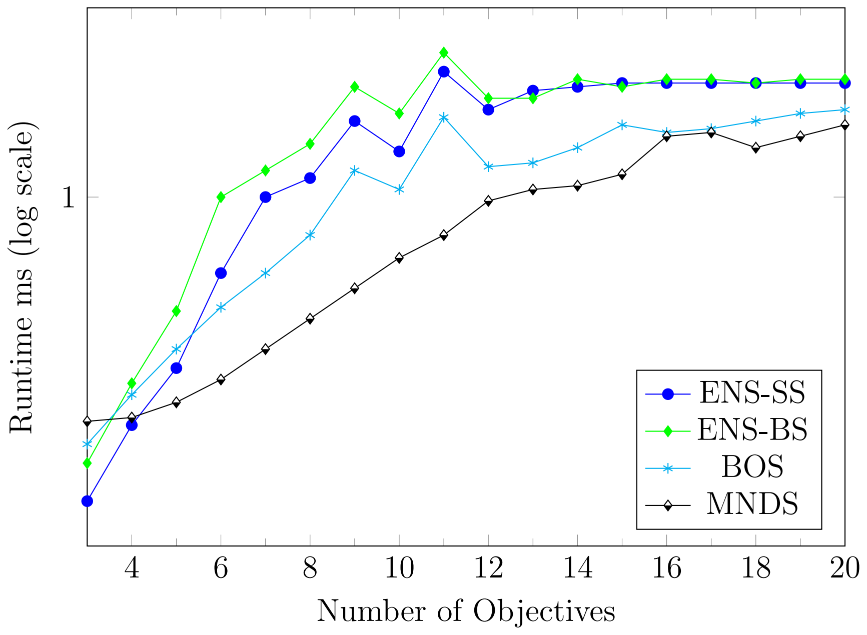


Figure 2.6: The runtime of various non-dominated sorting algorithms for 500 solutions (Moreno *et al.*, 2020)

The difference between ENS and many other non-dominated sorting algorithms is that rather than comparing each solution to all other solutions before assigning it to a front, the ENS algorithm compares each solution only to those which have already been assigned to a front. This is achieved by sorting the population according to the first objective before the algorithm is applied. This limits the number of required comparisons, which makes the algorithm more computationally efficient than many others. The main ENS algorithm is shown in Algorithm 9 (Zhang *et al.*, 2015).

Algorithm 9 Efficient Non-dominated Sort algorithm

```
1: **Input**: population $P$
2: **Output**: the set of fronts $F$
3: $F$ = empty;
4: Sort $P$ in an ascending order of the first objective value;
5: **for all** $P[n] \in$ sorted $P$ **do**
6:     Assign solution $P[n]$ into $F$ using the Sequential Search Strategy (Algorithm 10) or the Binary Search Strategy;
7: **end for**
8: **return** $F$;
```

ENS has been developed with two different search strategies: sequential and binary search strategies. The results by Zhang *et al.* (2015) showed that the sequential search strategy had a shorter runtime than the binary search strategy for problems with more than two objectives. Since the focus of the ENS algorithm implementation will be to solve problems with more than two objectives, only the sequential search strategy will be explored.

The Sequential Search Strategy (SS) used in the ENS algorithm is presented in Algorithm 10. For a solution p_n, SS checks whether a solution which dominates p_n exists in the first Pareto set. If not, solution p_n is assigned to the first Pareto set. Otherwise, p_n is assigned to the second Pareto set. The same check is then applied to the second Pareto set and this process is repeated until p_n is assigned to a Pareto set (existing or new).

2.6 Performance indicators and standard problems

In order to determine the quality of an algorithm, the performance of the algorithm must be assessed on a set of standard problems using performance indicators.

Algorithm 10 Sequential Search Strategy for finding the front of a solution used by Efficient Non-dominated Sort algorithm

```
1: Input: solution $P[n]$, the set of fronts $F$.
2: Output: the front number of solution $P[n]$.
3: $x = \text{size}(F)$; the number of fronts having been found
4: $k = 1$; the front now checked
5: while true do
6:     Compare $P[n]$ with the solutions in $F[k]$ starting from the last one and
       ending with the first one;
7:     if $F[k]$ contains no solution dominating $P[n]$ then
8:         return $k$; move $P[n]$ to $F[k]$
9:         break;
10:    else
11:        $k++$
12:        if $k > x$ then
13:            return $x + 1$; {move $P[n]$ to a new front}
14:            break;
15:        end if
16:    end if
17: end while
```

2.6.1 Performance indicators

A number of different performance indicators have been developed and can be categorised into four main groups (Jiang *et al.*, 2014):

- Capacity metrics,
  - Overall Non-dominated Vector Generation (ONVG)
  - Error Ratio
- Convergence metrics,
  - Generational Distance (GD)
  - ϵ indicator
  - Seven Points Average Distance (SPAD)

- Diversity metrics
  - Uniform Distribution (UD)
  - Overall Pareto Spread (OS)
  - Generalized Spread
- Convergence-Diversity metrics
  - Hypervolume (HV) (also referred to as the S-metric or the Lebesgue measure)
  - Inverted Generational Distance (IGD)
  - Maximum Pareto Front Error (MPFE)

Capacity metrics count the number of non-dominated solutions that satisfy predetermined requirements. Convergence metrics measure the proximity of the solution set to the true Pareto-optimal front. Diversity metrics include distribution and spread information. Distribution refers to how evenly points in a solution are spaced. Spread refers to how well the points in the solution set capture the true extremes. And lastly, Convergence-Diversity metrics measure both convergence and diversity of the solution set on a single scale (Jiang *et al.*, 2014).

Some of these indicators are classified as *Pareto compliant*, while others are *Pareto non-compliant*. The term *Pareto compliant* is formally defined as: An indicator $I\colon \Omega \rightarrow \mathbb{R}$ *is Pareto compliant if for all* $A, B \in \Omega : A \preceq B \Rightarrow I(A) \geq I(B)$, *assuming that greater indicator values correspond to higher quality (otherwise* $A \preceq B \Rightarrow I(A) \leq I(B)$*)* (Coello *et al.*, 2007). In essence, when comparing two solution sets using a Pareto quality indicator, "the quality indicator value for A should be at least as good as the indicator value for B, with respect to weak Pareto dominance" (Bekker, 2012). This implies that Pareto-compliant performance indicators are more reliable than Pareto non-compliant indicators with respect to algorithm solution set comparison. Table 2.3 indicates Pareto compliance or not of some popular performance indicators.

A study by Riquelme *et al.* (2015) found that the hypervolume indicator was the most widely used metric, followed by the generational distance, the ϵ indicator and the inverted generational distance.

Table 2.3: Pareto compliance of some performance indicators (Coello *et al.*, 2007)

Metric	Pareto Compliant
Hyperarea	Yes
ϵ indicator	Yes
Generational Distance	No
ONVG	Yes
Error Ratio	Yes
MPFE	No

The study by Jiang *et al.* (2014) concluded that the hypervolume indicator (I_h) is the most robust and reliable metric when comparing the quality of solution sets. Since hypervolume takes into account both closeness and spread of the solution set, it is a favoured solution amongst developers and researchers (While, 2005). Considering that the hypervolume indicator is the most widely accepted metric, the hypervolume will be the main indicator used to compare algorithm solutions in this study, followed by the ϵ indicator where the hypervolume cannot be calculated (since the GD is Pareto non-compliant).

The hypervolume (or hyperarea for problems in two-dimensional space) indicator was proposed as a performance indicator by Zitzler (1999). Essentially, hypervolume measures the difference in volume (or area) between a set of Pareto-optimal solutions found by an algorithm compared to the hypervolume of the true Pareto-optimal solution set. It relies on a reference point outside of the maximum of the objective function solution space. Figure 2.7 shows the hyperarea of a solution set of a minimisation problem with two dimensions. The area between the Pareto-optimal solution set (red) and the reference point (green) is the hyperarea. It is calculated as

$$I_H \triangleq \bigcup_i \text{area}_i | \text{vec}_i \in \text{PF}_{\text{known}}, \tag{2.21}$$

where vec_i represents a non-dominated vector in the known (or true) Pareto front (PF_{known}). The hypervolume is calculated similarly: the volume is used instead of the area (Coello *et al.*, 2007).

Solutions with a larger hyperarea indicate a greater distance from the reference point. This implies that the solutions are smaller, and thus, better, in the case of a minimisation problem. Therefore, the hyperarea should be maximised.

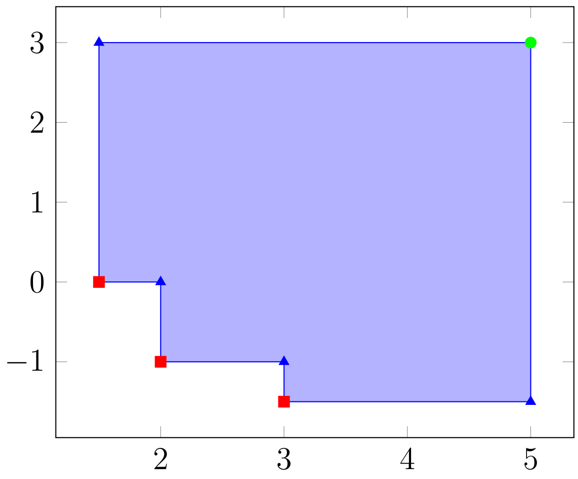


Figure 2.7: Hyperarea of a minimisation problem with two objectives (Pareto front is indicated by red squares and the reference point in green)

In certain cases, the hypervolume cannot be calculated intuitively due to the computational complexity. When the hypervolume cannot be computed, the ϵ indicator will be used as the performance indicator instead.

The ϵ indicator represents the minimum factor by which the approximation set must be translated to (weakly) dominate the true Pareto solution set. Two different ϵ indicators exist: an additive ϵ indicator ($I_{\epsilon+}$), based on the difference between the approximation set and the true Pareto solution set, and the multiplicative ϵ indicator ($I_{\epsilon\times}$), which is the ratio between the approximation set and the true Pareto solution set (Liefooghe & Derbel, 2016). The additive ϵ indicator is used in this study. It is calculated as

$$I_{\epsilon+} = \max_{r\in R} \min_{a\in A} \max_{i\in\{1,\ldots,d\}} (a_i - r_i). \tag{2.22}$$

As the ϵ indicator represents the difference between the approximation set and the true Pareto solution set, this indicator is to be minimised. When $I_{\epsilon+} = 0$, it implies that the approximation set and the true Pareto solution set consist of the same solutions. A small $I_{\epsilon+}$ (close to 0) implies a good approximation set which is close to the true Pareto solution set.

2.6.2 Standard problems

Since the main objective of this research paper is the enhancement of the MOO CEM algorithm, the enhanced algorithm must be tested on the same standard problems as those on which MOO CEM was tested for fair comparison. The MOO CEM algorithm was tested (amongst others) on the standard MOP testing problems (MOP1, MOP2, MOP3, MOP4 and MOP6) listed in Table 2.4. MOP5 and MOP7 are problems with more than two objectives and could not be solved with the original MOO CEM algorithm, due to the non-dominated sorting algorithm implementation. Table 2.5 contains a set of standard side-constraint test functions which can be used to test the performance of algorithms on problems with side-constraints (Coello, 2002).

The original MOO CEM algorithm and the enhanced algorithms will be tested on the problems shown in the tables. The results of these tests are analysed in Chapter 4.

Table 2.4: Standard MOP test problems

Function	Definition			Constraints
MOP1	$f_1(x)$	$=$	x^2	$-10^5 \le x \le 10^5$
(Min)	$f_2(x)$	$=$	$(x-2)^2$	
MOP2	$f_1(\mathbf{x})$	$=$	$1-\exp(-\sum_{i=1}^{n}(x_i-\frac{1}{\sqrt{n}})^2)$	$-4 \le x_i \le 4$
(Min)	$f_2(\mathbf{x})$	$=$	$1-\exp(-\sum_{i=1}^{n}(x_i+\frac{1}{\sqrt{n}})^2)$	$i=1,\ldots,n,\ n=3$
MOP3	$f_1(x,y)$	$=$	$-[1+(A_1-B_1)^2+(A_2-B_2)^2]$	$-\pi \le x,y \le \pi$
(Max)	$f_2(x,y)$	$=$	$-[(x+3)^2+(y+1)^2]$	$A_1=0.5\sin 1-2\cos 1+$
				$\sin 2-1.5\cos 2,$
				$A_2=1.5\sin 1-\cos 1+$
				$2\sin 2-0.5\cos 2,$
				$B_1=0.5\sin x-2\cos x+$
				$\sin y-1.5\cos y$
				$B_2=1.5\sin x-\cos x+$
				$2\sin y-0.5\cos y$
MOP4	$f_1(\mathbf{x})$	$=$	$\sum_{i=1}^{n-1}(-10\exp(-0.2)\sqrt{x_i^2+x_{i+1}^2}),$	$-5 \le x_i \le 5$
(Min)	$f_2(\mathbf{x})$	$=$	$\sum_{i=1}^{n}(\lvert x_i\rvert^a+5\sin(x_i)^b)$	$i=1,2,3,\ a=0.8,\ b=3$
MOP5	$f_1(\mathbf{x})$	$=$	$0.5*(x^2+y^2)+\sin(x^2+y^2),$	$-30 \le x,y \le 30$
(Min)	$f_2(\mathbf{x})$	$=$	$=\frac{(3x-2y+4)^2}{8}+\frac{(x-y+1)^2}{27}+15$	
	$f_3(\mathbf{x})$	$=$	$=\frac{1}{(x^2+y^2+1)}-1.1e^{-x^2-y^2}$	
MOP6	$f_1(x,y)$	$=$	x	$0 \le x,y \le 1$
(Min)	$f_2(x,y)$	$=$	$(1+10y)[1-(\frac{x}{1+10y})^\alpha-\frac{x}{1+10y}\sin(2\pi qx)]$	$q=6,\ \alpha=2$
MOP7	$f_1(\mathbf{x})$	$=$	$\frac{(x-2)^2}{2}+\frac{(y+1)^2}{13}+3,$	$-400 \le x,y \le 400$
(Min)	$f_2(\mathbf{x})$	$=$	$=\frac{(x-y-3)^2}{36}+\frac{(-x+y+2)^2}{8}-17$	
	$f_3(\mathbf{x})$	$=$	$=\frac{(x+2y-1)^2}{175}+\frac{(2y-x)^2}{17}-13$	
ZDT1	$f_1(\mathbf{x})$	$=$	x_1	$0 \le x_i \le 1,\ n=30$
(Min)	$f_2(\mathbf{x},g)$	$=$	$g(\mathbf{x})\cdot(1-\sqrt{f_1/g(\mathbf{x})})$	
	$g(\mathbf{x})$	$=$	$1+\frac{9}{n-1}\cdot\sum_{i=2}^{n}x_i$	
ZDT2	$f_1(\mathbf{x})$	$=$	x_1	$0 \le x_i \le 1,\ n=30$
(Min)	$f_2(\mathbf{x},g)$	$=$	$g(\mathbf{x})\cdot(1-(f_1/g(\mathbf{x}))^2)$	
	$g(\mathbf{x})$	$=$	$1+\frac{9}{n-1}\cdot\sum_{i=2}^{n}x_i$	
ZDT3	$f_1(\mathbf{x})$	$=$	x_1	$0 \le x_i \le 1,\ n=30$
(Min)	$f_2(\mathbf{x},g)$	$=$	$g(\mathbf{x})\cdot(1-\sqrt{f_1/g(\mathbf{x})}-f_1/g(\mathbf{x})\cdot\sin(10\pi f_1))$	
	$g(\mathbf{x})$	$=$	$1+\frac{9}{n-1}\cdot\sum_{i=2}^{n}x_i$	

Table 2.5: Standard constrained MOP test problems

Function	Definition			Constraints
MOP-C1 Binh(2)	$f_1(x,y)$	$=$	$4x^2+4y^2$	$0\leq x,y\leq 5$,
(Min)	$f_2(x,y)$	$=$	$(x-5)^2+(y-5)^2$	$0\geq (x-5)^2+y^2-25$,
				$0\geq -(x-8)^2+(y+3)^2+7.7$
MOP-C2 Osyczka(2)	$f_1(x,y)$	$=$	$-(25((x_1-2)^2+(x_2-2)^2+$	$0\leq x_1,x_2,x_6\leq 10$,
(Min)			$(x_3-1)^2+(x_4-4)^2+(x_5-1)^2)$	$1\leq x_3,x_5\leq 5$,
	$f_2(x,y)$	$=$	$x_1^2+x_2^2+x_3^2+x_4^2+x_5^2+x_6^2$	$0\leq x_4\leq 6$,
				$0\geq x_1+x_2-2$,
				$0\geq 6-x_1-x_2$,
				$0\geq 2-x_2+x_1$,
				$0\geq 2-x_1+3x_2$,
				$0\geq 4-(x_3-3)^2-x_4$,
				$0\geq (x_5-3)^2+x_6-4$
MOP-C3 Viennet(4)	$f_1(x,y)$	$=$	$\frac{(x-2)^2}{2}+\frac{(y+1)^2}{13}+3$	$-4\leq x,y\leq 4$,
(Min)	$f_2(x,y)$	$=$	$\frac{(x+y-3)^2}{175}+\frac{(2y-x)^2}{17}-13$	$y<-4x+4$,
				$x>-1$,
	$f_3(x,y)$	$=$	$\frac{(3x-2y+4)^2}{8}+\frac{(x-y+1)^2}{27}+15$	$y>x-2$,
MOP-C4 Tanaka	$f_1(x,y)$	$=$	x	$0\leq x,y\leq \pi$,
(Min)	$f_2(x,y)$	$=$	y	$0\geq -(x^2)-(y^2)+1+$
				$(a\cos(b\arctan(x/y)))$,
				$a=0.1,\ b=16$

2.7 Summary: Chapter 2

An overview of the literature relating to MOO, the MOO CEM algorithm and the four methods of enhancements was presented in this chapter. The aim was not to provide a comprehensive analysis, but rather to provide an overview of the field of MOO, the design of MOO CEM and the four areas of enhancement, with specific focus on those pertinent to this research. The most widely used performance indicators and benchmark problems were listed, as these will be of importance when testing the new proposed algorithms.

In the next chapter, four new algorithms will be developed to address the four areas of improvement and enhancement. The methods of enhancement have been selected in accordance with the research presented in this chapter.

Chapter 3

Algorithm development

In the previous chapter, an analysis and review of the fields pertaining to this study were presented. This includes at the centre, the MOO CEM algorithm, and the methods, algorithms and equations selected to enhance the algorithm. In this chapter, some of the reviewed techniques are implemented in order to enhance the MOO CEM algorithm. The algorithm is enhanced through the improvement of the sampling method, as well as through the incorporation of additional functionality. The two techniques investigated to improve sampling are the Beta distribution and covariance. The functionality of the algorithm is then extended to solve constrained problems and problems with more than two objective functions. Upon testing each of the proposed algorithms, a final algorithm will be included in Chapter 5, consisting of the most successful algorithms proposed in this chapter as implemented in the MOO CEM algorithm.

3.1 Proposed MOO CEM-Beta algorithm

This section investigates improving of the sampling method of the MOO CEM by using the Beta distribution (reviewed in Chapter 2.2), instead of the truncated normal distribution. It is theorised that, by using the Beta distribution, the quality of the Pareto solution set could be improved, i.e. the hypervolume calculated from the final solutions generated by the algorithm would be larger than that generated by the original MOO CEM algorithm. This hypothesis is based on the fact that the Beta distribution is more flexible, in that it can approximate a number of distributions, rather than a single normal distribution. As stated in Section 2.1.4, another disadvantage of the truncated normal distribution is the possibility of overlooking solutions at the extremes of the range. By using the Beta distribution, these solutions could be included and the spread of the Pareto solution set could be increased.

The MOO CEM algorithm (Algorithm 2) was reviewed in Section 2.1.4. The decision space is divided into $r + 2$ classes (where r is the number of classes of the elite vector), with the total of all class frequencies equal to N. In Step 8, a class-based mean and standard deviation are calculated for each histogram class $[c_{i\kappa}\, c_{i(\kappa+1)})$ of the decision space. The mean and standard deviation are then used to sample from the truncated normal distribution ϕ. Algorithm 11 is proposed to replace the truncated normal distribution, from which solutions are sampled, with the Beta distribution. This algorithm replaces steps $8 - 11$ in the original MOO CEM algorithm. This algorithm is executed once per histogram class.

Algorithm 11 MOO CEM-Beta: Algorithm to sample from Beta distribution to be implemented in MOO CEM algorithm

```
 1: Find the Elite solutions which fall into the range $[c_{i\kappa}\, c_{i(\kappa+1)})$.
 2: if 1 or no unique Elite solutions fall within this range then
 3:     Set $\alpha_{i\kappa} = 1$.
 4:     Set $\beta_{i\kappa} = 1$.
 5: else
 6:     Calculate a class based $\alpha_{i\kappa}$ and $\beta_{i\kappa}$ of the distribution of the corresponding
    Elite solutions over the normalised range $0 - 1$.
 7: end if
 8: Generate a subsample $Y$ according to the pdf $Beta\ (\alpha_i,\beta_i)$.
 9: with $\mathbf{x}_i \in [c_{i\kappa}\, c_{i(\kappa+1)})$ and $|\mathbf{Y}| = \tau_{i\kappa}, 1 \leq \kappa \leq r+2$ .
10: Append $\mathbf{Y}$ to $\mathbf{X}_i$.
```

The parameters required by the Beta distribution (α and β) are calculated for each histogram class (a class based α_i and β_i) using the elite solutions which fall within the range of that histogram class. Calculating the Beta distribution parameters requires at least two points. If only one or no unique elite solutions fall within the range of the histogram class, α_i and β_i are both set to 1, assuming a uniform distribution across the range. Building the initial solution set for the first iteration of the algorithm follows the same logic: assigning both α and β a value of 1, so a uniform distribution is assumed across the range.

As the Beta distribution is defined over the range $0 - 1$, the range of the histogram class must be normalised when calculating the values of α_i and β_i. This is achieved using the upper and lower limits of the histogram class.

In the MOO CEM algorithm, the mean and standard deviation are smoothed and a stopping criteria is calculated in steps $20 - 22$. This is replaced with the smoothing of the α and β parameters in a similar fashion, using (2.3). The stopping criteria is calculated using the difference between the values of the α and β for the current iteration and the values thereof in the previous iteration. If α and β do not change materially from one iteration to the next, the algorithm is assumed to have reached steady-state and there would be no benefit in running the algorithm for further iterations. These updated steps are contained in Algorithm 12 and intend to replace steps $20 - 22$ of the MOO CEM algorithm.

Algorithm 12 Algorithm to update α and β in the MOO CEM-Beta algorithm

```
1: Use the values in Elite and compute $\boldsymbol{\alpha}_{it}$ and $\boldsymbol{\beta}_{it}$ for all $i, 1 \leq i \leq D$.
2: Smooth the vectors $\boldsymbol{\alpha}_{it}$ and $\boldsymbol{\beta}_{it}$ using (2.3).
3: Calculate the differences between $\boldsymbol{\alpha}_{it}$ and $\boldsymbol{\alpha}_{it-1}$; and $\boldsymbol{\beta}_{it-1}$ and $\boldsymbol{\beta}_{it}$.
4: If all changes in $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i > \epsilon_c$, or less than the allowable number of evaluations have been done, increment $t$ and reiterate from Step 4 of the original MOO CEM algorithm.
```

3.2 Proposed MOO CEM-Cov algorithm

In this section, the improvement of the MOO CEM algorithm's sampling method through the use of covariance is proposed. The method of using covariance to improve sampling is based on the method developed by Igel *et al.* (2007), as discussed in Section 2.3.

The MOO CEM algorithm assumes independence between decision variables. However, this is not necessarily true. It is hypothesised that leveraging any potential correlation between decision variables to influence sampling, the rate and quality of samples could be improved. The proposed algorithm which utilises covariance between decision variables when sampling, is presented in Algorithm 13.

As with the proposed MOO CEM-Beta algorithm in the previous section, this algorithm intends to replace steps $8 - 11$ of the original MOO CEM algorithm. Again, this algorithm is executed once per each histogram class.

Algorithm 13 MOO CEM-Cov: Algorithm to sample using covariance to be implemented in MOO CEM algorithm

```
 1: Find the Elite solutions which fall into the range $[c_{i\kappa}\, c_{i(\kappa+1)})$.
 2: Calculate the mean $\tilde{\mu}_i$ of the Elite subset.
 3: Calculate the standard deviation $\tilde{\sigma}_i$ of the Elite subset.
 4: if 1 or no unique Elite solutions fall within this range then
 5:     Set covariance matrix $\tilde{\Sigma}$ to $I$.
 6: else
 7:     Calculate the covariance matrix $\tilde{\Sigma}$ of the Elite subset.
 8: end if
 9: Generate a subsample $Y$ according to the mvn $\mathcal{N}$ $(\tilde{\mu}_i,\tilde{\Sigma})$.
10: with $\mathbf{x}_i \,\epsilon\, [c_{i\kappa}\, c_{i(\kappa+1)})$ and $|\mathbf{Y}| = \tau_{i\kappa}, 1 \leq \kappa \leq r+2$ .
11: Append $\mathbf{Y}$ to $\mathbf{X}_i$.
```

The covariance of the decision variables is calculated for each class in the histogram. Similarly to Algorithm 11, the elite solutions which fall within the histogram class are identified and the covariance is calculated on this subset of solutions. The calculation of covariance requires at least two unique points. If fewer than two solutions fall within the histogram class, the identity matrix I replaces the covariance matrix. The identity matrix is generally used as the covariance matrix when there is no known covariance between the samples or when the samples are uncorrelated. The same logic is applied when the first iteration of the algorithm is completed (and no elite solutions have been generated yet). The identity matrix is defined as follows:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In step 9, the solutions are sampled from the multi-variate normal distribution (mvn) using the mean and covariance to sample from the joint normal distribution, taking into account the relationship between the decision variables.

Matlab®'s multivariate normal distribution function (*mvncdf* used in conjunction with the *norminv* function to change the range of the samples to the desired range) was used to sample from the normal distribution with a covariance $\tilde{\Sigma}$ for each class in the histogram. *mvncdf* uses four different methods to sample from the multivariate normal distribution, depending on the number of dimensions (or decision variables):

1. For 1 dimension: normal cumulative distribution function (*normcdf*)
2. For 2 dimensions: bivariate normal cumulative distribution function (*bvncdf*)
3. For 3 dimensions: trivariate normal cumulative distribution function (*tvncdf*)
4. For more than 4 dimensions: quasi-Monte Carlo integration algorithm (*mvtcdfqmc*)

It is also important to note that the *mvncdf* function is limited to a maximum of 25 decision variables. This can be increased by writing a custom function, but Matlab® discourages this, as the evaluation time of this function increases with the number of decision variables.

3.3 Proposed MOO CEM-Constraint algorithm

This section proposes two different methods of adding functionality to the MOO CEM algorithm, giving it the capability to solve constrained problems. The two constraint methods suggested by Coello (2002), reviewed in Section 2.4, applied are:

1. Discarding all solutions which do not adhere to constraints (Subsection 3.3.1).
2. A constraint method based on a dynamic penalty (Subsection 3.3.2).

3.3.1 Constraint method 1: Discarding solutions

Unlike the algorithms proposed in the previous sections, this proposed algorithm aims to add new functionality to the existing MOO CEM algorithm, not to improve its existing functionality. Algorithm 14 implements the discarding of solutions method of constraint-handling suggested by Coello (2002). This algorithm essentially consists of a single step, which should be added between steps 16 and 17 of MOO CEM algorithm.

Algorithm 14 Algorithm to solve constrained problems with the MOO CEM algorithm using the discarding of solutions method

```
1: for each constraint q do
2:     Evaluate each solution $\mathbf{X}_i$, $1 \leq i \leq D$ against constraint $q$.
3:     Delete all solutions which do not adhere to the constraint.
4: end for
5: Increase the number of samples sampled per iteration $N$ proportional to the
   number of discarded solutions.
```

Once the points are sampled from the distribution, those which do not adhere to the constraints are simply discarded. This is done before calculating the objective functions and ranking the solutions accordingly. As a result, no constraint-violating solutions are considered.

As this method discards a number of solutions, the sample size N is essentially decreased. To ensure that N remains approximately the same size as initially assigned, the proportion of discarded solutions is calculated and the size of N is increased proportionally to account for the number of discarded solutions.

Although the implementation of this method is simple, it may discard good solutions with small constraint violations. This could lead the algorithm to converge on a set of sub-optimal solutions or stop the algorithm from converging; both scenarios preventing the algorithm from finding the true Pareto-optimal front.

3.3.2 Constraint method 2: Dynamic penalty

The second method implemented to enable the MOO CEM algorithm to solve constrained problems, is the dynamic penalty method used by the CMOEA algorithm developed by Woldesenbet *et al.* (2007). As this method was developed specifically for evolutionary algorithms, it was adapted for MOO CEM algorithm implementation. This method is more complex than the first constraint method implemented, but allows for more solutions to be considered, even when they violate constraints. It increases the diversity of the solutions, while penalising infeasible solutions depending on the size of the violation (how far a solution is from the feasible solution space).

Algorithm 15 implements the penalty method by Woldesenbet *et al.* (2007) and should update the objective function values to include the dynamic penalty. This will be achieved by replacing steps 17 and 18 of MOO CEM algorithm with steps $1-6$ of the proposed algorithm and replacing ranking steps as indicated in Algorithm 15.

Algorithm 15 Algorithm to solve constrained problems with the MOO CEM algorithm using the dynamic penalty method

```
 1: for each constraint q do
 2:     Calculate the distance measure $d_i$ of each solution according to (2.14).
 3:     Calculate the penalty $p_i$ of each solution according to (2.20).
 4:     Calculate the final modified objective value of each solution using (2.13).
 5: end for
 6: Rank the final modified objective values using the Pareto ranking of Algo-
    rithm 3 with a relaxed $\rho = 2$ to obtain an updated elite vector Elite.
 7: Continue with steps 19-22 of the MOO CEM algorithm.
 8: Rank the elite vector Elite on the final modified objective values using the
    Pareto ranking of Algorithm 3 with $\rho = 1$.
 9: Continue with steps 24-25 of the MOO CEM algorithm.
10: Rank the elite vector Elite on the original objective values using the Pareto
    ranking of Algorithm 3 with $\rho = 0$ to obtain the final elite vector.
```

It should be noted that when ρ is relaxed (steps 6 and 8), the solutions are ranked according to the final modified objective value formulation (which includes the penalty value), not the original objective value. This is to ensure that not only solutions which fall within the feasible region, but also good solutions which lie close to the feasible region (solutions with small penalties) are added to the elite set. When the elite solutions are ranked for the last time to establish the final set of elite solutions ($\rho = 0$ in Step 10), solutions are ranked according to the original objective value. This ensures that no infeasible solutions exist within the final set of Elite solutions.

The implementation of this method is more time-consuming and intricate than the first constraint method discussed, as the maximum of each constraint violation and minimum and maximum values of each objective must be calculated.

3.4 Proposed MOO CEM-ENS algorithm

A limitation of the original MOO CEM algorithm is its inability to solve problems with more than two objectives. This is due to the implementation of the non-dominated sorting method used and the computational effort associated therewith. In order to extend the functionality of the MOO CEM algorithm to solve problems with more than two objectives, a non-dominated sorting method which can sort more than two objectives at the same time should be implemented.

One of the latest and best performing algorithms is the ENS-SS algorithm developed by Zhang *et al.* (2015) (discussed in Section 2.5). This algorithm has been identified as the best suited non-dominated sorting method to extend the functionality of the MOO CEM algorithm, as it is the non-dominated sorting method which has shown the best performance for problems with more than two objectives and fewer than five objectives (see Figure 2.6).

No changes or updates are required in the implementation of this algorithm. The ENS-SS is implemented according to Algorithms 9 and 10. This algorithm replaces Algorithm 3 used for non-dominated sorting by the MOO CEM algorithm and should be used for Pareto ranking in steps 18, 23 and 26 of the MOO CEM algorithm (Algorithm 2).

3.5 Summary: Chapter 3

In this chapter four algorithms were proposed to improve and enhance the MOO CEM algorithm: MOO CEM-Beta, MOO CEM-Cov, MOO CEM-Constraint and MOO CEM-ENS. Two algorithms which incorporate constraint methods are proposed. Both algorithms will be tested and the algorithm with the best performance will be selected for implementation in the final proposed algorithm. In the subsequent chapter, the performance of these four algorithms is tested and quality indicators are calculated to determine the quality of each of them.

Chapter 4

Testing of proposed enhanced algorithms

In this chapter, the four algorithms proposed in the previous chapter are applied to the set of benchmark problems listed in Section 2.6. For the two enhancement methods (implementing the Beta distribution and covariance), the results of the approximate Pareto sets obtained are compared to those of the MOO CEM algorithm to determine if the enhancements caused a statistically significant change in the performance of the MOO CEM algorithm. For the two methods which add functionality to the algorithm (constraint-handling and non-dominated sorting), results are presented graphically and compared only to the true Pareto fronts using performance indicators, as the MOO CEM algorithm did not previously have the functionality to solve these types of problems. The hypervolume performance indicator was used as the main indicator to compare the performance of the algorithms. Where the hypervolume could not be calculated, the ϵ indicator was used.

4.1 Test specifications

This section presents the test specifications for the various algorithms. General specifications of the machine used to perform the tests are also provided.

4.1.1 MOO CEM-Beta and MOO CEM-Cov algorithms test specifications

In order to determine whether or not the algorithm's performance was significantly improved, a fair method of comparison is required. The approximate Pareto sets obtained from the algorithms are used as datasets for statistical testing. The paired t-test was used to determine whether the difference between the results of the original algorithm and the enhanced algorithms are statistically significant, as this is a widely accepted standard. The paired t-test is a special case of the general Analysis of Variance (ANOVA) and assumes unknown, unequal variances and independent, normally distributed samples (Bekker, 2019).

Two types of hypothesis tests exist: one-tailed and two-tailed hypothesis tests. A two-tailed hypothesis would be used to determine whether or not there is a statistically significant difference between two datasets. This gives no indication of which dataset has the larger mean, only that there is a difference between them. A one-tailed hypothesis test, on the other hand, is formulated such that the alternative hypothesis indicates that one dataset has a larger mean than the other. In this case, the objective is to determine whether or not the enhanced algorithm performed better than the original MOO CEM algorithm. This is done using three metrics: the hypervolume (or hyperarea for the two-dimensional case), runtime and the size of the Pareto set. In the case of hypervolume, the aim is to maximise it (see Section 2.6). For the runtime metric, the algorithm with the shortest runtime is most superior. The algorithm which produces the largest set of Pareto solutions is deemed the better algorithm. These three indicators should not be considered in isolation, as a fast algorithm with a poor solution set is of little value, while implementing an extremely slow, but accurate algorithm, might not be practical. Therefore, this analysis utilises the one-tailed hypothesis test for comparing each of the metrics. The hypotheses for the three metrics are formally stated as:

1. Hyperarea: The one-tailed right-tail hypothesis used for assessment is stated as

$$\begin{aligned} H_0 &: m_{B_H} \leq m_{M_H} \\ H_1 &: m_{B_H} > m_{M_H} \end{aligned}$$

   where m_{B_H} represents the mean hyperarea produced by the MOO CEM-Beta algorithm and m_{M_H} represents the mean hyperarea produced by the MOO CEM algorithm.

2. Runtime: The one-tailed right-tail hypothesis used for assessment is stated as

$$\begin{aligned} H_0 &: m_{B_t} \geq m_{M_t} \\ H_1 &: m_{B_t} < m_{M_t} \end{aligned}$$

   where m_{B_H} represents the mean runtime of the MOO CEM-Beta algorithm and m_{M_H} represents the mean runtime of the MOO CEM algorithm.

3. Pareto Set Size: The one-tailed right-tail hypothesis used for assessment is stated as

$$\begin{aligned} H_0 &: m_{B_s} \leq m_{M_s} \\ H_1 &: m_{B_s} > m_{M_s} \end{aligned}$$

   where m_{B_H} represents the mean size of the Pareto solution set produced by the MOO CEM-Beta algorithm and m_{M_H} represents the mean size of the Pareto solution set produced by the MOO CEM algorithm.

The null hypothesis states that there is no statistically significant difference between the two datasets. In contrast, the alternative hypothesis states that the parameter of the second dataset is either statistically smaller or larger than that of the first dataset. The t-test specifies that if $t^* > t_{crit}$, H_0 is rejected. The t-test was performed with a significance level of $\alpha = 0.05$. The critical value t_{crit} for this significance level and a sample size of 500 is 1.962. See Figure 4.1 for reference. If the calculated t^* value is greater than 1.962, the null hypothesis is rejected. t^* is calculated according to

$$t^* = \frac{(\bar{x}_1 - \bar{x}_2) - d_0}{\sqrt{(s_1^2/n_1) + (s_2^2/n_2)}} \tag{4.1}$$

where $\bar{x}_1$ represents the mean of dataset 1 and $\bar{x}_2$ the mean of dataset 2. s_1^2 and s_2^2 are the variances of datasets 1 and 2 respectively. n_1 and n_2 represent the sizes of the two datasets. The value of d_0 is set to 0 for all tests performed in this research project.

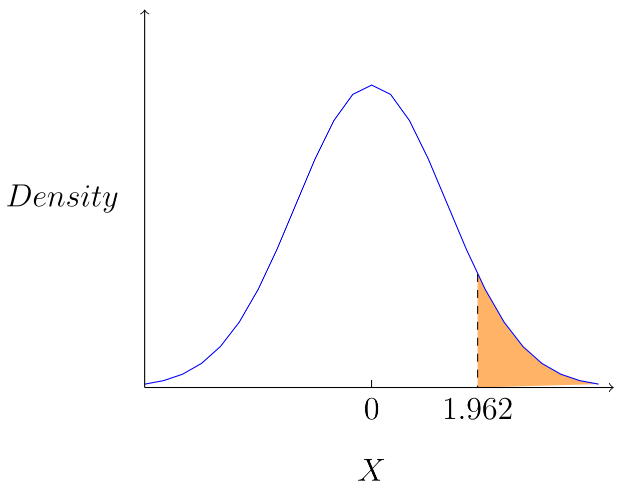


Figure 4.1: Probability distribution plot for a right-tailed t-test with $\alpha = 0.05$. The t_{crit} value for this level of significance is 1.9625. The null hypothesis is rejected when the test produces a t^* of greater than 1.9625

4.1.2 MOO CEM-Constraint and MOO CEM-ENS algorithms test specifications

The original MOO CEM algorithm did not have the functionality to solve problems with more than two objectives. Furthermore, it did not have constraint-handling capabilities. Therefore, no comparison could be made between the original MOO CEM algorithm and the two enhanced algorithms (MOO CEM-Constraint and MOO CEM-ENS) respectively.

Instead, the hyperarea is calculated, along with the ϵ indicator which serves as an additional performance indicator. In the case where the hyperarea could not be calculated, only the ϵ indicator is used. The hyperarea is maximised, while the ϵ indicator is minimised. Results are also presented graphically along with the true Pareto solution set.

4.1.3 General test specifications

The test simulations were conducted on a machine with an Intel i7 core (1.9 GHz) and 32 GB RAM. Each algorithm was run for 500 simulations with the following parameters:

1. Allowable number of loops: 100 (line 24 of Algorithm 2)
2. Population size N: 200
3. Maximum number of evaluations: 15 000

The above parameters were selected in line with those of the original MOO CEM algorithm. By selecting these parameters, advantage could be taken of the fact that the original MOO CEM algorithm requires fewer iterations (evaluations) than many other algorithms.

Upon testing, it was found that the MOO CEM-Constraint and MOO CEM-ENS algorithms require more iterations to converge to an acceptable solution set. Therefore, the maximum number of evaluations was increased from 15 000 to 30 000 for the constrained problems and problems with more than two objective functions.

The parameters were kept constant for the MOO CEM, MOO CEM-Beta and MOO CEM-Cov algorithms.

4.2 MOO CEM-Beta performance evaluation

In order to determine if the proposed MOO CEM-Beta algorithm performed statistically better than the original MOO CEM algorithm, both algorithms were tested on the set of standard test problems presented in Section 2.6.

Only the problems on which the original MOO CEM algorithm was tested by Bekker (2012) were considered. MOP5 and MOP7 were not tested as these problems have three objective functions and MOO CEM does not have the functionality to solve problems with more than two objectives. This was to ensure a comparable and fair test. The average hyperarea, execution time and Pareto set size results of each test are listed in Table 4.1 for reference.

The results indicate a larger hyperarea for some problems, shorter runtime and smaller Pareto set size, but simply comparing the average values does not indicate whether or not there is a statistically significant difference between the results. Therefore, the hypervolume, runtime and Pareto set size test results of the MOO CEM and MOO CEM-Beta algorithms were then compared using a one-tailed hypothesis test. The results of the one-tailed t-test are shown in Table 4.2. The hypothesis tests were formulated as stated in Section 4.1. For the hyperarea metric, the null hypothesis was rejected if a test produced a t^* value greater than 1.962, indicating that the MOO CEM-Beta algorithm produced a larger hyperarea, and this therefore indicates that the MOO CEM-Beta algorithm outperformed the MOO CEM algorithm. In terms of runtime, a left tailed t-test was used to compare the algorithm runtimes. A t^* value smaller than -1.962 will cause the null-hypothesis to be rejected, indicating a shorter runtime of the MOO CEM-Beta algorithm and therefore, superior performance.

Table 4.1: MOO CEM and MOO CEM-Beta results on some standard benchmark problems

		MOO CEM-Beta			MOO CEM		
Test Problem	Reference Hyperarea	Mean Hyperarea	Mean Runtime (s)	Mean Pareto Set Size	Mean Hyperarea	Mean Runtime (s)	Mean Pareto Set Size
MOP1	14.132	8.006	0.360	1728.874	13.771	1.216	5999.314
MOP2	0.323	0.322	0.760	896.958	0.322	0.284	888.728
MOP3	34.338	34.324	0.203	619.614	34.319	0.280	575.770
MOP4	28.918	27.935	0.508	193.748	28.136	0.196	223.888
MOP6	0.777	0.774	0.624	844.87	0.773	0.252	801.44
ZDT1	0.767	0.704	2.542	476.50	0.695	6.772	464.600
ZDT2	6.833	6.772	2.244	289.430	6.189	4.377	310.150
ZDT3	1.040	0.950	1.800	163.92	0.587	2.243	167.038

Table 4.2: One-tailed t-test results of MOO CEM compared MOO CEM-Beta results on some standard benchmark problems

	Hyperarea		Runtime		Pareto Set Size	
Test Problem	t^*	Outcome	t^*	Outcome	t^*	Outcome
MOP1	-18.647	No reject	43.578	Reject	-46.980	No reject
MOP2	0.180	No reject	-68.593	No reject	1.008	No reject
MOP3	0.333	No reject	23.002	Reject	6.887	Reject
MOP4	-38.637	No reject	-43.746	No reject	-7.054	No reject
MOP6	2.529	Reject	-64.186	No reject	7.043	Reject
ZDT1	2.631	Reject	21.804	Reject	1.878	No reject
ZDT2	7.800	Reject	34.014	Reject	1.326	No reject
ZDT3	17.161	Reject	12.832	Reject	1.570	No reject

The tests indicate that, for problems with a smaller number of decision variables (MOPs 1-4), the MOO CEM-Beta algorithm does not outperform the MOO CEM algorithm in terms of maximising the hyperarea. In fact, by considering the mean hyperareas alongside the t-test results for MOP4 and MOP6, it is clear that the quality of the MOO CEM algorithm solutions exceed those produced by the MOO CEM-Beta algorithm. However, for MOPs ZDT1, ZDT2 and ZDT3, which each have 30 decision variables, MOO CEM-Beta outperforms the MOO CEM algorithm and produces solutions with a better hyperarea. MOO CEM-Beta also shows superior performance for MOP6 regarding the hyperarea performance indicator.

With respect to runtime, the MOO CEM-Beta algorithm has faster execution times for 5 out of the 8 problems (62.5%). The size of the MOO CEM algorithm Pareto sets is generally larger (for 75% of problems) than the size of the Pareto sets produced by the MOO CEM-Beta algorithm.

Considering these results, it can be concluded that the MOO CEM-Beta algorithm produces similar results to the MOO CEM algorithm, generally with a shorter runtime. Furthermore, the MOO CEM-Beta algorithm delivers superior results to the MOO CEM algorithm for problems with a large number of decision variables.

Figures 4.2-4.5 display the MOO CEM and MOO CEM-Beta approximate Pareto sets and the true Pareto sets. The approximate Pareto sets were recalculated after the completion of the 500 simulations. The Pareto sets produced by the MOO CEM and MOO CEM-Beta are almost indistinguishable for most test problems, but the superiority of the MOO CEM-Beta algorithm can be clearly seen in test problems ZDT1-3.

Refer to Appendix A for box-and-whisker diagrams comparing the hyperareas of the MOO CEM, MOO CEM-Beta and true Pareto solution sets.

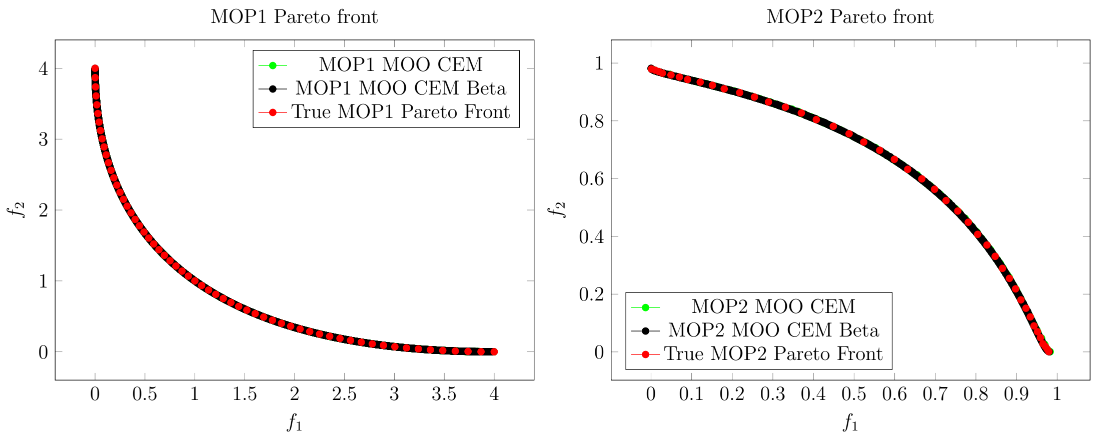


Figure 4.2: MOO CEM and MOO CEM Beta Pareto sets compared to the true Pareto set for test problems MOP1 and MOP2

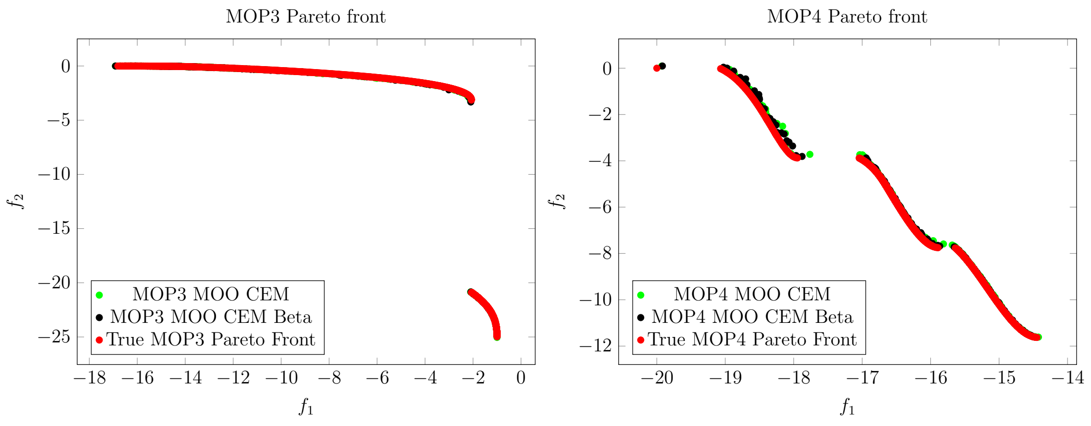


Figure 4.3: MOO CEM and MOO CEM Beta Pareto sets compared to the true Pareto set for test problems MOP3 and MOP4

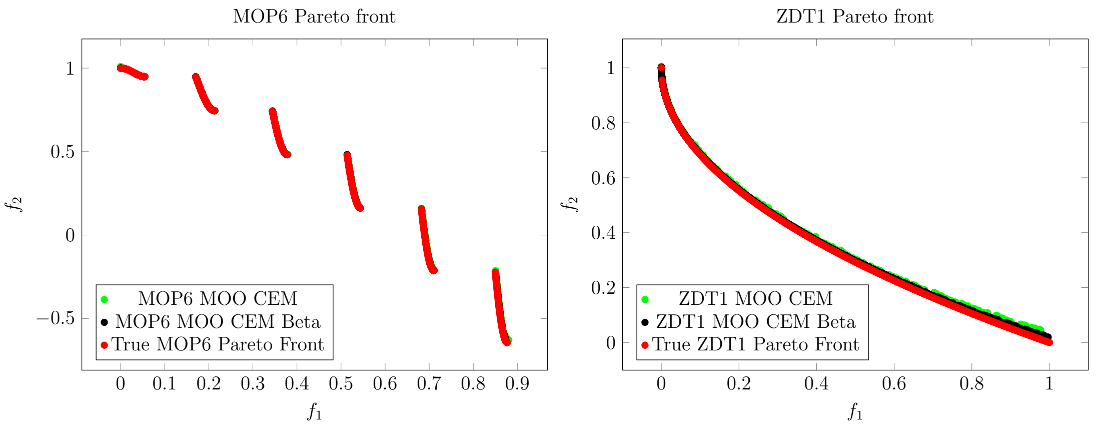


Figure 4.4: MOO CEM and MOO CEM Beta Pareto sets compared to the true Pareto set for test problems MOP6 and ZDT1

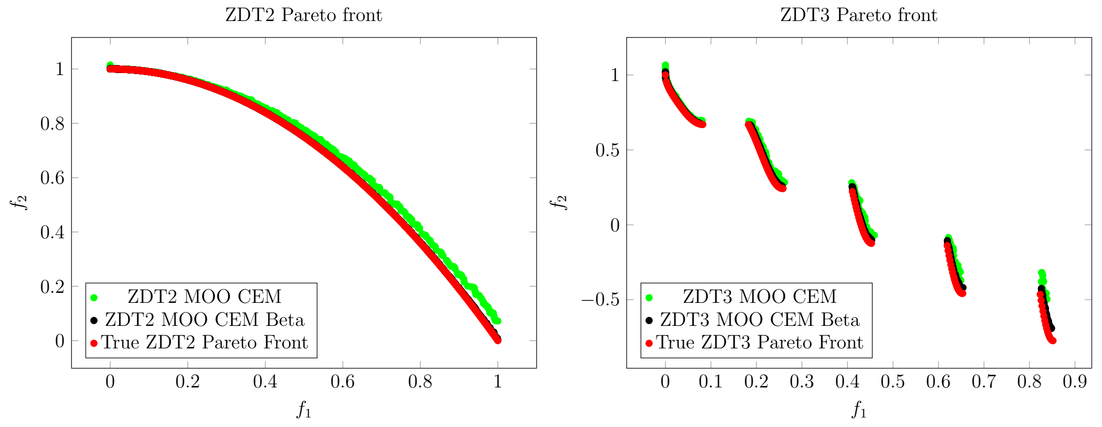


Figure 4.5: MOO CEM and MOO CEM Beta Pareto sets compared to the true Pareto set for test problems ZDT2 and ZDT3

4.3 MOO CEM-Cov performance evaluation

Covariance exploits the relationship between multiple decision variables. As the MOP1 test problem only has one decision variable, it is required that this problem is solved using the original MOO CEM algorithm.

The one-tailed hypothesis tests described in Section 4.1 were conducted similarly to the MOO CEM-Beta - MOO CEM comparison discussed in the previous section. The mean hyperarea, runtime and Pareto set size test results for problems MOP2, MOP3, MOP4 and MOP6 are shown in Table 4.3 and the results of the hypothesis tests are recorded in Table 4.4.

The MOO CEM-Cov algorithm could not be tested on the ZDT1, ZDT2 and ZDT3 test problems. This was due to Matlab's® maximum number of decision variables allowed (25), whereas these test problems each have 30 decision variables. This limitation was circumvented by manually adjusting the decision variable threshold, but due to the quasi-Monte Carlo integration algorithm used to solve multivariate problems with more than three variables (see Section 3.2), the extensive runtime made solving these problems using covariance infeasible.

Considering the hyperarea results of the two algorithms, the MOO CEM-Cov algorithm does not produce a t^* value greater than the critical t-value (1.962), and therefore does not yield better results than the MOO CEM algorithm. Regarding the runtime performance indicator, the MOO CEM algorithm has a shorter runtime for all problems. The MOO CEM algorithm generally produces a larger Pareto set than the MOO CEM-Cov algorithm (for 75% of problems). The only problem for which the MOO CEM-Cov algorithm produces a larger Pareto set than the MOO CEM algorithm, is MOP6.

Given the results, it can be concluded that the MOO CEM-Cov does not produce better results than the MOO CEM algorithm.

Figures 4.6-4.7 display the MOO CEM and MOO CEM-Cov approximate Pareto sets and the true Pareto sets. The approximate Pareto sets were recalculated after the completion of the 500 simulations. For more detail regarding these results, refer to Appendix A which contains box-and-whisker diagrams comparing the hyperareas produced by MOO CEM and MOO CEM-Cov and the true Pareto set.

Table 4.3: MOO CEM and MOO CEM-Cov results on some standard benchmark problems

		MOO CEM-Cov			MOO CEM		
Test Problem	Reference Hyperarea	Mean Hyperarea	Mean Runtime (s)	Mean Pareto Set Size	Mean Hyperarea	Mean Runtime (s)	Mean Pareto Set Size
MOP2	0.323	0.320	14.307	933.098	0.322	0.284	888.728
MOP3	34.338	33.524	0.800	1321.004	34.319	0.280	575.770
MOP4	28.918	28.035	8.481	474.082	28.136	0.196	223.888
MOP6	0.777	0.771	2.150	820.322	0.773	0.252	801.44

Table 4.4: One-tailed t-test results of MOO CEM compared MOO CEM-Cov results on some standard benchmark problems

	Hyperarea		Runtime		Pareto Set Size	
Test Problem	t^*	Outcome	t^*	Outcome	t^*	Outcome
MOP2	-15.915	No reject	-66.465	No reject	5.771	Reject
MOP3	-13.129	No reject	-59.217	No reject	71.923	Reject
MOP4	-2.327	No reject	-55.126	No reject	-38.757	Reject
MOP6	-2.122	No reject	-70.370	No reject	0.664	No reject

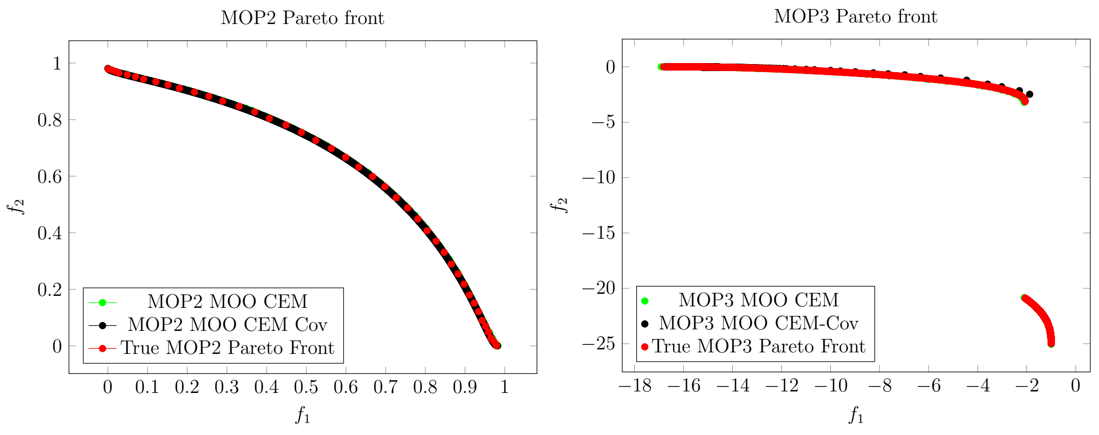


Figure 4.6: MOO CEM and MOO CEM-Cov Pareto sets compared to the true Pareto set for test problems MOP2 and MOP3

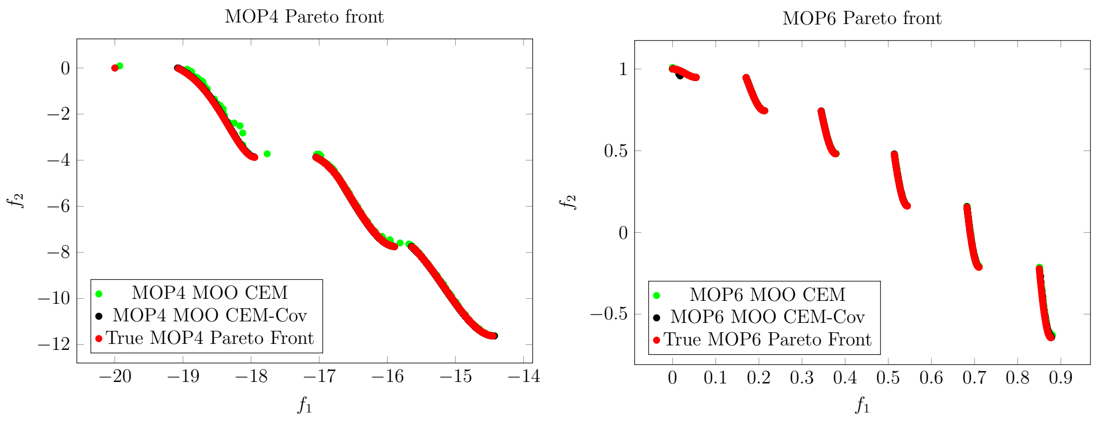


Figure 4.7: MOO CEM and MOO CEM-Cov Pareto sets compared to the true Pareto set for test problem MOP4 and MOP6

4.4 MOO CEM-Constraint performance evaluation

The tests conducted on the MOO CEM-Beta algorithm showed promising results. Therefore, this algorithm (rather than the original MOO CEM algorithm) was enhanced to include both constraint methods discussed in Section 3.3.

This enhancement gives the algorithm the ability to solve problems with side-constraints. Four standard side-constraint problems (MOP-C1, -C2 , -C3 and -C4, see Table 2.5 for details) were selected to test the performance of the algorithm. The results of MOP-C3 are discussed in the subsequent chapter, as this problem has three objective functions, which cannot be solved by the current variation of the MOO CEM algorithm.

The method of discarding infeasible solutions (Section 3.3.1) was applied to the algorithm first (according to Algorithm 14), as this is the simpler of the two techniques. Although this method could locate the Pareto solution set for some test problems, a high number of iterations and an increased population size was required. The result was a long runtime and an inaccurate Pareto solution set.

Thereafter, the dynamic penalty method considered in Section 3.3.2 was applied to the MOO CEM-Beta algorithm according to Algorithm 15 to develop the MOO CEM-Constraint algorithm. The hyperarea and ϵ indicator were calculated for these two-objective problems and compared to the true Pareto solution set. The average runtime and size of the Pareto solution set were also recorded in Table 4.5. Figures 4.8 and 4.9 show the solution sets generated by the enhanced algorithm compared to the true Pareto solution set for problems MOP-C1, MOP-C2 and MOP-C4. Box-and-whisker plots of the hyperarea can be found in Appendix A.3.

Table 4.5: MOO CEM-Constraint results on some standard constrained benchmark problems

		MOO CEM-Constraint			
Test Problem	Reference Hyperarea	Mean Hyperarea	Epsilon Indicator	Mean Runtime (s)	Mean Pareto Set Size
MOP-C1	8333.333	8332.675	$< 10^{-3}$	7.330	10031.262
MOP-C2	13530.128	11481.3372	2.885	1.011	140.656
MOP-C4	0.319	0.294	0.106	0.399	170.742

Comparing the hyperarea generated by the solution set to the reference hyperarea (hyperarea of the true Pareto solution set), for problem MOP-C1 a mean difference in hyperarea of less than 0.01% is observed. This indicates that the solution set produced by the algorithm and the true Pareto solution set are very similar. This observation is supported by the very small ϵ indicator ($< 10^{-3}$), indicating a very small difference between the solution set produced by the algorithm and the true Pareto solution set. The Pareto set produced is large and the runtime is rather lengthy. This long runtime could be attributed to the additional complexity of the dynamic penalty method.

For problem MOP-C2, a larger difference between the hyperarea of generated solution set and true Pareto solution set is seen (15%). Considering Figure 4.8, it is observed that the generated solutions closely approximate the true Pareto front for larger values of f_1, but deviate from the true front for smaller values of f_1. Nevertheless, the ϵ indicator remains relatively small (2.885), indicating that the generated solution set is usually not far from the true Pareto solution set. The runtime of this problem is significantly shorter than that of MOP-C1, which could be as a result of the notably smaller Pareto set. The accuracy of the generated solutions could possibly be improved by increasing the number of iterations for which the algorithm is run. Since the runtime of this problem is currently short, this may be a feasible solution.

The results of MOP-C4 show a mean difference in hyperarea of the generated solution set and the true Pareto solution set of 8%. The ϵ indicator denotes a relatively small difference between the two solution sets (0.106), suggesting that the generated solution set closely approximates the true Pareto set. A short runtime and relatively small Pareto size are observed. For MOP-C2, the generated solution set could be improved as suggested above.

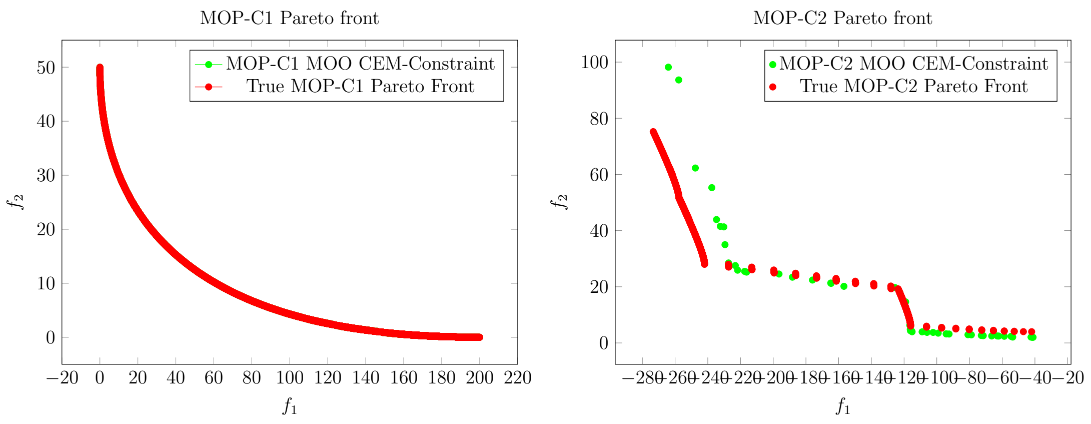


Figure 4.8: MOO CEM-Constraint Pareto sets compared to the true Pareto set for test problems MOP-C1 and MOP-C2

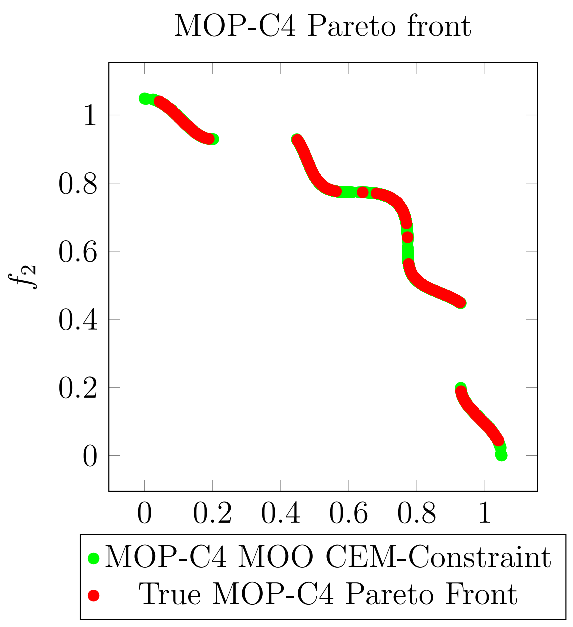


Figure 4.9: MOO CEM-Constraint Pareto sets compared to the true Pareto set for test problem MOP-C4

4.5 MOO CEM-ENS performance evaluation

The MOO CEM-Beta algorithm was enhanced by including the ENS-SS ranking algorithm (Algorithms 9 and 10) as discussed in Section 3.4. This algorithm is referred to as MOO CEM-ENS. This enhancement gives the algorithm the ability to solve problems with more than two objective functions. In order to evaluate the performance of this enhanced algorithm, it was tested on three benchmark problems (MOP5, MOP7 and MOP-C3), each having three objective functions.

Due to the computational complexity associated with calculating hypervolume, the ϵ indicator was used as the performance metric for evaluation. The average runtime and size of the Pareto solution set were recorded and results can be found in Table 4.6. Figures 4.10 and 4.11 show the solution sets generated by the enhanced algorithm as well as the true Pareto solution set. Box-and-whisker plots of the hyperarea are displayed in Appendix A.4.

Table 4.6: MOO CEM-ENS results on some standard benchmark problems with more than two objectives

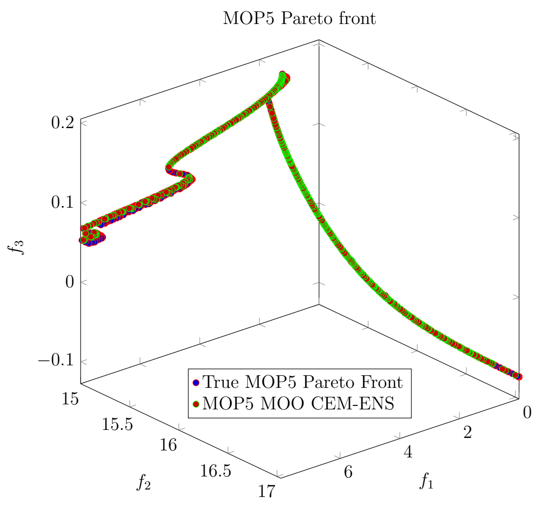
	MOO CEM-ENS		
Test Problem	Epsilon Indicator	Mean Runtime (s)	Mean Pareto Set Size
MOP5	0.030	178.977	216.24
MOP7	0.004	73.209	4333.357
MOP-C3	0.480	284.382	8131.985

It is observed that the runtime of these problems is significantly longer than those of problems with only two objective functions. A portion of this lengthy runtime can be attributed to the time required by the ranking and selection algorithm (ENS). The runtimes cannot be compared to those of the MOPs in Section 4.2, as these problems were run for 15 000 iterations, while MOP5, MOP7 and MOP-C3 required double the number of iterations to converge to a reasonable solution set.

For MOP5, when comparing the solution set of the algorithm to the true Pareto solution set, the ϵ indicator was found to be very small (0.03). This indicates that there is very little difference between the algorithm solution set and the true Pareto solution set. The size of the generated Pareto set is relatively small compared to MOP7 and MOP-C3; however, given the small ϵ indicator, the solution set closely approximates the true Pareto solution set.

The ϵ indicator calculated for MOP7 is extremely small (0.004), implying that the algorithm solution set and the true Pareto solution set are almost identical. This observation is supported by Figure 4.10, showing that the two sets are almost indistinguishable. For this problem, the algorithm produced a large Pareto solution set.

The results of MOP-C3 show a relatively small ϵ indicator (0.48). It can be deduced that the algorithm produces a good solution set which approximates the true Pareto solution set. From Figure 4.11, it is observed that the algorithm produces extremely good solutions for smaller values of f_3, where solutions are concentrated. As values of f_3 increase, the solutions become less concentrated and resemble a line. For these larger values of f_3, the solutions produced by the algorithm are further from the true Pareto set. It is theorised that the accuracy of the solutions could be increased by increasing the number of iterations, but this would increase the already lengthy runtime. For this problem, a very large Pareto solution set is produced.



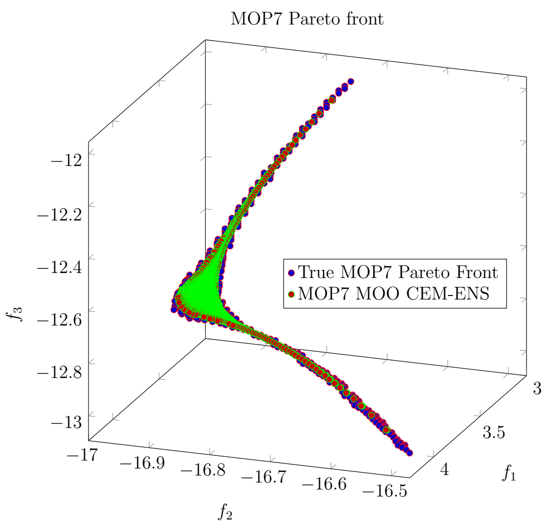


Figure 4.10: MOO CEM-ENS Pareto sets compared to the true Pareto set for test problems MOP5 and MOP7

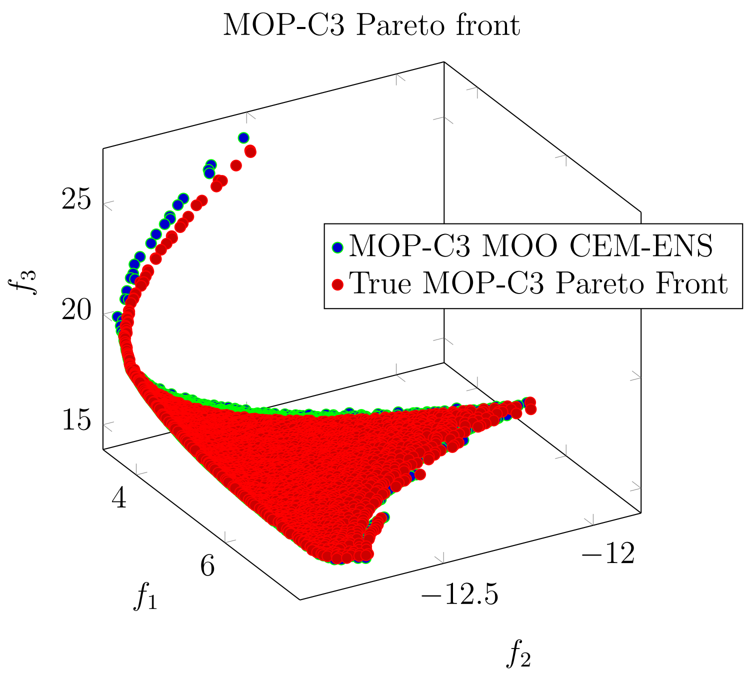


Figure 4.11: MOO CEM-ENS Pareto sets compared to the true Pareto set for test problem MOP-C3

4.6 Summary: Chapter 4

The results showed that the MOO CEM-Beta algorithm outperforms the original MOO CEM algorithm on some benchmark problems. This indicates that replacing the truncated normal distribution with the Beta distribution is a suitable method of improving the algorithm's sampling method.

Using covariance of decision variables does not improve sampling and the MOO CEM-Cov algorithm did not perform as well as the MOO CEM algorithm.

The MOO CEM-Beta algorithm was further enhanced by using the dynamic penalty method to solve constrained problems (MOO CEM-Constraint). Furthermore by implementing the ENS-SS non-dominated sorting algorithm, MOO CEM-Beta also boasts the functionality to solve problems with more than two objective functions (MOO CEM-ENS).

In the next chapter, a final algorithm combining the enhancement and improvement methods is proposed.

Chapter 5

Proposed enhanced MOO CEM algorithm

Taking into account the results presented in the previous chapter, a final algorithm is proposed. This algorithm uses the Beta distribution for sampling (replacing the original truncated normal distribution sampling), enhances the original MOO CEM algorithm by giving it the ability to solve constrained problems (through the addition of a dynamic penalty function) and problems with more than two objective functions (by replacing the Pareto ranking algorithm with ENS-SS). Test results indicate that considering covariance when sampling is not an appropriate method to improve the sampling of the MOO CEM algorithm, especially not for problems with a very large number of decision variables, and it is therefore not included in the final algorithm.

This final algorithm pseudo code is presented in Algorithm 16. The algorithm follows the same logic as the original MOO CEM algorithm, with the addition of the proposed enhancements. The Matlab® code of the algorithm is presented in Appendix B.

Algorithm 16 Enhanced MOO CEM Algorithm

```
1: Set $Elite = \varnothing, t = 1, k = 1$.
2: Initialise variable vectors $\mathbf{X}_i = \varnothing, 1 \leq i \leq D$, and compute initial objective values.
3: For each decision variable $x_i, 1 \leq i \leq D$ initialise a histogram class vector $\mathbf{C}_i = \{c_{i1}, c_{i2}, \ldots, c_{i(r+2)}, c_{i(r+2)+1}\}$ and histogram frequency vector $\mathbf{R}_i = \{\tau_{i1}, \tau_{i2}, \ldots, \tau_{i(r+1)}, \tau_{i(r+2)}\}$.
4: Set $i = 1$.
5: Set $\kappa = 0$.
6: Increment $\kappa$.
7: **for** each frequency element $\tau_{i\kappa}$ in $\mathbf{R}_i$ **do**
8:     Find the Elite solutions which fall into the range $[c_{i\kappa}\, c_{i(\kappa+1)})$.
9:     **if** one or no unique Elite solutions fall within this range **then**
10:        Set $\alpha_{i\kappa} = 1$.
11:        Set $\beta_{i\kappa} = 1$.
12:    **else**
13:        Calculate a class based $\alpha_{i\kappa}$ and $\beta_{i\kappa}$ of the distribution of the corresponding Elite solutions over the normalised range $0 - 1$.
14:    **end if**
15:    Generate a subsample $Y$ according to the pdf $Beta\ (\alpha_i,\beta_i)$.
16:    with $\mathbf{x}_i \ \epsilon\ [c_{i\kappa}\, c_{i(\kappa+1)})$ and $|\mathbf{Y}| = \tau_{i\kappa}, 1 \leq \kappa \leq r + 2$ .
17:    Append $\mathbf{Y}$ to $\mathbf{X}_i$.
18: **end for**
19: If $\kappa < r + 2$ , return to Step 6.
20: Invert the histogram counts with probability $p_h$.
21: Increment $i$.
22: If $i \leq D$, return to Step 5.
23: Compute the $NK$ objective function values using $\mathbf{X}_i, 1 \leq i \leq D$
24: **for** each constraint **q do**
25:    Calculate the distance measure $d_i$ of each solution according to (2.14).
26:    Calculate the penalty $p_i$ of each solution according to (2.20).
27:    Calculate the final modified objective value of each solution using (2.13).
28: **end for**
29: If the problem is constrained, rank the final modified objective values, otherwise rank the objective function values using the Pareto ranking of Algorithm 9 with a relaxed $\rho_E = 2$ to obtain an updated elite vector **Elite**.
```

```
30: Form new histogram class vectors $\mathbf{C}_i$ and histogram frequency vectors $\mathbf{R}_i$
    based on Elite , $1 \leq i \leq D$.
31: Use the values in Elite and compute $\boldsymbol{\alpha}_{it}$ and $\boldsymbol{\beta}_{it}$ for all $i, 1 \leq i \leq D$.
32: Smooth the vectors $\boldsymbol{\alpha}_{it}$ and $\boldsymbol{\beta}_{it}$ using (2.3).
33: Calculate the differences between $\boldsymbol{\alpha}_{it}$ and $\boldsymbol{\alpha}_{it-1}$; and $\boldsymbol{\beta}_{it-1}$ and $\boldsymbol{\beta}_{it}$.
34: If all changes in $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i > \epsilon_c$, or less than the allowable number of evalua-
    tions have been done, increment $t$ and reiterate from Step 4.
35: If the problem is constrained, rank the elite vector Elite on the final modified
    objective values, otherwise rank the elite vector Elite on the objective values,
    using the Pareto ranking of Algorithm 9 with $\rho_E = 1$.
36: Increment $k$.
37: If $k$ is less than the allowable number of loops, return to Step 2.
38: Rank the elite vector Elite using the Pareto ranking Algorithm 9 with $\rho_E = 0$
    to obtain the final elite vector.
```

The algorithm commences by creating a number of variables: *Elite*, which will contain the Pareto solution set, the allowable number of evaluations t, the allowable number of loops k and a vector for each decision variable x_i. In Step 3, the decision space of each decision variable is divided into $r + 2$ histogram classes with a total frequency of N (the number of solutions).

Beta distribution parameters $\alpha_{i\kappa}$ and $\beta_{i\kappa}$ are calculated for each histogram class. This is done in Step 7 by first identifying which of the solutions of the previous iteration fall into the current histogram class. If fewer than two solutions fall within the class, $\alpha_{i\kappa}$ and $\beta_{i\kappa}$ cannot be calculated and are, therefore, each assigned a value of 1 (Steps 10 and 11). With parameters having a value of 1 a uniform distribution is used for sampling for the particular class. Otherwise, the solutions in the class are normalised (to fall within the range $0-1$) and parameters $\alpha i\kappa$ and $\beta i\kappa$ are calculated for the distribution in Step 13. New solutions are then generated according to the probability density function of the Beta distribution with parameters $\alpha_{i\kappa}$ and $beta_{i\kappa}$ in Steps 15 to 17. The number of solutions sampled is determined by the frequency element $\tau_{i\kappa}$. To ensure diversity, in Step 20 the histogram counts (or frequency elements) are inverted with a probability p_h (for example, if N is 10 and the histogram count is 2, the inverted count would be 8). This process is repeated for each decision variable.

The objective functions of each of the N solutions are calculated in Step 23. If the problem is constrained, the penalty value of each sample is then calculated according to the dynamic penalty method (Subsection 3.3.2) in Steps 25 and 26. In Step 27, a modified objective value is then calculated by taking the penalty into account. For a minimisation problem, the penalty is added to the original objective value. For a maximisation problem, on the other hand, the penalty value is subtracted. In this way, solutions far from the feasible region will have very large penalties and therefore large modified objective values, similar to poor solutions that fall within the feasible region (which will have no added penalty value).

The solutions are ranked according to their calculated objective values in Step 29. Unconstrained problems are ranked according to their original objective function values. For constrained problems, the solutions are ranked according to the modified objective value. ENS-SS is used as the ranking algorithm, as this algorithm can rank more than two objective values at once and is relatively computationally efficient. A relaxed $\rho_E = 2$ value is chosen in this step, which refers to the third best solution set. It includes solutions which have a rank of two or less, which includes the best ($\rho_E = 0$), second best ($\rho_E = 1$) and third best solution sets. This is to ensure exploration and exploitation of the algorithm. The second time the solution is ranked (Step 35), ρ_E is set to 1 and objectives are again ranked on the modified objective values. However, when solutions are ranked for the last time (in Step 38), only solutions with a rank of $\rho_E = 0$ are preserved and solutions are ranked on the original objective values. This is to ensure all solutions are feasible.

Based on the new solution set (or *Elite* set), new histogram classes and frequencies are calculated in Step 30. A single α_{it} and β_{it} per iteration are calculated for each decision variable and smoothed in Steps 31 and 32. Smoothing is done using the ω parameter, which can be increased or decreased to change the ratio of old to new (current iteration) solutions.

These values of α_{it} and β_{it} are used as one of the methods to determine whether the stopping criteria are met. This is done by comparing these values to the values of α_{it-1} and β_{it-1} of the previous iteration. If there is no change in either α or β between iterations, the algorithm is assumed to have reached steady state, indicating that the solution has converged and additional iterations would not improve the solution (Step 34). In Step 35, the solutions are ranked with an adjusted value of $\rho_E = 1$, discarding weaker solutions. This is the final step of the iteration and the iteration number k is incremented in Step 36. The second stopping criterion is reached when the number of the current iteration reaches the maximum allowable number of loops in Step 37. If the maximum number of iterations is reached, the algorithm is terminated with one final ranking of the solutions in Step 37. Otherwise, the algorithm is repeated from Step 2 for the next iteration.

The enhanced MOO CEM algorithm outlined in Algorithm 16 incorporates functionality which makes it more flexible and suited to a variety of problem classes. In the next chapter, the research conducted in this thesis is summarised and conclusions are reached regarding the enhancement techniques and proposed algorithms. Future areas of research which build on the findings presented in this thesis are also put forward.

Chapter 6

Research summary and conclusions

This chapter summarises the research conducted and presents the research conclusions. Future areas of research are suggested and an appraisal of the research is performed. Finally, some concluding remarks are made.

6.1 Project summary and conclusions

This study explored some methods of improving and enhancing the MOO CEM algorithm developed by Bekker (2012). After studying the MOO CEM algorithm, the following three areas were identified with direction from the developer of the original MOO CEM algorithm, Bekker (2012):

- Improve the method of sampling.
- Add functionality to solve constrained problems.
- Add functionality to solve problems with more than two objectives.

Upon performing a literature review of the techniques available for each of the suggested improvements and enhancements, four methods were selected to address the areas of improvement and enhancement:

- Use the Beta distribution as a method of improving sampling, rather than the truncated normal distribution.

- Consider using the covariance of the decision variables to improve sampling.
- Enhance the MOO CEM algorithm by adding functionality to solve constrained problems through the use of one of two techniques: the elimination method or the dynamic penalty method.
- Use the ENS-SS algorithm to sort solutions according to more than two objective functions, thereby adding the functionality to solve problems with more than two objectives.

Each of the four techniques was applied to the MOO CEM algorithm separately and the new proposed algorithms were tested on benchmark problems. The results were compared to those produced by the original MOO CEM algorithm where possible (for problems which could be solved by MOO CEM) by means of a one-tailed t-test, based on the hyperarea, runtime and size of the observed Pareto set. For those problems which could not be solved by MOO CEM, due to the limitations of the algorithm, two Pareto-compliant indicators were calculated and reported: the hyperarea (which is to be maximised) and the ϵ indicator (which is to be minimised). For certain problems, the hyperarea could not be calculated due to the shape of the solution set. In these cases, only values for the ϵ indicator were presented.

The results produced by the MOO CEM-Beta algorithm (which replaced the truncated normal distribution with the Beta distribution) showed great promise. Results indicate that the MOO CEM-Beta algorithm produces a hyperarea no worse than that produced by the MOO CEM algorithm for some problems (MOP1-MOP4) and a larger hyperarea for other problems (MOP6 and ZDT1-ZDT3), suggesting a better solution set, and therefore, better performance. The MOO CEM-Beta algorithm performs particularly well for problems with a high number of decision variables (ZDT problems with 30 decision variables each). In general, the MOO CEM-Beta is faster (has a shorter runtime) than the MOO CEM algorithm. Although MOO CEM-Beta generally produces a smaller observed Pareto set, this metric cannot be considered in isolation and the performance of the algorithm is evaluated by considering all three metrics simultaneously. In summary, replacing the truncated normal distribution with the Beta distribution improves the quality of the solution set produced by the algorithm.

The solution sets produced by the MOO CEM-Cov algorithm (which takes covariance of the decision variables into account, rather than using the truncated normal distribution for sampling) were compared to those produced by the original MOO CEM algorithm. The test results indicate that considering covariance of decision variables does not improve sampling and, therefore, the quality of the solution sets. This was particularly evident for problems with a high number of decision variables (such as ZDT problems with 30 decision variables each) where the algorithm did not converge within a reasonable amount of time.

On account of the promising results shown by the MOO CEM-Beta algorithm, the two suggested enhancements were applied to this algorithm, rather than to the original MOO CEM algorithm. In terms of constraints, both suggested constraint methods were applied to the MOO CEM-Beta algorithm.

The simplest method of elimination was found not to be suitable, as the number of iterations and sample size required to achieve an acceptable solution set were too large. It is concluded that this is not a practicable method of solving constrained problems. The second method was considered: using a dynamic penalty method to evaluate infeasible solutions. This technique produced positive results. All problems showed a small difference in hyperarea compared to that of the true Pareto solution set and the calculated ϵ indicator was small, both indicating that the enhanced algorithm produces a solution set which approximates the true Pareto solution set. This implies that applying the dynamic penalty method is an appropriate method of enhancing the MOO CEM algorithm, giving it the functionality to solve constrained problems.

The ENS-SS algorithm for the ranking and selection of solution sets with more than two objectives was applied to the MOO CEM-Beta algorithm. The performance of the algorithm was tested on three problems, using the ϵ indicator as a performance metric. The tests of each problem produced extremely small ϵ indicators (all less than 0.5), suggesting that the solutions produced by the enhanced algorithm and the true Pareto solution set are acceptably similar. This indicates that the integration of the ENS-SS algorithm with the MOO CEM-Beta algorithm was successful, giving it the ability to solve problems with more than two objectives.

Considering the results produced by each of the improved and enhanced algorithms, a final algorithm was presented in Chapter 5, which includes the Beta distribution, ENS-SS algorithm and the dynamic penalty method, as these algorithms used in conjunction, produced the best results.

In summary, the research objectives set out in Chapter 1 were achieved through the following:

1. The four suggested methods of improvements and enhancements were each applied to the original MOO CEM algorithm.

2. It was established which of these techniques are suitable as methods of improvement

3. Using the Beta distribution as a sampling method improved the results of the algorithm.

4. Incorporating the covariance of the decision variables in the sampling method did not improve the performance of the algorithm.

5. The dynamic penalty method added constraint handling functionality to the new MOO CEM-Beta algorithm.

6. Replacing the Pareto ranking and selection algorithm with ENS-SS gave the new MOO CEM-Beta algorithm the ability to solve problems with more than two objectives.

7. A final enhanced MOO CEM algorithm was proposed, incorporating the successful improvements and enhancements.

Considering the summary and conclusions drawn from the research, a few suggestions are made for further research in the subsequent section.

6.2 Future research

Building on the research presented in this study, the following suggestions are made for future research:

1. Test the proposed algorithm on more problems with high numbers of decision variables. The results of the MOO CEM-Beta tests indicated that the algorithm is especially effective on problems with a high number of decision variables, but this observation should be verified through additional testing.
2. The proposed algorithm should be tested on problems with more than three objective functions.
3. A faster sorting algorithm could be considered to decrease the runtime of problems with a high number of objectives.
4. The performance of the algorithm should be compared to other industry leading optimisation algorithms.

In the next section, the study presented is critically evaluated.

6.3 Appraisal of research

During the course of conducting the research required for this study, the researcher gained knowledge in various areas of optimisation. A number of techniques related to the four methods of enhancement were investigated. The researcher is confident that sufficient evidence has been provided of the value added to the original MOO CEM algorithm through the addition of the selected techniques, thereby meeting the objectives set out at the start of the research.

However, the performance of the newly proposed algorithm should be tested on additional test problems, as well as on practical industry problems. Specifically, constrained and many-objective problems should be included.

The developed enhanced algorithm has been tested on a representative, yet limited, set of benchmark problems on which it performed satisfactorily. The researcher does not claim that the enhanced algorithm will outperform all other optimisation algorithms on all problems, or that the algorithm would be able to solve every new optimisation problem. This is in line with the *No Free Lunch Theorem* by Wolpert & Macready (1997).

During the development of the proposed algorithm, a new sorting algorithm was published by Moreno *et al.* (2020): MNDS. This algorithm has proven to be especially efficient for problems with many dimensions. Should the new enhanced MOO CEM algorithm be applied to these types of problems, it is recommended that this algorithm be investigated.

The research is concluded with some closing remarks in the last section.

6.4 Concluding remarks

In this final section, the author wishes to share some reflections regarding this study.

This research introduced the researcher to the field of optimisation. Having spent some time working in the industry, made the researcher acutely aware of the importance and value of optimisation, specifically multi-objective optimisation. Realistically, few problems have only one single goal and even fewer are unconstrained. By enhancing the original MOO CEM algorithm, significant functionality has been added, making it suitable for practical implementation in the industry.

In industry, time is limited and there is little opportunity for exploratory research. This project has given the researcher a chance to explore different areas of optimisation improvement and implement these techniques. Some techniques proved successful, while others did not. Although unsuccessful, this knowledge is not without value, as it would prevent future researchers from spending time exploring the same unfruitful areas.

In closing, an insight by William Of Occam, inspiring us all to optimise: *"It is futile to do with more things that which can be done with fewer"*.

Bibliography

Astolfi, A. (2006). Optimization: An introduction. 12

Bekker, J. (2012). *Applying the cross-entropy method in multi-objective optimisation of dynamic stochastic systems*. Ph.D. thesis, University of Stellenbosch. 3, 4, 5, 10, 16, 17, 20, 21, 38, 58, 78

Bekker, J. (2019). Introduction to discrete-event simulation. 54

Burkardt, J. (2014). The Truncated Normal Distribution. *Department of Scientific Computing , Florida State University*, 1–35. 4

Bury, K. (2012). Beta Distributions. *Statistical Distributions in Engineering*, **1**, 238–266. 22

Coello, C.A.C. (2002). Theoretical and Numerical Constrain-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. 93. 29, 30, 41, 48, 49

Coello, C.A.C. (2006). Evolutionary Multi-Objective optimization: A Historical View of the Field. *IEEE Computational Intelligence Magazine*, **12**, 66–70. 15, 29

Coello, C.A.C., Lamont, G.B., Veldhuizen, D.A.V., Goldberg, D.E. & Koza, J.R. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems Second Edition Genetic and Evolutionary Computation Series Series Editors Selected titles from this series :*. 38, 39

Fan, Z., Li, W., Cai, X., Li, H., Wei, C., Zhang, Q., Deb, K. & Goodman, E. (2019). Difficulty Adjustable and Scalable Constrained Multiobjective Test Problem Toolkit. *Evolutionary Computation*, 1–40. 2

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-wesley Publishing Company Inc. 19, 21, 34

Igel, C., Hansen, N. & Roth, S. (2007). Covariance matrix adaptation for multi-objective optimization. *Evolutionary Computation*, **15**, 1–28. 3, 5, 21, 27, 28, 46

Jiang, S., Ong, Y.S., Zhang, J. & Feng, L. (2014). Consistencies and contradictions of performance metrics in multiobjective optimization. *IEEE Transactions on Cybernetics*, **44**, 2391–2404. 37, 38, 39

Liefooghe, A. & Derbel, B. (2016). A correlation analysis of set quality indicator values in multiobjective optimization. *GECCO 2016 - Proceedings of the 2016 Genetic and Evolutionary Computation Conference*, 581–588. 40

McClymont, K. & Keedwell, E. (2012). Deductive sort and climbing sort: New methods for non-dominated sorting. *Evolutionary Computation*, **20**, 1–26. 34

Moreno, J., Rodriguez, D., Nebro, A.J., Lozano, J.A. & Member, S. (2020). Merge Non-Dominated Sorting Algorithm for Many-Objective Optimization. *IEEE*, 1–13. 34, 35, 83

Quiza Sardiñas, R., Rivas Santana, M. & Alfonso Brindis, E. (2006). Genetic algorithm-based multi-objective optimization of cutting parameters in turning processes. *Engineering Applications of Artificial Intelligence*, **19**, 127–133. 13

Rice, J.A. (2007). *Mathematical statistics and data analysis*, vol. 34. Thomson Brooks/Cole, 3rd edn. 24, 25

Richardson, J.T., Palmer, M.R., Liepins, G. & Hilliard, M. (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. *Proceedings of the Third International Conerence on Genetic Algorithms*. 30

Riquelme, N., Von Lücken, C. & Barán, B. (2015). Performance metrics in multi-objective optimization. *Proceedings - 2015 41st Latin American Computing Conference, CLEI 2015*. 38

Roy, P.C., Islam, M.M. & Deb, K. (2016). Best order sort: A new algorithm to non-dominated sorting for evolutionary multi-objective optimization. *GECCO 2016 Companion - Proceedings of the 2016 Genetic and Evolutionary Computation Conference*, 1113–1120. 34

Rubinstein, R.Y. & Kroese, D.P. (2004). *The Cross-Entropy Method: A Unfied Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer. 16

Savic, D. (2002). Single-objective vs Multiobjective Optimisation for Integrated Decision Support. *Proceedings of the First Biennial Meeting of the International Environmental Modelling and Software Society*, 7–12. 2, 13

Tessema, B. & Yen, G.G. (2006). A self-adaptive constrained evolutionary algorithm. *Proceedings of the IEEE International Conference on Evolutionary Computation*. 31

Theodossiou, N., Karakatsanis, D. & Kougias, I. (2014). The history of optimization. Applications in water resources management. *Conference of the Hellenic Hydrotechnical Association (H.H.A)*. 13

Venter, G. (2010). Review of Optimization Techniques. *Encyclopedia of Aerospace Engineering*. 13, 15

Wang, G. (2018). Brief History of Optimization. 13

Wang, H. & Yao, X. (2014). Corner sort for pareto-based many-objective optimization. *IEEE Transactions on Cybernetics*, **44**, 92–102. 34

While, L. (2005). A new analysis of the LebMeasure algorithm for calculating hypervolume. *Lecture Notes in Computer Science*, **3410**, 326–340. 39

Woldesenbet, Y.G., Tessema, B.G. & Yen, G.G. (2007). Constraint handling in multi-objective evolutionary optimization. *2007 IEEE Congress on Evolutionary Computation, CEC 2007*, 3077–3084. 4, 29, 30, 31, 33, 50

Wolpert, D.H. & Macready, W.G. (1997). No Free Lunch Theorems for Optimisation. *IEEE Transactions on Evolutionary Computation*, **1**, 287–322. 83

Zhang, X., Tian, Y., Cheng, R. & Jin, Y. (2015). An Efficient Approach to Non-dominated Sorting for Evolutionary Multi-objective. *IEEE Transactions on Evolutionary Computation*, **19**, 1–15. 34, 36, 51

Zitzler, E. (1999). Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. *TIK-SCHRIFTENREIHE*, **30**, 691–698. 14, 19, 39

Appendix A

Performance indicator box-and-whisker plots

The box-whisker plots of the benchmark problems discussed in Chapter 4 are displayed in figures A.1 to A.18. Figures A.1 to A.15 refer to the hyperarea metric, while A.16 to A.18 refer to the ϵ indicator.

A.1 MOO CEM-Beta

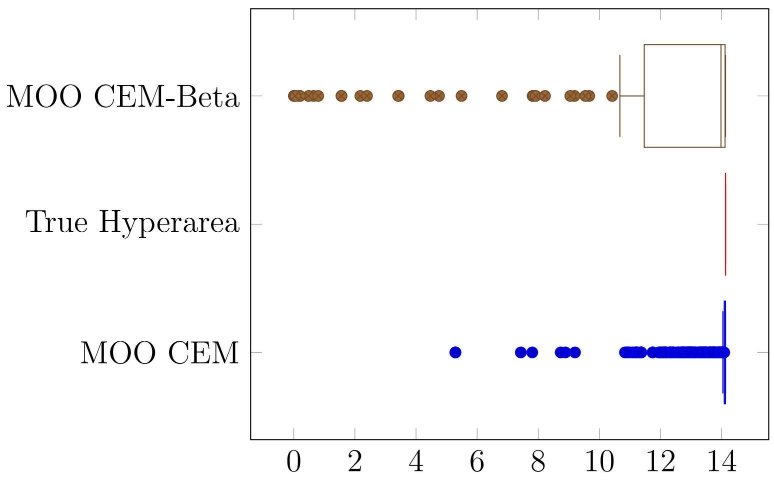


Figure A.1: MOP1 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Beta hyperarea

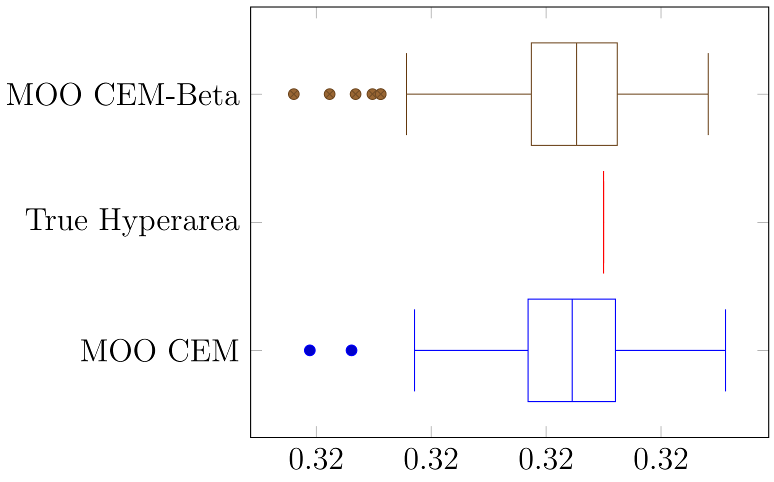


Figure A.2: MOP2 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Beta hyperarea

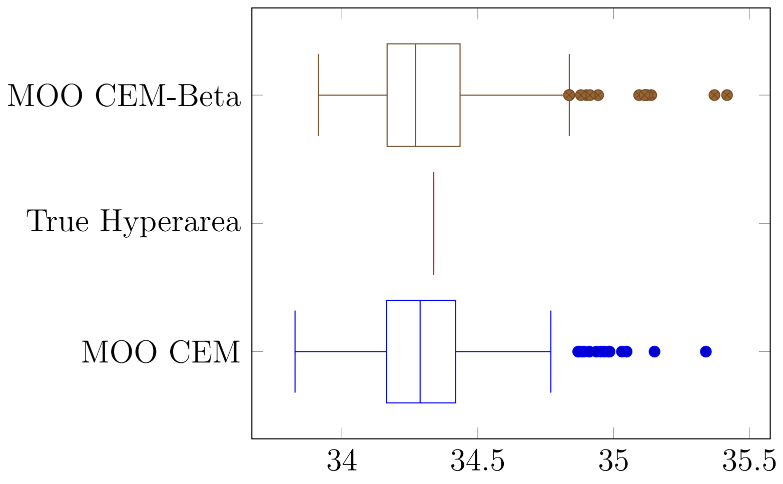


Figure A.3: MOP3 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Beta hyperarea

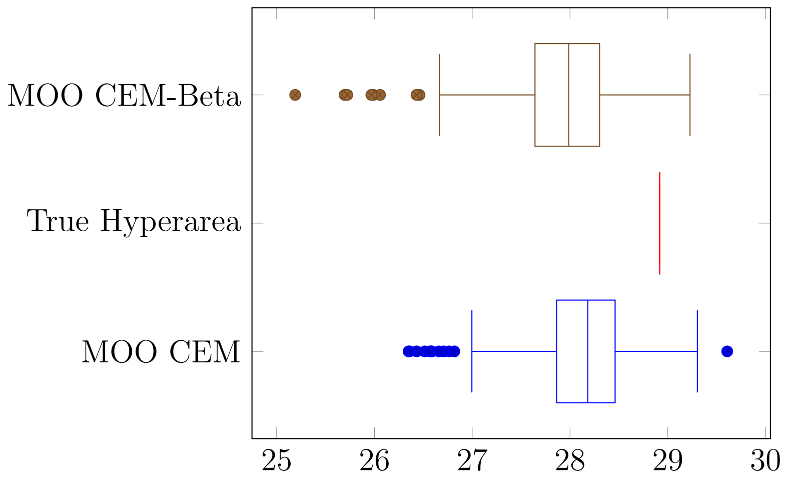


Figure A.4: MOP4 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Beta hyperarea

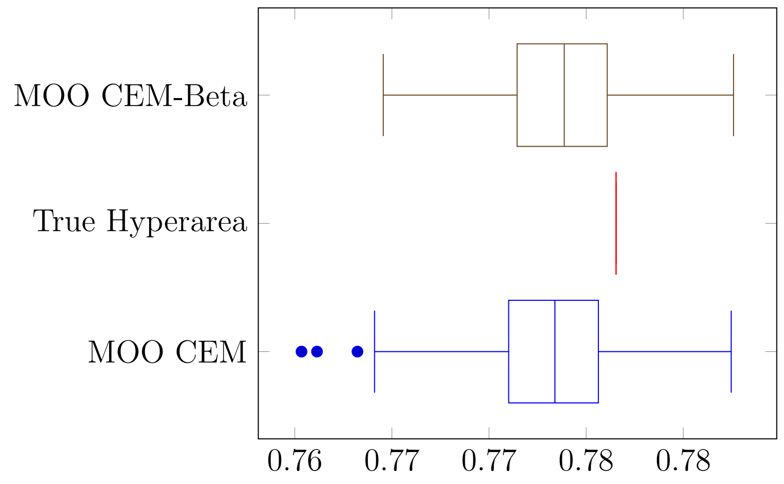


Figure A.5: MOP6 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Beta hyperarea

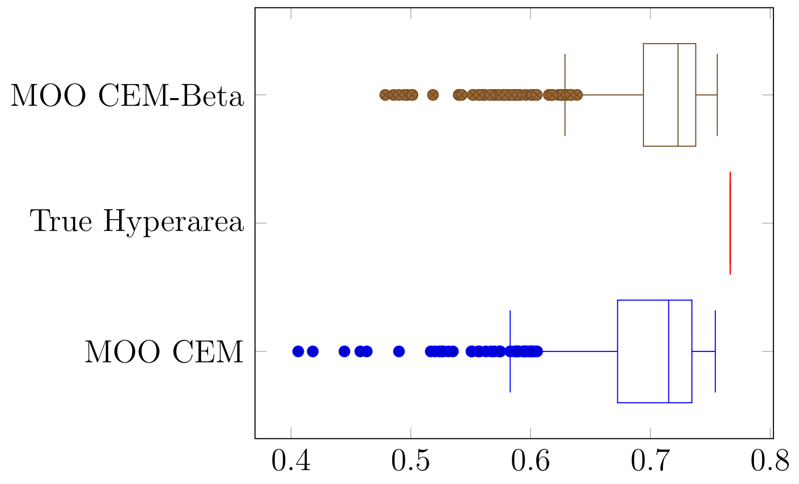


Figure A.6: ZDT1 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Beta hyperarea

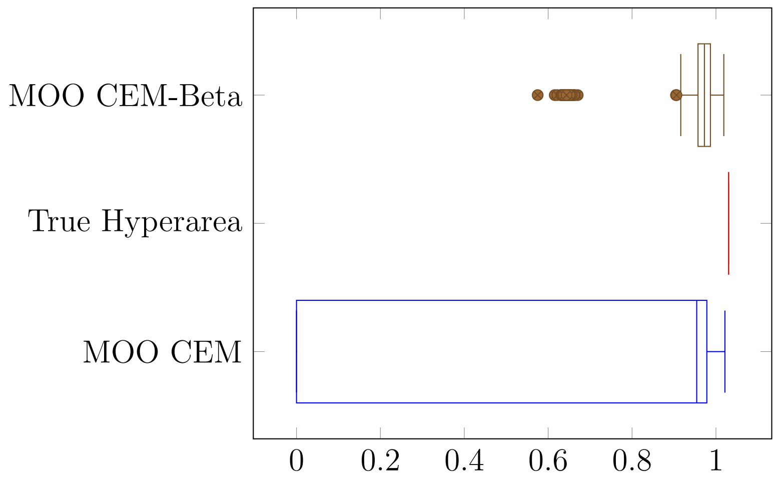


Figure A.7: ZDT2 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Beta hyperarea

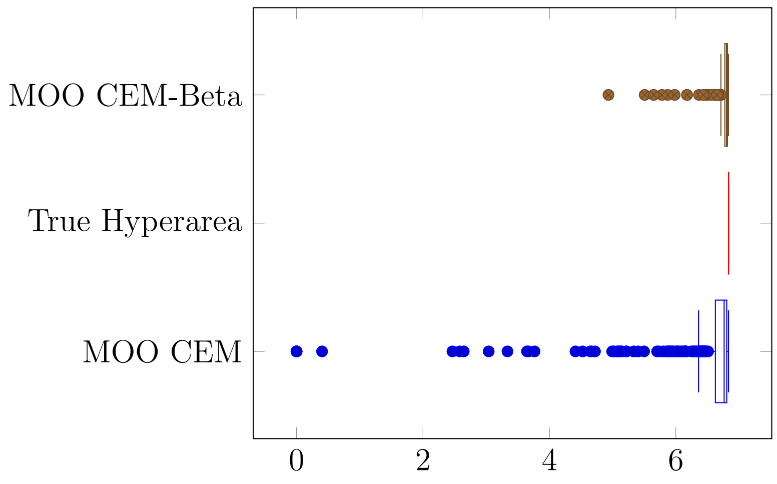


Figure A.8: ZDT3 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Beta hyperarea

A.2 MOO CEM-Cov

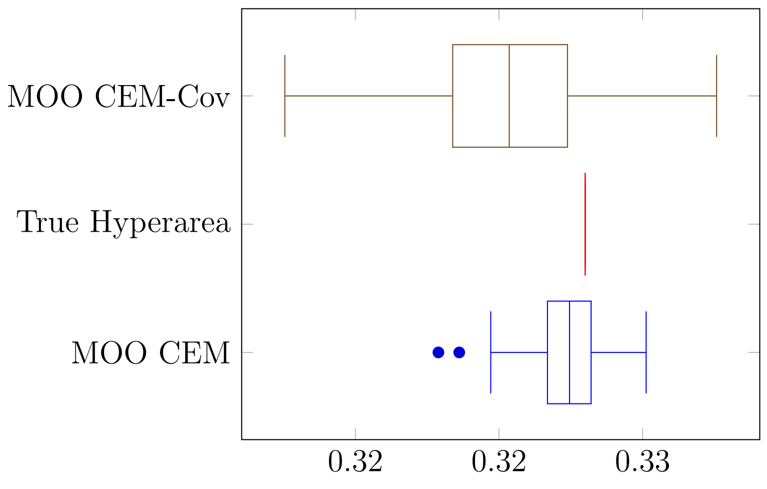


Figure A.9: MOP2 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Cov hyperarea

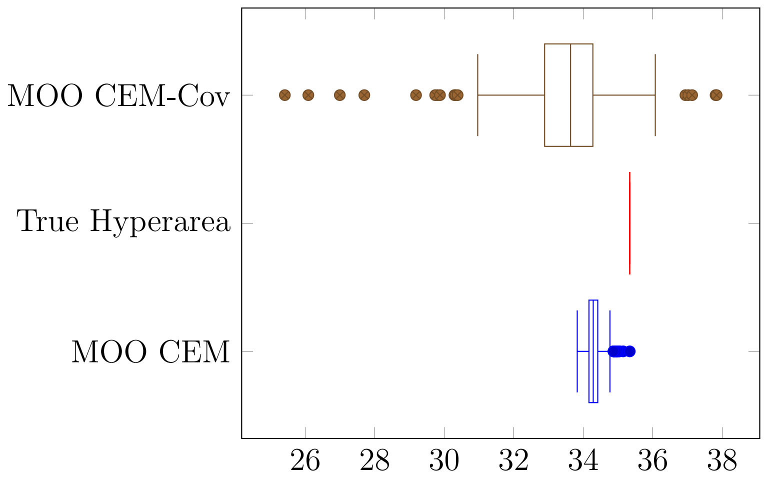


Figure A.10: MOP3 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Cov hyperarea

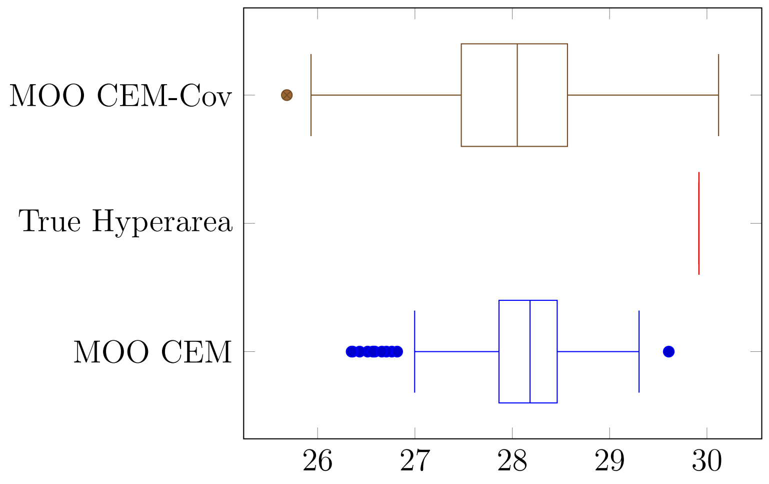


Figure A.11: MOP4 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Cov hyperarea

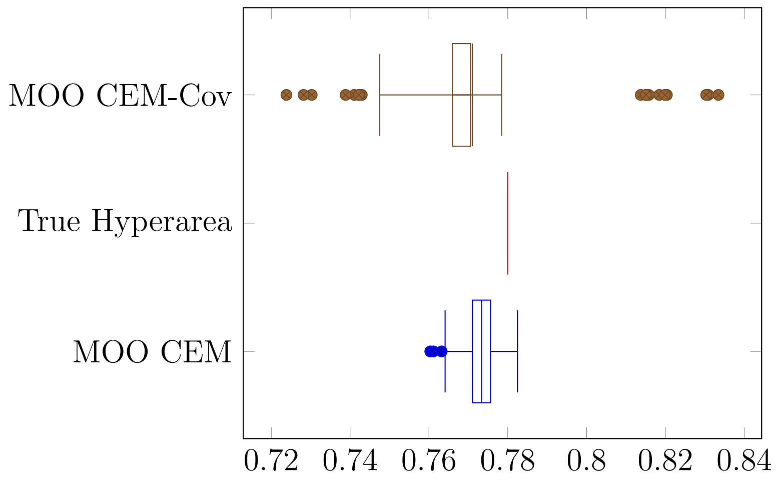


Figure A.12: MOP6 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Cov hyperarea

A.3 MOO CEM-Constraint

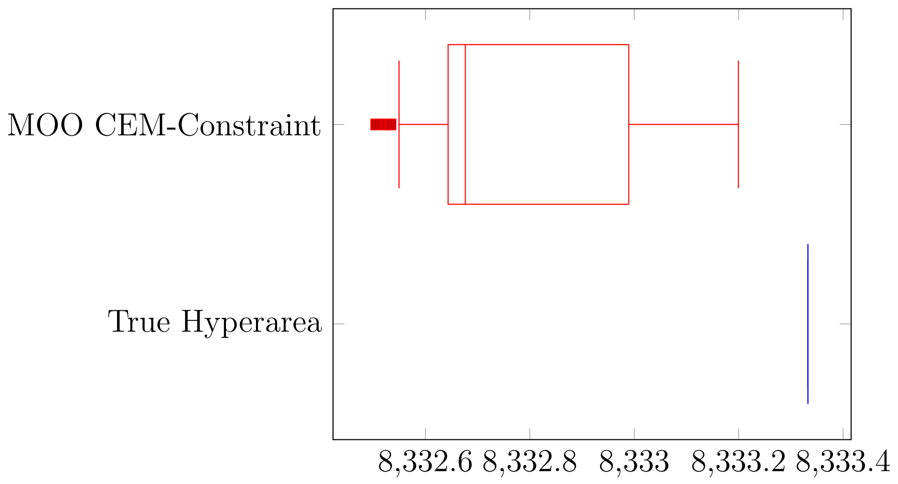


Figure A.13: MOP-C1 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Constraint hyperarea

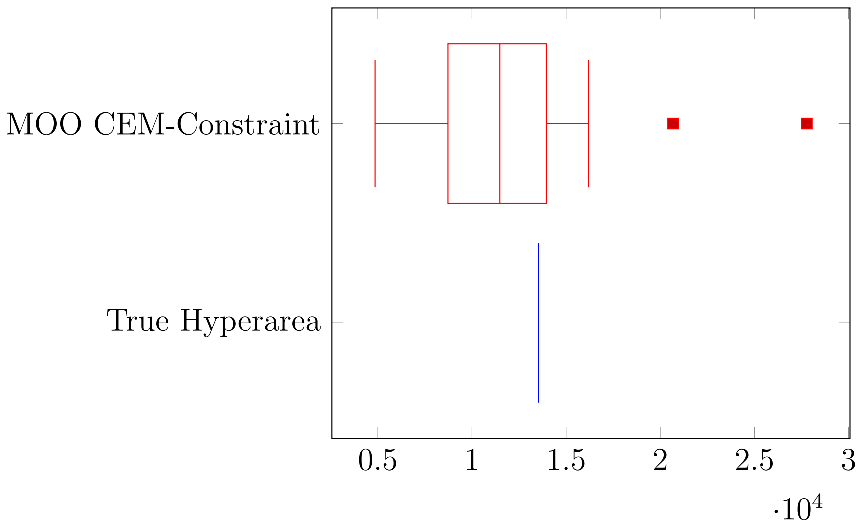


Figure A.14: MOP-C2 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Constraint hyperarea

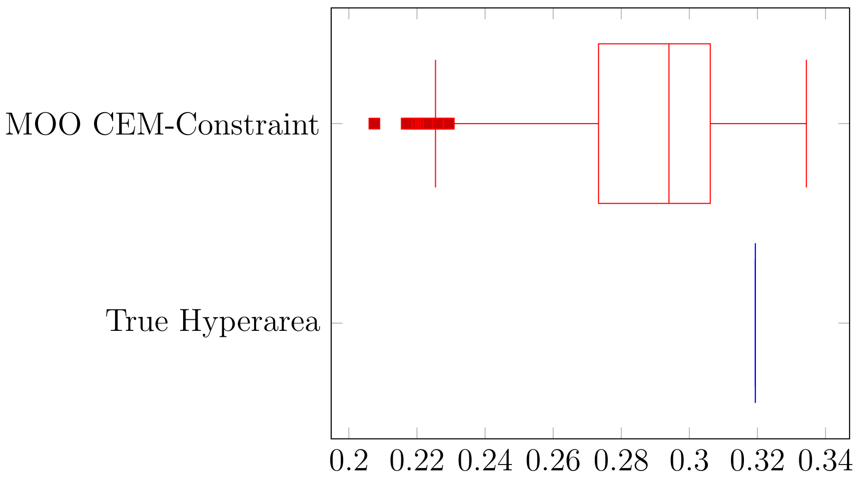


Figure A.15: MOP-C4 box-whisker plot of the MOO CEM algorithm hyperarea compared to MOO CEM-Constraint hyperarea

A.4 MOO CEM-ENS

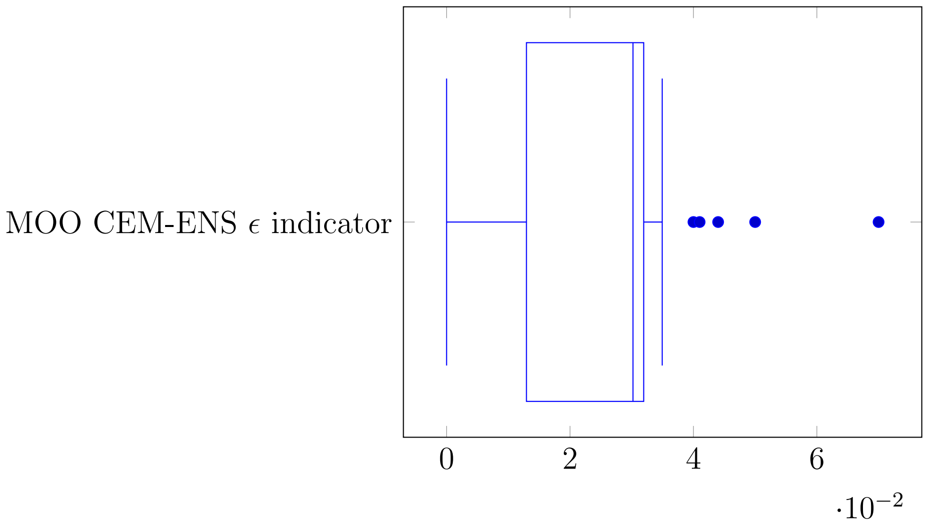


Figure A.16: MOP5 box-whisker plot of the MOO CEM-ENS algorithm ϵ indicator

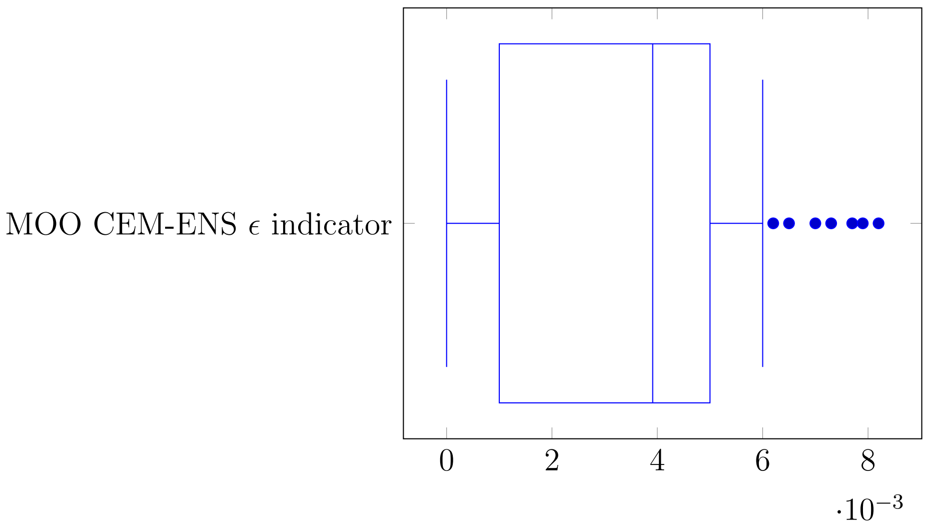


Figure A.17: MOP7 box-whisker plot of the MOO CEM-ENS algorithm ϵ indicator

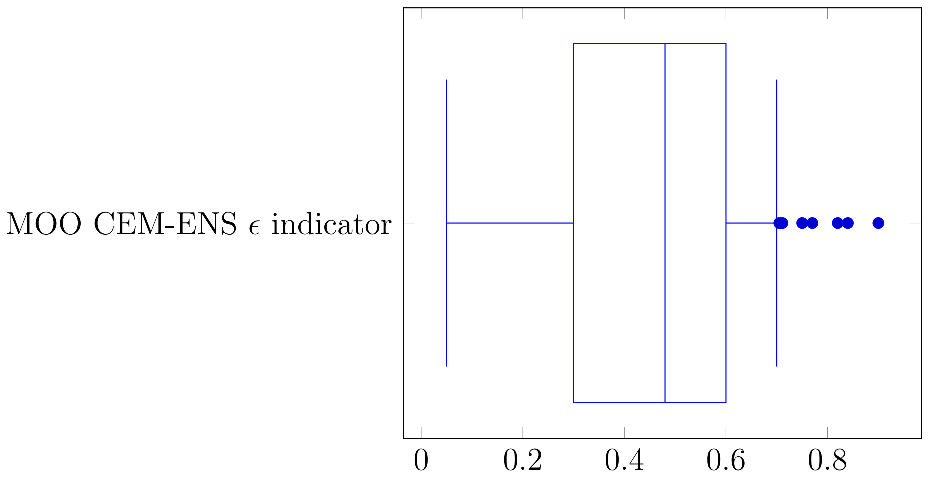


Figure A.18: MOP-C3 box-whisker plot of the MOO CEM-ENS algorithm ϵ indicator

Appendix B

Matlab® code for the enhanced MOO CEM algorithm

Below the Matlab® code for the enhanced MOO CEM algorithm is shown. This code includes all proposed enhancements, including using the Beta distribution to improve sampling, the Penalty method to solve constrained problems and the ENS-SS algorithm to solve problems with more than two objectives. All problems solved as part of this research are pre-coded in the algorithm below. New problems can be solved by adding the problem specifications to the *InitializeProblem* function.

```
global WorkArea
global Elite;
global CM;  global mu; global glue; global SetMu; global SetSigma;

%Choose which MOP to run (refer to InitializeProblem function ...
    for MOP
%specifications
MOP=1;
%Initialise MOP (objective functions, decision variable limits, ...
    etc.)
[NumVars, NumObjectives, Limits, L, SheetName, ProblemN] = ...
     InitializeProblem(MOP);
z=[];
WorkArea = []; mu=[]; Elite=[]; CM=[]; sigma=[]; glue=[]; SetMu=[];
```

```
SetSigma=[];abrec=[];ab=[];AllElite=[];

%Define stopping criteria determined by the threshold
eps = 0.01;
%Choose which percentage of the old and new alpha and beta ...
   parameters are
%used to update the value of alpha and beta
w=0.7;
%Number of outer loops (maximum). Used for MOO
NoOfLoops = 100;
%Population Size
N = 200;
%Number of simulations to run (solve same problem simNum times)
simNum=1;
format long;
HA=0.0000000000000;
simProp= -2.5:1.5;
simProp1=-2.5:1.5;
abrec=zeros(1,6);

%Probability of inverting histograms (frequency of ranges ...
   defined over the
%decision variable space). This avoids getting stuck in a local ...
   optimum and
%diversifies the search.
Prob = 0.3;
%Maximum number of evaluations (to be kept as low as possible to ...
   decrease
%computational time)
MaxEvaluations = 15000;

%Execute for number of simulations specified
for l=1:simNum
    NumEvaluations = 0;
    Elite=[];
    EliteTemp=[];
    time_start=tic; %Start timer to record runtime
    NotTerminate=true;

    %Execute maximum number of loops specified or until stopping ...
       criteria
    %is reached
    for k=1:NoOfLoops
        k;
        %Initialise mean and standard deviation. These are for ...
           reference
        %only and do not influence sampling or stopping criteria.
        sigmaeps(1:NumVars)=Inf;
        %Initially, sigma is large and mu is random (within decision
        %variable limits)
        sigma(1:NumVars) = 10*(L(1:NumVars,2) - L(1:NumVars,1));
        mu(1:NumVars) = (L(1:NumVars,1) + (L(1:NumVars,2) - ...
            L(1:NumVars,1)).*rand(NumVars,1))';
        t=0;
        WorkArea = [];
        NotTerminate = true;

        %Continue executing until stopping criteria is reached ...
           (indicating
        %convergence)
        while NotTerminate
            t=t+1;
            bin_freq = [];
            bin_edges = [];
             if size(Elite,1) > 0 && k>1 %If not the first iteration
               for i=1:NumVars
                   %If the Elite contains only 1 unique ...
                      solution, number of
                   %histograms = 1, otherwise number of loops + 5
                   if min(Elite(:, i))==max(Elite(:, i))
                       r=2;
                   else
                       r = k + 5;
                   end
                   bin_freq = [];
                   bin_edges = [];
                   bin_edges(1:r+1) = 0;
                   bin_edges(1) = L(i, 1); %First histogram ...
                      range starts at
                                           %top limit of ...
                                              decision space
                   bin_edges(r+1) = L(i, 2);%Last histogram ...
                      range ends at
                                            %bottom limit of ...
                                               decision space
                   bin_edges(2) = min(Elite(:, i));
                   bin_edges(r) = max(Elite(:, i));

                   %Assign other histogram range values. If ...
                      Elite has more
                   %than 3 values, split remaining range equally.
                   if bin_edges(2)≠bin_edges(r)
                      bin_edges(2:r)= ...
                          bin_edges(2):max((bin_edges(r) - ...
                          bin_edges(2))/(r-2),0):bin_edges(r);
                   end
                   %Edge vector must be monotonically ...
                      non-decreasing.
```

```
for m=1:r
     if bin_edges(m+1)<bin_edges(m)
         bin_edges(m+1)=bin_edges(m);
     end
end
%Allocate frequencies to each range to create ...
   histogram
try
bin_freq(1:r)   = histc(Elite(:,i), ...
   bin_edges(1:r));
catch err
    bin_freq
end
%Invert the histogram if random value is less ...
   than Prob
if rand<Prob
   bin_freq(1:r) = max(bin_freq) - bin_freq(1:r);
end
%Plot (for reference only)
if i==200
dn=subplot(2,2,4);
bar(bin_freq(1:r))
[d, I] = max(bin_freq);
[bin_edges(I), bin_edges(I+1)];
end

bin_freq = floor(N*bin_freq./sum(bin_freq));
s = sum(bin_freq(1:r));
bin_freq(r) = N - sum(bin_freq(1:r)) + ...
   bin_freq(r);
s=sum(bin_freq);

UpTo=0;  %Indices into WorkArea
%Sample per histogram range (number of sample ...
   according
%to assigned frequency per range).
for a=1:r
    Start = UpTo + 1;
    UpTo  = UpTo + bin_freq(a);
    Limits(i, 1) = bin_edges(a);
    Limits(i, 2) = bin_edges(a+1);
    if Start ≤ UpTo
       %Determine which Elite values fall ...
          within the
       %current histogram range. These values ...
          are used
       %to determine alpha and beta of the Beta
       %distribution
       correspondingElite=Elite(Elite(:,i) ≥ ...
                          Limits(i,1) & Elite(:,i) ≤ ...
                             Limits(i,2), i);
                      if size(correspondingElite,1)<2 || ...
                          size(unique(correspondingElite),2)==1
                          %Beta distribution requires 2 ...
                             different
                          %points to estimate alpha and ...
                             beta. Use a
                          %uniform distribution if not ...
                             enough samples
                          %fall within the current range.
                          alpha=1;
                          beta=1;
                      else
                          %Calculate alpha and beta
                          params=mle(max((correspondingElite-...
                                Limits(i, 1))/(Limits(i, 2)-...
                                Limits(i, ...
                                   1)),0),'distribution',...
                                  'beta');
                          alpha=params(1);
                          beta=params(2);
                      end
                      %Sample points from Beta distribution ...
                         of current
                      %histogram using calculated alpha and ...
                         beta.
                      WorkArea(Start:UpTo, i)=(icdf('beta',...
                          rand(UpTo-Start+1,1),alpha,beta)*...
                          (Limits(i, 2)-Limits(i, ...
                             1)))+Limits(i, 1);
                      ab=[l,k,t,i,alpha,beta];
                      abrec=vertcat(abrec,ab);
                   end
               end
           end
         else %Build initial population (for first iteration ...
            - uniform
             %distribution).
          for i=1:NumVars
           alpha=1;
           beta=1;
           WorkArea(1:N, ...
              i)=(icdf('beta',rand(N,1),alpha,beta)*...
               (L(i, 2)-L(i, 1))) +L(i, 1);
          end
         end

        s = size(WorkArea, 1);
```

```
WorkArea(1:s, NumVars+NumObjectives+2) = zeros(s, 1);
% +2 is for ranking and distance
%f1 returns [f1, normalised f1]. Store f1 in last ...
    columns for
%plotting. Normalised f1 is used for constrained ...
    problems.
[WorkArea(1:s, NumVars+NumObjectives+2+1),...
    WorkArea(1:s, NumVars+1)] = f1(WorkArea, ...
        NumVars, MOP);

if MOP<15 || MOP≥20
 %f2 returns [f2, normalised f2]. Store f2 in last ...
     columns
 %for plotting. Normalised f2 is used for ...
     constrained problems.
  [WorkArea(1:s, NumVars+NumObjectives+2+2),...
      WorkArea(1:s, NumVars+2)] = f2(WorkArea, ...
          NumVars, MOP);
end

if MOP==5 || MOP==7 || MOP == 13 || MOP == 14 || MOP ...
    == 22
   %Only these problems have 3 objectives. f3 returns
   %[f3, normalised f3]. Store f3 in last columns ...
       for plotting.
   %Normalised f3 is used for constrained problems.
   [WorkArea(1:s, ...
       NumVars+NumObjectives+2+3),WorkArea(1:s, ...
        NumVars+3)] = f3(WorkArea, NumVars, MOP);
end

%%%%%%%%%%%%%%%%% CONSTRAINT PENALTY %%%%%%%%%%%%%%%%%
%Determine which samples are feasible and calculate ...
    rf and
%penalty.
if MOP≥20 && MOP≤26
    [feasible,rf]=CountFeasible(WorkArea,MOP,NumVars,...
        NumObjectives,ETA,SIGMA,GAMMA);
    rf=rf/N;
    Penalty = CalculatePenalty(WorkArea,MOP,NumVars,...
        NumObjectives,ETA,SIGMA,GAMMA);
    if rf==0
        dist=ones(N,NumObjectives).*Penalty;
        Xi=zeros(N,NumObjectives);
    else
        dist=sqrt((WorkArea(:,NumVars+1:NumVars+...
            NumObjectives)).^2+(Penalty).^2);
        %need xi for each objective : times penalty ...
            by 1 for N
        Xi=ones(N,NumObjectives).*Penalty;
    end

    Yi=feasible.*(WorkArea(:,NumVars+1: ...
        NumVars+NumObjectives));

    %Add the two penalties to the normalised objective
    %functions.
    for i=1:NumObjectives
        p(:,i)=(1-rf).*Xi(:,i)+rf.*Yi(:,i);
        WorkArea(:,NumVars+i)=dist(:,i)+p(:,i);
    end
end

%Sort the samples according to all normalised ...
   objectives using
%ENS-SS. Note: relaxed value of theta=3 to keep ...
   solutions for
%diversity.
Temp = ENSort(WorkArea, 3, NumVars, NumObjectives);

EliteTemp=vertcat(EliteTemp, Temp);
Elite = vertcat(Elite, Temp);

%%%%%%%%%%%%%%%%%%%%%UNCOMMENT THIS FOR ...
   PLOT%%%%%%%%%%%%%%%%%%
PlotDetailProgress(MOP, NumVars, NumObjectives, ...
   WorkArea, ...
ProblemN, NoOfLoops, t, k, N)

%Determine if stopping criteria is reached. If alpha ...
   and beta
%do not change, solution is assumed to have converged.
if size(Elite,1) > 1
    AllEps = 0;
    for i=1:NumVars
        %mu and sigma are updated for reference only.
        mu(i)         = (1-w)*mu(i)      + ...
           w*mean(Elite(:,i));
        sigmaeps(i)  = sigma(i);
        sigma(i)     = (1-w)*sigma(i)  + ...
           w*std(Elite(:,i));
        sigmaeps(i)  = abs(sigma(i) - sigmaeps(i));

        %Calculate total difference between alpha ...
           and beta for
        %each variable (mean per variable, not one per
        %histogram range).
        alpha_curr=mean(abrec(abrec(abrec(:,2)==k,4)...
                ==i,5));
            beta_curr=mean(abrec(abrec(abrec(:,2)==k,4)...
                ==i,6));
            alpha_prev=mean(abrec(abrec(abrec(:,2)==k-1,4)...
                ==i,5));
            beta_prev=mean(abrec(abrec(abrec(:,2)==k-1,4)...
                ==i,6));
            AllEps = abs(alpha_curr-alpha_prev)+...
                abs(beta_curr-beta_prev);
        end

        if AllEps == 0
            %If alpha and beta have not changed for any ...
               variable
            %since the previous iteration, solution has ...
               converged
            %and algorithm terminates.
            NotTerminate = false;
        end
        %SetMu = vertcat(SetMu, mu);
        %SetSigma = vertcat(SetSigma, sigma);
    end

    NotTerminate = (NotTerminate && (N*t ≤ ...
        MaxEvaluations/2));

    %Algorithm terminates if solution has converged.
    if ¬NotTerminate, break, end

    NumEvaluations = NumEvaluations + N;
    %Algorithm terminates if maximum number of ...
        evaluations is
    %reached.
    if  (NumEvaluations ≥ MaxEvaluations), break, end
end  %while not Terminate

%Sort the samples according to all normalised objectives ...
    using
%ENS-SS. Note: relaxed value of theta=2 to keep ...
    solutions for
%diversity.
Elite=ENSort(Elite, 2, NumVars, NumObjectives);

%%%%%%%%%%%%%%%%%UNCOMMENT THIS FOR ...
    PLOT%%%%%%%%%%%%%%%%%%%%%%%%%%%
i = subplot(2,2,2);
hold on
if MOP≥20
    scatter(Elite(:,NumVars+1+2+NumObjectives), ...
            Elite(:, NumVars+2+2+NumObjectives), 3, '*');
    else
        scatter(Elite(:,NumVars+1), Elite(:, NumVars+2), 3, ...
            '*');
    end
    xlabel('f1')
    ylabel('f2')
    title(['Current Pareto front, after iteration ' ...
        int2str(t) ...
        ' ' int2str(NumEvaluations)])
    drawnow
    grid on
    hold off

    if (NumEvaluations ≥ MaxEvaluations)
        break
    end
end % for k

EliteWithNorm=Elite;
Elite(:,NumVars+1:NumVars+NumObjectives)=Elite(:,NumVars+...
    NumObjectives+2+1:NumVars+NumObjectives+2+NumObjectives);
%Sort the samples according to all original objectives using ...
   ENS-SS.
%This ensures that all final solutions are feasible and best ...
   according
%to all objetcives.
Elite=ENSort(Elite, 1, NumVars, NumObjectives);

%Record end time.
time_end=toc(time_start);
%Calculate hyperarea.
if MOP==1
    HA=hyperArea2([Elite(:,2),Elite(:,3)]);
elseif MOP==2
    HA=hyperArea2([Elite(:,4),Elite(:,5)]);
elseif MOP==3
    HA=hyperArea2([Elite(:,3),Elite(:,4)]);
elseif MOP==4
    HA=hyperArea2([Elite(:,4),Elite(:,5)]);
elseif MOP==5
    HA=hyperArea2([Elite(:,3),Elite(:,4)]);
elseif MOP==6
    HA=hyperArea2([Elite(:,3),Elite(:,4)]);
elseif MOP==8
    HA=hyperArea2([Elite(:,31),Elite(:,32)]);
elseif MOP==9
    HA=hyperArea2([Elite(:,31),Elite(:,32)]);
elseif MOP==10
```

```
        HA=hyperArea2([Elite(:,31),Elite(:,32)]);
    elseif MOP==20
        HA=hyperArea2([Elite(:,3),Elite(:,4)],[200,50]);
    elseif MOP==21
        HA=hyperArea2([Elite(:,7),Elite(:,8)],[0,80]);
    elseif MOP==23
        HA=hyperArea2([Elite(:,3),Elite(:,4)],[3.3,3.3]);
    end
    %%%%%%%%%%%%%%%%%UNCOMMENT THIS FOR ...
       PLOT%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if MOP ≤ 6 || (MOP≥8 && MOP ≤12)
       Plot_WorkArea(Elite(:,NumVars+1), Elite(:,NumVars+2), ...
          MOP, ...
           SheetName, NumEvaluations)
    elseif MOP≥20 && MOP≠22
        Plot_WorkArea(Elite(:,NumVars+1+2+NumObjectives), ...
           Elite(:,...
            NumVars+2+2+NumObjectives), MOP, SheetName, ...
               NumEvaluations)
    else
       scatter3(Elite(:,NumVars+1), Elite(:,NumVars+2), ...
           Elite(:,NumVars+3), 2.4, '*');
    end
simProp=[double(l),double(time_end),double(size(Elite,1)),...
    double(HA),NumEvaluations];
simProp1=vertcat(simProp1,simProp);
AllElite=vertcat(AllElite,[l+zeros(size(Elite,1),1),Elite]);
end
%Record all generated data (final solutions, objective functions ...
   and values
%of alpha and beta.
%writematrix(simProp1,strcat('MOOCEMBeta_MOP',int2str(MOP),...
%'_HyperArea_TEST2.csv'));
%writematrix(AllElite,strcat('MOOCEMBeta_MOP',int2str(MOP),...
%'_Elite_TEST2.csv'));
%writematrix(abrec,strcat('MOOCEMBeta_MOP',int2str(MOP),...
%'_AlphaBeta_TEST2.csv'));

%Evaluate first objective function. Also calculate normalised ...
   objective
%function value for constrained problems.
function [Do_f1, Do_f1_norm] = f1(X, NVars, MOP)
    Do_f1_norm=X.*0;
    if MOP == 1
        c=cos(3*pi/4);
        c=1;
       Do_f1=(X(:,1)*c).^2;
    elseif MOP == 2
       rt = 1/sqrt(NVars);
       Do_f1=1-exp(-((X(:,1)-rt).^2+(X(:,2)-rt).^2+(X(:,3)-rt).^2));
    elseif MOP == 3
       A1 = 0.5*sin(1)-2*cos(1)+  sin(2) - 1.5*cos(2);
       A2 = 1.5*sin(1)-  cos(1)+2*sin(2) - 0.5*cos(2);
       B1 = 0.5*sin(X(:,1)) - 2* cos(X(:,1)) +   sin(X(:,2)) - ...
           1.5*cos(X(:,2));
       B2 = 1.5*sin(X(:,1)) -    cos(X(:,1)) + 2*sin(X(:,2)) - ...
           0.5*cos(X(:,2));
       Do_f1 = -(1 + (A1 - B1).^2 + (A2 - B2).^2);
    elseif MOP == 4
       Do_f1 = -10*(exp(-0.2*sqrt(X(:,1).^2 + X(:,2).^2)) + ...
           exp(-0.2*sqrt(X(:,2).^2 + X(:,3).^2)));
    elseif MOP ==5
       Do_f1 = 0.5*(X(:,1).^2 + X(:,2).^2) + sin(X(:,1).^2 + ...
          X(:,2).^2);
    elseif MOP == 6
       Do_f1=X(:,1);
    elseif MOP == 7
       Do_f1 = 0.5*(X(:,1) - 2).^2 +(1/13)*(X(:,2) + 1).^2 + 3;
    elseif MOP ≥ 8 && MOP ≤ 11
       Do_f1 = X(:,1);  %ZDT1, 2 & 3 & 4
    elseif MOP==12  %ZDT6
       SixPi = 6*pi;
       Do_f1 = 1 - exp(-4*X(:,1)).*sin(SixPi*X(:,1)).^6;
    elseif MOP == 13
       Do_f1 = cos(pi/12)*X(:,1) - sin(pi/12)*X(:,2);
    elseif MOP == 14
        S=0;
        for i=1:NVars-1
            S = S + 100*(X(:,i+1) - X(:,i).^2).^2 + ((X(:,i) - ...
                1).^2);
        end
        Do_f1 = S;
    elseif MOP == 15
        TwentyPi = 20*pi;
        gxM = 0;  %zeros(1:size(X,1),1:NVars);
        for i=3:NVars
            gxM = gxM + ((X(:,i)-0.5).^2) - ...
                cos(TwentyPi*(X(:,i)-0.5));
        end
        gxM = 100*(gxM + 5);
        Do_f1 = 0.5*X(:,1).*X(:,2).*(1+gxM);
    elseif MOP == 16
        y = (X(:,1).^2);
        Do_f1=2*28.3*X(:,2).*sqrt(1+y);
    elseif MOP==20
        Do_f1=4*(X(:,1).^2)+4*(X(:,2).^2);
        %Normalise
        f1_max=200;
```

```
        f1_min=0;
        Do_f1_norm=(Do_f1-f1_min)/(f1_max-f1_min);
    elseif MOP==21
        Do_f1=-(25*((X(:,1)-2).^2)+((X(:,2)-2).^2)+((X(:,3)-1).^2)+...
            ((X(:,4)-4).^2)+((X(:,5)-1).^2));
        %Normalise
        f1_max=0;
        f1_min=-1712;
        Do_f1_norm=(Do_f1-f1_min)/(f1_max-f1_min);
    elseif MOP==22
        Do_f1=((X(:,1)-2).^2)/2 +((X(:,2)+1).^2)/13 + 3;
        %Normalise
        f1_max=(6^2)/2+(5^2)/13+3;
        f1_min=3;
        Do_f1_norm=(Do_f1-f1_min)/(f1_max-f1_min);
    elseif MOP==23
        Do_f1=X(:,1);
        %Normalise
        f1_max=pi;
        f1_min=0.0000001;
        Do_f1_norm=(Do_f1-f1_min)/(f1_max-f1_min);
    end
end
%Evaluate second objective function. Also calculate normalised ...
   objective
%function value for constrained problems.
function [Do_f2, Do_f2_norm]= f2(X, NVars, MOP)
    Do_f2_norm=X.*0;
    if MOP == 1
         c=cos(3*pi/4);
         c=1;
       Do_f2=((X(:,1)-2)*c).^2;
    elseif MOP == 2
        rt = 1/sqrt(NVars);
        Do_f2=1-exp(-((X(:,1)+rt).^2+(X(:,2)+rt).^2+(X(:,3)+rt).^2));
    elseif MOP == 3
       Do_f2 = -((X(:,1) + 3).^2 + (X(:,2) + 1).^2);
    elseif MOP == 4
       Do_f2   = abs(X(:,1)).^(0.8)+ 5*sin((X(:,1)).^3) + ...
           abs(X(:,2)).^(0.8) + 5*sin((X(:,2)).^3) + ...
              abs(X(:,3)).^(0.8) ...
           + 5*sin((X(:,3)).^3);
    elseif MOP == 5
       Do_f2 = ((3*X(:,1) -2*X(:,2) + 4).^2)/8  +   ...
           ((X(:,1) - X(:,2) + 1).^2)/27 + 15;
    elseif MOP == 6
        x=X(:,1)./(1+10*X(:,2));
        y=x;
        x=x.^2;
        x=1-x;
        Do_f2=(1+10*X(:,2)).*(x - y.*sin(12*pi*X(:,1)));
    elseif MOP == 7
        Do_f2 = (1/36)*(X(:,1) +X(:,2) - 3).^2 + ...
            0.125*(-X(:,1) + X(:,2) + 2).^2 - 17;
    elseif MOP ≥ 8 && MOP ≤ 10 %ZDT1-3
        c = 9/(NVars-1);
        x = transpose(sum(transpose(X(:,2:NVars))));
        gx = 1 + x.*c;
        gx_inv = 1./gx;
        if MOP == 8     %ZDT1
            Do_f2 = gx.*(1 - sqrt(gx_inv.*X(:,NVars+1)));
        elseif MOP == 9 %ZDT2
            Do_f2 = gx.*(1 - (gx_inv.*X(:,NVars+1)).^2);
        elseif MOP == 10 %ZDT3
            Ten_Pi = 10*pi;
            Do_f2 = gx.*(1 - sqrt(gx_inv.*X(:,1)) - ...
                gx_inv.*X(:,1).*sin(Ten_Pi*X(:,1)));
        end
    elseif MOP == 11  %ZDT4
        gx = 1 + 10*(NVars-1) + sum(X(:,2:NVars).^2 - ...
            10*cos(4*pi*X(:,2:NVars)),2);  %The "2" is to add ...
                columns
        %gx=2;
        gx_inv = 1./gx;
        Do_f2 = gx.*(1 - sqrt(gx_inv.*X(:,NVars+1)));  %NVars+1 ...
           is f1
    elseif MOP==12  %ZDT6
        gx = 1 + ...
           (NVars-1)*(1/(NVars-1)*sum(X(:,2:NVars),2)).^(0.25);
        %gx=1;
        gx_inv = 1./gx;
        Do_f2 = gx.*(1 - (gx_inv.*X(:,NVars+1)).^2);
    elseif MOP == 13
        x1 = cos(pi/12)*X(:,1) - sin(pi/12)*X(:,2);
        x2 = sin(pi/12)*X(:,1) + cos(pi/12)*X(:,2);
        Do_f2 = sqrt(2*pi)*ones(size(X,1),1) - sqrt(abs(x1)) + ...
            2*sqrt(abs(x2 - 3*cos(x1) - 3));
    elseif MOP == 14
       Do_f2 = X(:,NVars + 1);
    elseif MOP == 15
        TwentyPi = 20*pi;
        gxM = 0;  %zeros(1:size(X,1),1:NVars);
        for i=3:NVars
            gxM = gxM + ((X(:,i)-0.5).^2) - ...
                cos(TwentyPi*(X(:,i)-0.5));
        end
        gxM = 100*(gxM + 5);
        Do_f2 = 0.5*X(:,1).*(1-X(:,2)).*(1+gxM);
    elseif MOP == 16
        y = X(:,1).^2;
        s = (2*sqrt(2)*3E7*y.*X(:,2));
        z=0;
        for i =1:size(X,1)
            z(i,1) = ((1+X(i,1)^2)^1.5)*((1 + ...
               X(i,1)^4).^0.5)/s(i,1);
        end
        Do_f2 = 1E6*z;
    elseif MOP==20
        Do_f2=((X(:,1)-5).^2)+((X(:,2)-5).^2);
        %Normalise
        f2_max=50;
        f2_min=0;
        Do_f2_norm=(Do_f2-f2_min)/(f2_max-f2_min);
    elseif MOP==21
        Do_f2=(X(:,1).^2)+(X(:,2).^2)+(X(:,3).^2)+(X(:,4).^2)+...
            (X(:,5).^2)+(X(:,6).^2);
        %Normalise
        f2_max=386;
        f2_min=2;
        Do_f2_norm=(Do_f2-f2_min)/(f2_max-f2_min);
    elseif MOP==22
        Do_f2=((X(:,1)+X(:,2)-3).^2)/175 ...
           +(2*X(:,2)-X(:,1)).^2/17 - 13;
        %Normalise
        f2_max=((-4-4-3).^2)/175 +(2*(-4)-(+4)).^2/17 - 13;
        f2_min=-13;
        Do_f2_norm=(Do_f2-f2_min)/(f2_max-f2_min);
    elseif MOP==23
        Do_f2=X(:,2);
        %Normalise
        f2_max=pi;
        f2_min=0.0000001;
        Do_f2_norm=(Do_f2-f2_min)/(f2_max-f2_min);
    end
end
%Evaluate third objective function. Also calculate normalised ...
   objective
%function value for constrained problems.
function [Do_f3,Do_f3_norm] = f3(X, NVars, MOP)
    Do_f3_norm=X.*0;
    if MOP ==5
       x = X(:,1).^2;
       y = X(:,2).^2;
       Do_f3 = (1./(x + y + 1)) - 1.1*exp(-(x + y));
    elseif MOP == 7
       Do_f3 = (1/175)*(X(:,1) + 2*X(:,2) -3).^2 + ...
           (1/17)*(2*X(:,2) - X(:,1)).^2 - 13;
    elseif MOP==13
        TwentyPi = 20*pi;
        gxM = 0;  %zeros(size(X,1),NVars);
        for i=3:NVars
            gxM = gxM + ((X(:,i)-0.5).^2) - ...
                cos(TwentyPi*(X(:,i)-0.5));
        end
        gxM = 100*(gxM + 5);
        Do_f3 = 0.5*(1-X(:,1)).*(1+gxM);
    elseif MOP==22
        Do_f3=((3*X(:,1)-2*X(:,2)+4).^2)/8 ...
            +(X(:,1)-X(:,2)+1).^2/27 +15;
        %Normalise
        f3_max=((3*4-2*(-4)+4).^2)/8 +(4-(-4)+1).^2/27 +15;
        f3_min=15;
        Do_f3_norm=(Do_f3-f3_min)/(f3_max-f3_min);
    end
end
%Function to plot solutions vs true Pareto front.
function PPF = Plot_WorkArea(x,y, MOP, xlSheetName, NEval)
subplot(2,2,4);
hold on;
if MOP ≤ 4 || (MOP ≥ 6 && MOP < 11)
    z=[];
    z = xlsread('True_PFs_Coello.xls', xlSheetName);
    scatter(z(:,1), z(:,2), 5, 'v', 'filled');
end
scatter(x, y, 3, 'o');
hold off
grid on
xlabel('f1');
ylabel('f2');
title(['Final Pareto Front of MOP', int2str(MOP), ' after ', ...
    int2str(NEval), ' evaluations']);
end
%Calculate penalty value for infeasible solutions
%(for constrained problems).
function Penalty = CalculatePenalty(Elite,MOP, NVars, ...
    NObj,ETA,SIGMA,GAMMA)
    if MOP==20
        Violation1=max(((Elite(:,1)-5).^2) + (Elite(:,2).^2) -25,0);
        %normalise: violation / max violation
        Violation1=Violation1/(25);
        Violation2=max(-((Elite(:,1)-8).^2)-((Elite(:,2)+3).^2)+...
            7.7,0);
        Violation2=Violation2/65.3;
        %contraint 2 can never be violated given x and y
        TotalViolation=(1/2)*(Violation1+Violation2);
     elseif MOP==21
```

```
        Violation1=abs(min(Elite(:,1) + Elite(:,2) -2, 0));
        Violation1=Violation1/(2);
        Violation2=abs(min(6-Elite(:,1) - Elite(:,2),0));
        Violation2=Violation2/(14);
        Violation3=abs(min(2-Elite(:,2) + Elite(:,1),0));
        Violation3=Violation3/(8);
        Violation4=abs(min(2-Elite(:,1) + (3*Elite(:,2)),0));
        Violation4=Violation4/(8);
        Violation5=abs(min(2-Elite(:,1) + (3*Elite(:,2)),0));
        Violation5=Violation5/(6);
        Violation6=abs(min(2-Elite(:,1) + (3*Elite(:,2)),0));
        Violation6=Violation6/(4);
        TotalViolation=(1/6)*(Violation1+Violation2+Violation3+...
            Violation4+Violation5+Violation6);
    elseif MOP==22
        Violation1=max(Elite(:,2) + 4*(Elite(:,1)) -4,0.000001);
        Violation1=Violation1/(16);
        Violation2=max(-Elite(:,1) -1,0.000001);
        Violation2=Violation2/(3);
        Violation3=max(Elite(:,1) - 2 - Elite(:,2),0.000001);
        Violation3=Violation3/(6);
        TotalViolation=(1/3)*(Violation1+Violation2+Violation3);
    elseif MOP==23
        a=0.1;
        b=16;
        TotalViolation=max(-(Elite(:,1).^2) -(Elite(:,2).^2) + 1 ...
           + ...
            (a*cos(b*atan(Elite(:,1)./Elite(:,2)))), 0);
        TotalViolation=TotalViolation/(1.1);
    end
    Penalty=TotalViolation;
end
%Find feasible solutions and calculate rf (for constrained ...
   problems).
function ...
   [feasible,Rf]=CountFeasible(Elite,MOP,NVars,NObj,ETA,SIGMA,GAMMA)
    if MOP==20
        Constraint1=Elite(((Elite(:,1)-5).^2) +
        %(Elite(:,2).^2) -25 ≤ 0,:);
        Constraint=Constraint1(-((Constraint1(:,1)-8).^2)-...
            ((Constraint1(:,2)+3).^2)+7.7≤0,:);
        Rf=size(Constraint,1);
        feasible=((Elite(:,1)-5).^2) + (Elite(:,2).^2) -25 > 0;
        feasible=min(feasible+(-((Elite(:,1)-8).^2)-...
            ((Elite(:,2)+3).^2)+7.7 > 0),1);
    elseif MOP==21
        Constraint1=Elite(Elite(:,1) + Elite(:,2) -2 ≥ 0,:);
        Constraint2=Constraint1(6-Constraint1(:,1) - ...
            Constraint1(:,2) ≥ 0,:);
        Constraint3=Constraint2(2-Constraint2(:,2) + ...
            Constraint2(:,1) ≥ 0,:);
        Constraint4=Constraint3(2-Constraint3(:,1) + ...
            (3*Constraint3(:,2)) ≥ 0,:);
        Constraint5=Constraint4(4-(Constraint4(:,3)-3).^2 - ...
            Constraint4(:,4) ≥ 0,:);
        Constraint=Constraint5((Constraint5(:,5)-3).^2 + ...
            Constraint5(:,6) -4 ≥ 0,:);
        Rf=size(Constraint,1);
        feasible1=(Elite(:,1) + Elite(:,2) -2) < 0;
        feasible2=6-Elite(:,1) - Elite(:,2) < 0;
        feasible3=2-Elite(:,2) + Elite(:,1) < 0;
        feasible4=2-Elite(:,1) + (3*Elite(:,2)) < 0;
        feasible5=4-(Elite(:,3)-3).^2 - Elite(:,4) < 0;
        feasible6=(Elite(:,5)-3).^2 + Elite(:,6) -4 < 0;
        feasible=min(feasible1+feasible2+feasible3+feasible4+...
            feasible5+feasible6,1);
    elseif MOP==22
        Constraint1=Elite(Elite(:,2) + 4*(Elite(:,1)) -4 < 0,:);
        Constraint2=Constraint1(-Constraint1(:,1) -1 < 0,:);
        Constraint=Constraint2(Constraint2(:,1) - ...
           Constraint2(:,2) ...
            - 2 < 0,:);
        Rf=size(Constraint,1);
        feasible1=Elite(:,2) + 4*(Elite(:,1)) -4 ≥ 0;
        feasible2=-Elite(:,1) -1 ≥ 0;
        feasible3=Elite(:,1) - 2 - Elite(:,2) ≥ 0;
        feasible=min(feasible1+feasible2+feasible3,1);
    elseif MOP==23
        a=0.1;
        b=16;
        Constraint=Elite(-(Elite(:,1).^2) -(Elite(:,2).^2) + 1 + ...
            (a*cos(b*atan(Elite(:,1)./Elite(:,2)))) ≤ 0,:);
        Rf=size(Constraint,1);
        feasible=min(-(Elite(:,1).^2) -(Elite(:,2).^2) + 1 + ...
            (a*cos(b*atan(Elite(:,1)./Elite(:,2))))>0,1);
    end
end
%Apply constraints (not used, first attempt at constraint method).
function Constraint = SideConstraint(Elite, MOP)
    %after every Elite is created, discard the ones which do not ...
        fall
    %within the constraints line 147
    if MOP==20
        %(x-5)^2+y^2-25≤0
        Constraint1=Elite(((Elite(:,1)-5).^2) + ...
            (Elite(:,2).^2) -25 ≤ 0,:);
        Constraint=Constraint1(-((Constraint1(:,1)-8).^2)-...
            ((Constraint1(:,2)+3).^2)+7.7≤0,:);
    elseif MOP==21
        %(x-5)^2+y^2-25≤0
        Constraint1=Elite(Elite(:,1) + Elite(:,2) -2 ≥ 0,:);
        Constraint2=Constraint1(6-Constraint1(:,1) - ...
            Constraint1(:,2) ≥ 0,:);
        Constraint3=Constraint2(2-Constraint2(:,2) + ...
            Constraint2(:,1) ≥ 0,:);
        Constraint4=Constraint3(2-Constraint3(:,1) + ...
            (3*Constraint3(:,2)) ≥ 0,:);
        Constraint5=Constraint4(4-(Constraint4(:,3)-3).^2 - ...
            Constraint4(:,4) ≥ 0,:);
        Constraint=Constraint5((Constraint5(:,5)-3).^2 + ...
            Constraint5(:,6) -4 ≥ 0,:);
    elseif MOP==22
        Constraint1=Elite(4*(Elite(:,1)) + Elite(:,2) - 4 < 0,:);
        Constraint2=Constraint1(Constraint1(:,1) +1 > 0,:);
        Constraint=Constraint2(Constraint2(:,1) - ...
            Constraint2(:,2) - 2 > 0,:);
    elseif MOP==23
        a=0.1;
        b=16;
        Constraint=Elite(-(Elite(:,1).^2) -(Elite(:,2).^2) + 1 + ...
            (a*cos(b*atan(Elite(:,1)./Elite(:,2)))) ≤ 0,:);
    end
end
%Initialise problems - number of variables, number of objectives,
%variable limits, etc.
function [NumVars, NumObjectives, Limits, L, SheetName, ...
   ProblemN] = ...
    InitializeProblem(MOP)
MOP_Config = [1 1 2 -1E5 1E5, %mop num, number of variables, ...
   number of
    %objectives, lower limit for variable 1, upper limit for ...
        variable 1,
    %lower limit for variable 2, upper limit for variable 2 etc.
             2 3 2 -4 4,
             3 2 2 -pi pi,
             4 3 2 -4 4,
             5 2 3 -30 30,
             6 2 2 0 1,
             7 2 3 -400 400,
             8 30 2 0 1,   % ZDT1
             9 30 2 0 1,    % ZDT2
             10 30 2 0 1,   % ZDT3
             11 3 2 -5 5,  % ZDT4 en x1 is (0,1)!
             12 10 2 0 1,   %ZDT6
             13 2 2 0 1,   %OKA1
             14 5 1 -1 1,  %Rosenbrock (NB: One objective
             15 7 3 0 1,
             16 2 2 0 1, %Truss
             17 2 1 -2 2,
             18 2 1 -8 8,
             19 2 1 0 10, %Shekel
             20 2 2 0 5, %MOP-C1
             21 6 2 0 10, % 0 10 1 5 0 6 1 5 1 10];
             22 2 3 -4 4,
             23 2 2 10^-6 pi
             ];
NumVars        = MOP_Config(MOP, 2);
NumObjectives = MOP_Config(MOP, 3);

for i=1:NumVars    %Set problem boundaries
    L(i,1) = MOP_Config(MOP, 4);
    L(i,2) = MOP_Config(MOP, 5);
end

ProblemName = ['MOP1 ', 'MOP2 ', 'MOP3 ', 'MOP4 ', 'MOP5 ',  ...
    'MOP6 ', 'MOP7 ', 'ZDT1 ', 'ZDT2 ', 'ZDT3 ', 'ZDT4 ',  ...
    'ZDT6 ', 'OKA1 ', 'ROSB ', 'DTLZ1', 'Truss', ...
    'Single Objective', 'MOP-C1', 'MOP-C2', 'MOP-C3','MOP-C4'];

if MOP==11 % x2 to x10 are on [-5,5], x1 is on [0,1]
    L(1,1)=0;
    L(1,2)=1;
end

if MOP==13
    L(1,1) = 6*sin(pi/12); L(1,2) = 6*sin(pi/12) + 2*pi*cos(pi/12);
    L(2,1) = -2*pi*sin(pi/12); L(2,2) = 6*cos(pi/12);
end

if MOP == 14  %Truss problem
    L(1,1) = 0.1;  L(1,2) = 2.25;
    L(2,1) = 0.5;  L(2,2) = 2.5;
end

if MOP == 20  %MOP-C1
    L(2,1) = 0;
    L(2,2) = 5;
end

if MOP == 20   %MOP-C2
    L(2,1) = 0;
    L(2,2) = 10;
    L(3,1) = 1;
    L(3,2) = 5;
    L(4,1) = 0;
    L(4,2) = 6;
    L(5,1) = 1;
    L(5,2) = 5;
    L(6,1) = 0;
    L(6,2) = 10;
end
for i=1:NumVars
    Limits(i,1) = L(i,1);
    Limits(i,2) = L(i,2);
end;

ProblemN = ProblemName((MOP-1)*5+1:5*MOP);
SheetName = ProblemN(1:4);
end %function InitializeProblem
%Plot solutions
function PlotDetailProgress(MOP, NumVars, NumObjectives, ...
    WorkArea, ...
    ProblemN, NoOfLoops, t, k, N)
  h = subplot(2,2,[1 3]);
  hold on
  if MOP≥20
    scatter(WorkArea(1:N, NumVars+1+2+NumObjectives), ...
        WorkArea(1:N, ...
         NumVars+2+2+NumObjectives), 3, '*');
  else
    scatter(WorkArea(1:N, NumVars+1), WorkArea(1:N, NumVars+2), ...
        3, '*');
  end
  xlabel('f1')
  ylabel('f2')
  title(['Search space for ' ProblemN ': k=' int2str(k) ' of ' ...
      int2str(NoOfLoops) ', t=' int2str(t)]);
  drawnow
  grid on
  hold off
end  %PlotDetailProgress
%ENS-SS algorithm used to rank solutions.
function Out = ENSort(Pop, Rank, NumVars, NumObj)

    N = size(Pop,1);                        % Determine size of ...
        population
    Obj1 = NumVars+1;                       % Column number of first ...
        objective
    K = NumVars+NumObj;                     % Column number of last ...
        objective
    Fronts = cell(N,1);
    Pop=sortrows(Pop, [Obj1,Obj1+1]);   % Sort pop in ascending ...
        order of
    %independent axis Obj, ties broken by dependent axis
    Fronts{1} = 1;
    Pop(1,K+1) = 1;
    Pop(1,K+2) = 1;

    for p = 2:N
       k = 0;
       sorted = false;
       while ¬sorted
           k = k + 1;
           r = size(Fronts{k},2);
           if r == 0
              Fronts{k} = p;
              Pop(p,K+1) = k;
              break
           end
           for i = r:-1:1
              if DominationCheck(Pop(p,Obj1:K),Pop(Fronts{k}(i),...
                     Obj1:K))
                  break
              end
              if i == 1
                 sorted = true;
                 Fronts{k} = [Fronts{k}, p];
                 Pop(p, K+1) = k;
              end
           end
       end
    end

    Out = Pop(Pop(:,K+1)≤Rank,:);

    function result = DominationCheck(s,t)       % Algorithm 4
        M = size(s,2);
        for j = 1:M
           if s(j) < t(j)
               result = false;
               break
           end
           result = true;
        end
    end
end
```