Volume 38 (2), pp. 177–201
http://orion.journals.ac.za

ORiON
ISSN 0259–191X (print)
ISSN 2224–0004 (online)

Methods of enhancing the MOO CEM algorithm

V Tränkle* JF Bekker†

Received: 15 September 2021; Revised: 17 November 2022; Accepted: 17 November 2022

Abstract

With the increasing need to solve problems faster and with fewer resources, great emphasis is placed on optimisation. Many real-world problems require addressing more than one objective that are in conflict, as well as taking into consideration a number of practical restrictions or constraints. The *multi-objective optimisation using the cross-entropy method* (MOO CEM) algorithm is one of many algorithms that addresses the need to solve multi-objective problems effectively, but it has a number of limitations. This paper explores methods of enhancing the MOO CEM algorithm in order to improve the efficiency and increase the functionality of the algorithm, allowing for it to be applied to additional classes of problems. Three possible methods of enhancement were identified: using the beta distribution to improve sampling, adding functionality to solve constrained problems and, lastly, implementing a non-dominated sorting algorithm to solve problems with more than two objectives. The new algorithms incorporating these enhancements were developed and tested on benchmark problems. Subsequently, the results were analysed using standard performance indicators and compared to results produced by the original MOO CEM algorithm. The findings of this study indicate that using the beta distribution improves sampling and therefore algorithm efficiency. Methods of handling constraints and solving problems with an increased number of objectives were implemented successfully. Based on these results, a final algorithm implementing the enhancements is presented.

Key words: Cross-entropy method, multi-objective optimisation

1 Introduction

In recent times, increasing emphasis is being placed on optimisation. Due to time restrictions and financial demands, problems must be solved and decisions made as quickly as possible with as few resources as possible. Banks are reliant on real-time predictive models

†Department of Industrial Engineering, University of Stellenbosch, South Africa, email: veetrankle@gmail.com

†Corresponding author: Department of Industrial Engineering, University of Stellenbosch, South Africa, email: jb2@sun.ac.za

http://dx.doi.org/10.5784/38-2-765

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 optimisation problems, a number of goals (or objectives) and restrictions must be taken into account. When a single best solution does not exist, a solution must be selected from a number of good solutions. Given time and monetary restrictions, not all solutions to a problem can be considered. Problems such as these, which require taking into account more than one requirement, are termed multi-objective optimisation problems (MOPs). Conversely, problems which only consider one single objective and aim to find the single best solution, are known as single objective optimisation problems (SOPs).

To solve MOPs, a variety of types of multi-objective optimisation (MOO) algorithms have been developed over the years, many falling within the genetic algorithm (GA) category. Some of the best-known GAs 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) [4].

Some MOO algorithms were developed through the extension of single objective optimisation (SOO) algorithms to MOO problems. One such algorithm is the *multi-objective optimisation using the cross-entropy method* (MOO CEM) algorithm developed in [1]. Through the application of the principles of the cross-entropy method (CEM) to MOPs, [1] created an algorithm with the ability to solve multi-objective deterministic, continuous and dynamic, stochastic MOPs. The MOO CEM algorithm was found to be comparable with the likes of OptQuest® on the problems evaluated and outperformed Matlab®'s MOO GA on the same evaluation set. The algorithm, however, has limitations: it has only been applied to MOO problems with exactly two objectives, and its constraint-handling was absent.

This paper reports on improvements of the MOO CEM algorithm to handle tri-objective problems while making its sampling more efficient by using alternative distributions, and adding constraint-handling capabilities. To begin with, a brief literature study of multi-objective optimisation, the cross-entropy method (CEM) and the MOO CEM algorithm are presented, whereby deficiencies in the algorithm are identified. Based on these, three methods of enhancements are suggested and three algorithms are presented to address each enhancement. Thereafter, the newly developed algorithms were tested on a set of standard benchmark problems and results are discussed. In light of the results, a final algorithm incorporating the successful methods of enhancement is proposed, and the paper is concluded with some final remarks and suggestions for future research.

2 MOO, the MOO CEM algorithm and areas of enhancement

MOO algorithms differ from SOO algorithms in that, while the goal of SOO algorithms is to find the single best solution to a problem, MOO algorithms require taking into account more than one objective of which some are in conflict, and making trade-offs between conflicting objectives.

2.1 MOO problem formulation

A MOO problem (MOP) can be mathematically formulated as [6]:

Minimise or Maximise

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

subject to

$$g_j(\mathbf{x}) \geq 0,\ j = 1, \ldots, q, \tag{2}$$

and

$$h_l(\mathbf{x}) = 0,\ l = 1, \ldots, p, \tag{3}$$

with

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

where f_i represents the K objective functions to be minimised or maximised, $g_j(\mathbf{x}) \geq 0$ are the q inequality constraints and $h_l(\mathbf{x}) = 0$ are the p equality constraints.

For many MOPs, no single optimal solution exists, but rather a set of good solutions belonging to the *Pareto-optimal* solution set. By considering the solutions in the Pareto-optimal solution set, an informed decision can be made when selecting the 'best' solution [13].

Consider purchasing a vehicle as an example of a bi-objective optimisation problem: a vehicle must be selected based on cost and safety rating. The first objective, cost, should be minimised, while the second objective, safety rating, should be maximised. Generally, safer vehicles tend to be more expensive. Therefore, by considering all solutions in the Pareto-optimal solution set, the best vehicle can be selected by considering both objectives simultaneously. The Pareto-optimal solution set is based on the principle of Pareto Dominance. Pareto Dominance is defined as

$$\begin{aligned}
&\mathbf{a} \succ \mathbf{b}\ (\mathbf{a} \text{ dominates } \mathbf{b}) && \textit{iff}\ \mathbf{f(a)} > \mathbf{f(b)},\\
&\mathbf{a} \succeq \mathbf{b}\ (\mathbf{a} \text{ weakly dominates } \mathbf{b}) && \textit{iff}\ \mathbf{f(a)} \geq \mathbf{f(b)},\\
&\mathbf{a} \sim \mathbf{b}\ (\mathbf{a} \text{ is indifferent to } \mathbf{b}) && \textit{iff}\ \mathbf{f(a)} \not\geq \mathbf{f(b)} \wedge \mathbf{f(b)} \not\geq \mathbf{f(a)},
\end{aligned}$$

where $\mathbf{a}$ and $\mathbf{b}$ are two objective vectors [17].

If solution vector $\mathbf{a}$ is found to dominate solution vector $\mathbf{b}$, $\mathbf{a}$ is a non-dominated solution vector. This implies that $\mathbf{a}$ is optimal and cannot be improved in one objective without worsening the solution in at least one other objective. These solutions are referred to as *Pareto-optimal* [17].

2.2 Cross-entropy method

The cross-entropy method (CEM) was developed by Rubinstein and Kroese [12] 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 that 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 [12].

The cross-entropy method is iterative, each iteration consisting of two distinct parts [12]:

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 (4) below
3: Use the sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ and solve the stochastic program in (5) below. This solution is $\mathbf{v}_t$ below
4: Smooth the vector $\mathbf{v}_t$ using the expression in (6)
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 the decision vector $(X_1, \ldots, X_n)$ (initially random) and γ_t is updated adaptively according to

$$\hat{\gamma}_t = f_{([(1-\varrho)N])}, \tag{4}$$

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

$$\max_{\mathbf{v}} \hat{D}(\mathbf{v}) = \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^{N} I_{\{f(\mathbf{X} \geq \hat{\gamma}_t)\}} \ln h(\mathbf{X}_i; \mathbf{v}). \tag{5}$$

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

$$\hat{\mathbf{v}}_t = \omega \tilde{\mathbf{v}}_t + (1-\omega)\hat{\mathbf{v}}_{t-1} \tag{6}$$

where ω is a smoothing constant in the range $0-1$ (typically chosen to be between 0.6 and 0.9).

2.3 The MOO CEM algorithm

Based on the principles of the CEM developed by Rubinstein and Kroese [12], Bekker [1] created the MOO CEM algorithm. Bekker [1] first applied the CEM to SOO benchmark problems, and then extended the algorithm to MOPs to create Algorithm 2 below.

Algorithm 2 MOO CEM Algorithm

1: Set **Elite** $= \varnothing, t \leftarrow 1, k \leftarrow 1$
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 κ
7: **for** each frequency element $\tau_{i\kappa}$ in $\mathbf{R}_i$
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 $\phi_i(\mathbf{x}_i\, \tilde{\mathbf{v}}\text{'})$
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 (6)
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: $k \leftarrow k+1$
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 Algorithm 3 with $\rho_E = 0$ to obtain the final elite vector

The value 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 a steady-state and further iterations would most likely not improve the solution. D represents

the number of decision variables, K is the number of objectives and N is the number of solutions.

The MOO CEM algorithm uses a dominance-based ranking algorithm [7] (see Algorithm 3) as a means of finding the set of best solutions, or Pareto-optimal solution set.

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 ρ_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 ρ_{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 ρ_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 set. $\rho_E = 1$ represents the second Pareto set with the second-best solutions.

Bekker [1] achieved good results when the MOO CEM algorithm was tested on a number of benchmark as well as practical problems. These problems included deterministic and continuous problems, as well as dynamic, stochastic problems. The practical problems included the buffer allocation problem (BAP) and variants thereof, a reconfigurable manufacturing system, an extrusion equipment design problem and CO_2 gas management problem at a mine. The algorithm achieved good results for a limited number of iterations, achieving proximity to the true Pareto front, while maintaining diversity. The algorithm was then compared to two standard algorithms: Matlab® MOO GA and OptQuest®. MOO CEM proved to be competitive when compared to OptQuest® and outperformed Matlab® MOO GA on a number of benchmark problems.

3 Methods of enhancement and algorithm development

Considering the quality of the solutions produced by the MOO CEM algorithm, it seems only natural to further *improve the performance* of the algorithm and to *extend the functionality* of the algorithm. Currently, the algorithm is unable to accommodate problems with side-constraints, and can only solve problems with two objectives (due to the manner in which the ranking method is implemented). It was also hypothesised that the sampling method could be improved by considering a different distribution. Three methods of enhancement were identified:

1. Improve the sampling method of the MOO CEM algorithm by using the beta distribution, rather than truncated normal distributions.
2. Extend the capability of the existing MOO CEM algorithm to include functionality to solve side-constrained problems.
3. Extend the scope of the existing MOO CEM algorithm to include functionality to solve problems with three objectives.

These three methods are subsequently discussed in separate subsections.

3.1 Proposed MOO CEM-Beta algorithm

Many 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, the uniform distribution, the chi-square distribution, the binomial distribution, the Poisson distribution and the beta distribution. The MOO CEM algorithm in its original form uses truncated normal distributions to provide values for the decision variables. The decision variable distributions (truncated normals) converge in central tendency and reduced variance to yield the optimal values for the decision variables. It was inferred that more 'flexible' distributions might increase the performance of the MOO CEM algorithm because the optimum shapes of the decision variable distributions can be better described.

The beta distribution is extremely versatile and can approximate many other continuous distributions using only two parameters: α and β. These two parameters specify whether the distribution is symmetrical and where the distribution's mode falls within the range of the distribution. The beta distribution is continuous and defined over the interval $(0, 1)$ [8]. When α and β are positive integers, calculating the beta distribution is relatively computationally inexpensive. However, when α and β are non-integer, evaluating the gamma function found in the beta distribution becomes considerably more computationally expensive.

Figure 1 shows the shape of the beta distribution for different values of α and β. When α and β are both equal to 1, the distribution simplifies to a uniform distribution (line labelled 'a'). When α and β are equal and $\alpha + \beta$ is large enough, the shape of the distribution is reminiscent of that of a normal distribution (curve labelled 'b'). 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 (curve labled 'c'). When α is less than 1 and β is greater than 1, the distribution resembles an exponential distribution (curve labled 'd'). When α is greater than β, the distribution is skewed to the right (curve labled 'e') and *vice versa*. Let X be a beta distributed random variable. The range of the beta distribution can be extended from (0,1) to a range (a, b), $a < b$, by scaling a new variable $Y = a + X(b - a)$, therefore $X = (Y - a)/(b - c)$. For further detail pertaining to the beta distribution, see [9].

Based on the MOO CEM algorithm (Algorithm 2), the MOO CEM-Beta algorithm was developed, replacing the truncated normal distributions used when sampling decision variable values, with appropriately parameterised beta distributions. Algorithm 4 replaces steps $8 - 11$ in the original MOO CEM algorithm.

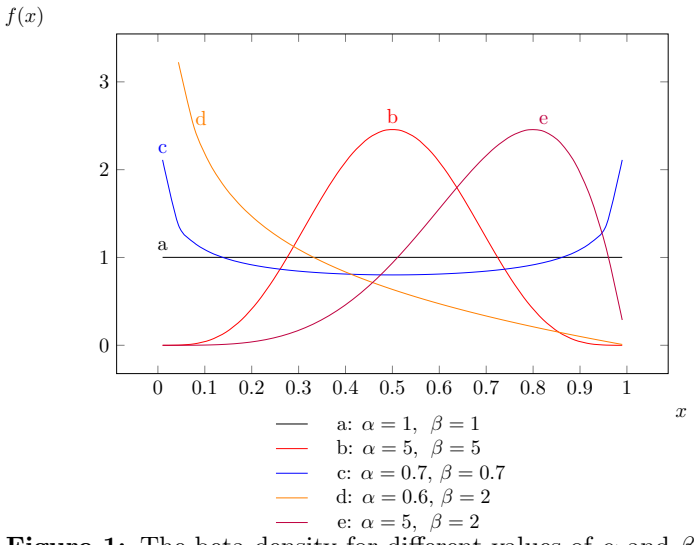

Figure 1: The beta density for different values of α and β

Algorithm 4 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** one or no unique Elite solutions fall within this range
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(α_i,β_i) with $\mathbf{x}_i \, \epsilon \, [c_{i\kappa}\, c_{i(\kappa+1)})$ and $|\mathbf{Y}| = \tau_{i\kappa}, 1 \leq \kappa \leq r + 2$
9: Append $\mathbf{Y}$ to $\mathbf{X}_i$

The parameters required by the beta distribution (α and β) are calculated for each sample set using the elite solutions which fall within the allowed decision variable ranges. Calculating α and β requires at least two points. If only one or no unique elite solutions fall within the ranges, 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 decision variable ranges.

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 distribution parameter vectors are smoothed and a stopping criterion is calculated in steps $20 - 22$. This is replaced with the smoothing of the α and β parameters in a similar fashion, using (6). The stopping criterion 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 significantly from one iteration to the next, the algorithm is assumed to have reached a steady-state and there would be no benefit in running the algorithm for further iterations. These updated steps are contained in Algorithm 5 and intend to replace steps $20 - 22$ of the MOO CEM algorithm.

Algorithm 5 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 (6)
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 $\boldsymbol{\beta}_i$ exceed ϵ_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-Constraint algorithm

At a high level, two types of MOO problems exist: constrained and unconstrained problems. (2) and (3) represent the constraints which apply to an unconstrained problem formulated by (1). Constraints add a dimension of complexity to a MOO problem, as they limit the feasible region of the decision variables [4]. Additional challenges arise from the interference among constraints and the relationships among decision variables, constraints and the objective functions [15].

Many different methods have been developed to solve constrained MOPs, one of the simplest methods being the removal of infeasible solutions post-processing. Once all solutions have been created, 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 solutions, leaving the algorithm with little direction for the next iteration and, ultimately, a poor Pareto solution set [4].

Other popular constraint-handling methods include [3]:

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

This paper focuses specifically on the Penalty Function method, as this is a well-researched and widely accepted constraint-handling method and an appropriate method to be applied to the MOO CEM algorithm.

Woldesenbet *et al.* [15] 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 in [14] to MOO problems. The Constrained Multi-objective Evolutionary Algorithm (CMOEA) developed in [15] calculates the penalty according to the size of the solution's constraint violation, defined as

$$F_i(\mathbf{x}) = d_i(\mathbf{x}) + p_i(\mathbf{x}), \tag{7}$$

where i is an index referring to each objective function.

The penalty has two components: a distance measure $d_i(\mathbf{x})$ and an adaptive penalty $p_i(\mathbf{x})$. Let $v(\mathbf{x})$ be the constraint violation, this is made precise below. The distance measure is calculated using Algorithm 6, based on

$$d_i(\mathbf{x}) = \begin{cases} v(\mathbf{x}), & \text{if } r_f = 0, \\ \sqrt{\tilde{f}_i(\mathbf{x})^2 + v(\mathbf{x})^2)} & \text{otherwise,} \end{cases} \tag{8}$$

where

$$r_f = \frac{\text{number of feasible solutions in the current population}}{\text{population size}}. \tag{9}$$

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 $v(\mathbf{x})$. 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 $v(\mathbf{x})$ is the normalised violation of each constraint and is calculated according to

$$v(\mathbf{x}) = \frac{1}{m} \sum_{j=1}^{m} \frac{c_j(\mathbf{x})}{c_{\max}^j}, \tag{10}$$

where

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

and

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

The equality constraints (with the format (3)) are converted to inequality constraints (as per (2)) by the addition of the tolerance value δ. 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. $c^j_{\max}$ is the maximum possible violation of the j-th (or l-th) constraint, which allows for the normalisation according to (10). $g_j(\mathbf{x})$ and $h_l(\mathbf{x})$ are the equality and inequality constraints as per (2) and (3) in the MOO problem definition.

CMOEA requires each solution for each objective i to be normalised according to (11), where $\tilde{f}_i(\mathbf{x})$ is the normalised objective value of i and $f^i_{\min}$ and $f^i_{\max}$ refer to the minimum and maximum values of objective i, respectively. Normalisation is determined by

$$\tilde{f}_i(\mathbf{x}) = \frac{f_i(\mathbf{x}) - f^i_{\min}}{f^i_{\max} - f^i_{\min}}, \tag{11}$$

where

$$f^i_{\min} = \min_{\mathbf{x}} f_i(\mathbf{x}),$$

and

$$f^i_{\max} = \max_{\mathbf{x}} f_i(\mathbf{x}).$$

The penalty value $p_i(\mathbf{x})$ is calculated according to Algorithm 7, based on

$$p_i(\mathbf{x}) = (1 - r_f)X_i(\mathbf{x}) + r_f Y_i(\mathbf{x}), \tag{12}$$

where

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

and

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

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

Woldesenbet *et al.* [15] 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 set for all test problems.

Algorithm 6 Distance Measure of the constraint-handling multi-objective evolutionary optimisation algorithm

1: **if** $r_f = 0$ **then**
2: **for** $i = 1$ to M **do** ▷ Number of objectives
3: **for** $k = 1$ to N **do** ▷ Population size
4: $d_i(\mathbf{x}_k) \leftarrow v(\mathbf{x}_k)$
5: **end for**
6: **end for**
7: **else**
8: **for** $i = 1$ to M **do**
9: **for** $k = 1$ to N **do**
10: $\tilde{f}_i(\mathbf{x}_k) \leftarrow \frac{f_i(\mathbf{x}_k) - f^i_{\min}}{f^i_{\max} - f^i_{\min}}$
11: $d_i(\mathbf{x}_k) \leftarrow \sqrt{\tilde{f}_i(\mathbf{x}_k)^2 + v(\mathbf{x}_k)^2}$
12: **end for**
13: **end for**
14: **end if**

Algorithm 7 Penalty Value of the constraint-handling multi-objective evolutionary optimisation algorithm

1: **for** $i = 1$ to M **do** ▷ Number of objectives
2: **for** $k = 1$ to N **do** ▷ Population size
3: **if** $r_f = 0$ **then**
4: $X_i(\mathbf{x}_k) \leftarrow 0$
5: **else**
6: $X_i(\mathbf{x}_k) \leftarrow v(\mathbf{x}_k)$
7: **end if**
8: **if** $v(\mathbf{x}_k) = 0$ **then**
9: $Y_i(\mathbf{x}_k) \leftarrow 0$
10: **else**
11: $Y_i(\mathbf{x}_k) \leftarrow \tilde{f}_i(\mathbf{x}_k)$
12: **end if**
13: $p_i(\mathbf{x}_k) \leftarrow (1 - r_f)X_i(\mathbf{x}_k + r_f Y_i(\mathbf{x}_k))$
14: **end for**
15: **end for**

Both the method of discarding solutions and the Penalty Method were applied to the MOO CEM algorithm as a method of solving constrained problems. It was found that using the simple discarding method caused too many solutions to be ignored, resulting in the algorithm not converging to a good solution set. Therefore, only the Penalty Method is presented next.

Algorithm 8 describes the penalty method in [15] and should update the objective function values to include the dynamic penalty. This was achieved by replacing steps 17 and 18 of the MOO CEM algorithm with steps 1–6 of the proposed algorithm and replacing ranking steps as indicated in Algorithm 8.

Algorithm 8 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 (8)
3: Calculate the penalty p_i of each solution according to (12)
4: Calculate the final modified objective value of each solution using (7)
5: **end for**
6: Rank the final modified objective values using the Pareto ranking of Algorithm 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 execution of this penalty method is more computationally expensive and time-consuming than the first constraint method discussed (the method of discarding all infeasible solutions), as the maximum of each constraint violation and minimum and maximum values of each objective must be calculated. However, the benefit of this method is that "good" solutions which do not adhere to the constraints, but lie close to the feasible region, are added to the elite set of each iteration. These infeasible solutions help "guide" the solution set through the iterations, but are excluded from the final elite set.

3.3 Proposed MOO CEM-ENS algorithm

The MOO CEM algorithm was improved further by adding a more efficient sorting and selection algorithm. In the field of MOO, sorting and selection algorithms are important for finding the best members in a population and many such algorithms have been developed. Inefficiency of a sorting and selection algorithm will be detrimental to the overall performance of a MOO algorithm. One of the best-known sorting and selection algorithms is the Pareto-ranking algorithm [7] which was originally implemented in the MOO CEM algorithm. The method was implemented in such a manner that only problems with two objective functions can be solved efficiently, according to Algorithm 3. This method has a complexity of $O(KN^2)$.

Since the algorithm in [7] was published in 1989, a number of faster sorting algorithms

have been developed, including Efficient Non-Dominated Sort (ENS) [16]. ENS performs better at lower dimensions (objectives fewer than five) than other non-dominated sorting algorithms. Since one of the objectives of this study is to add the functionality to efficiently solve problems with three objectives to the MOO CEM algorithm, ENS was selected as a suitable non-dominated sorting algorithm for this case and will be discussed next; the pseudo-code is presented in Algorithm 9.

Algorithm 9 Efficient Non-dominated Sort algorithm

1: **Input**: population P
2: **Output**: the Pareto subsets F
3: F = empty;
4: Sort P in an ascending order of the first objective value;
5: **for all** $P[n] \in$ sorted P
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 in [16] 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 three 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 if 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).

No changes or updates were required in the implementation of this algorithm. The ENS was implemented according to Algorithms 9 and 10. This algorithm replaces Algorithm 3 used for non-dominated sorting in the MOO CEM algorithm and should be used for Pareto ranking in steps 17, 22 and 25 of the MOO CEM algorithm (Algorithm 2).

4 Algorithm testing and results

In order to effectively assess the difference in performance of the adapted algorithms, a number of tests were conducted on benchmark problems. In the case of the beta distribution enhancement (the MOO CEM-Beta algorithm), the algorithm was tested on a number of the same benchmark problems on which the original MOO CEM algorithm was tested. This allowed for the algorithm performance to be compared on the same basis. For the addition of the constraint-handling and increased number of objectives techniques, the MOO CEM-Constraint and MOO CEM-ENS algorithms were tested on some standard benchmark problems on which the original algorithm could not be tested, as the original MOO CEM did not have the functionality to solve these types of problems. Instead, stan-

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

Input: solution $P[n]$, the set of fronts F.
Output: the front number of solution $P[n]$.
$x = \text{size}(F)$; the number of fronts having been found
$k = 1$; the front now checked
while true **do**
    Compare $P[n]$ with the solutions in $F[k]$ starting from the last one and ending with the first one;
    if $F[k]$ contains no solution dominating $P[n]$ **then**
        return k; move $P[n]$ to $F[k]$
        break;
    else
        $k \leftarrow k + 1$
        if $k > x$ **then**
            return $x + 1$; {move $P[n]$ to a new front}
            break;
        end if
    end if
end while

dard indicators such as hyperarea and the epsilon indicator were used as indication of the performance of the algorithms.

4.1 Benchmark problems and performance indicators

During development, the MOO CEM algorithm was tested on (amongst others) the standard MOP test suite: MOP1, MOP2, MOP3, MOP4 and MOP6 as developed by Coello [3]. As such, the MOO CEM-Beta algorithm was tested on the same problem set and three performance indicators were used to compare the performance thereof to the original MOO CEM algorithm: hyperarea, runtime and the Pareto set size.

Hyperarea is a Pareto compliant indicator, which measures the difference in area between a set of Pareto optimal solutions found by an algorithm compared to the hyperarea of the true Pareto-optimal solution set. It relies on a reference point outside of the maximum of the objective function solution space. The term *Pareto compliant* is formally defined as: An indicator *I:* $\Omega \rightarrow \mathbb{R}$ *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))* [5]. This type of indicator was selected, as Pareto-compliant performance indicators are more reliable than Pareto non-compliant indicators with respect to algorithm solution set comparison.

In order to draw a fair comparison between the two result sets, a one-tailed hypothesis test with a significance level of $\alpha = 0.05$ was used. This test is formulated such that the alternative hypothesis indicates that one data set has a larger mean than the other. The objective is to determine whether or not the enhanced algorithm performs better than the

original MOO CEM algorithm. This is done using the following metrics: the hyperarea (two-dimensional cases) and epsilon indicator, runtime and the size of the Pareto set. In the case of hyperarea, the aim is to maximise it. For the runtime metric, the algorithm with the shortest runtime is superior. The algorithm which produces the largest set of Pareto solutions is deemed the better algorithm. These three indicators were not 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.

As the performance of the MOO CEM-Constraint and MOO CEM-ENS algorithms could not be compared to original the MOO CEM algorithm, two Pareto compliant indicators were selected to assess the performance of the MOO CEM-Constraint and MOO CEM-ENS algorithms, through comparison of the generated solution set to the true solution set: hyperarea and the epsilon indicator (ϵ). In cases where a solution had more than two dimensions, indicating that hypervolume should be used, rather than hyperarea, and the hypervolume could not be calculated due to complexity of the solution set shape, only the epsilon indicator was used to compare the generated solution set to the true solution set. Runtimes and Pareto set sizes were recorded for reference. The performance of MOO CEM-Constraint was tested on side-constrained MOPs: MOP-C1, MOP-C2 and MOP-C4 and the MOO CEM-ENS algorithm on problems with three objectives: MOP5, MOP7 and MOP-C3 (which includes side-constraints).

500 simulations were completed for the testing of each algorithm on each test problem and a population size of 200 was specified. The MOO CEM-Beta algorithm was allowed to run for a maximum number of 15 000 evaluations, while for the MOO CEM-Constraint and MOO CEM-ENS algorithms the maximum number of evaluations was increased to 30 000. This was due to increased complexity of the test problems, requiring additional evaluations for the algorithms to converge to an acceptable solution.

4.2 MOO CEM-Beta algorithm results

The average hyperarea, execution time and Pareto set size results of each test are listed in Table 1 for reference. The results of the one-tailed t-test are shown in Table 2, based on 500 simulations.

Table 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

The t-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 average execution times for five out of the eight 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 2–4 display the MOO CEM and MOO CEM-Beta approximate Pareto sets and the true Pareto sets. Final 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.

4.3 MOO CEM-Constraint algorithm results

The tests conducted on the MOO CEM-Beta algorithm showed promising results. Therefore, the MOO CEM-Beta algorithm (rather than the original MOO CEM algorithm) was enhanced using both constraint methods (the method of discarding infeasible solutions and the dynamic penalty method). This enhancement gives the algorithm the ability to solve problems with side-constraints. Four standard side-constraint problems (MOP-C1, MOP-C2 , MOP-C3 and MOP-C4) were selected to test the performance of the algorithm. The results of MOP-C3 are discussed in the subsequent section, as this problem has three

Table 2: One-tailed t-test results of MOO CEM compared MOO CEM-Beta results on some standard benchmark problems

Test Problem	Hyperarea		Runtime		Pareto Set Size	
	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

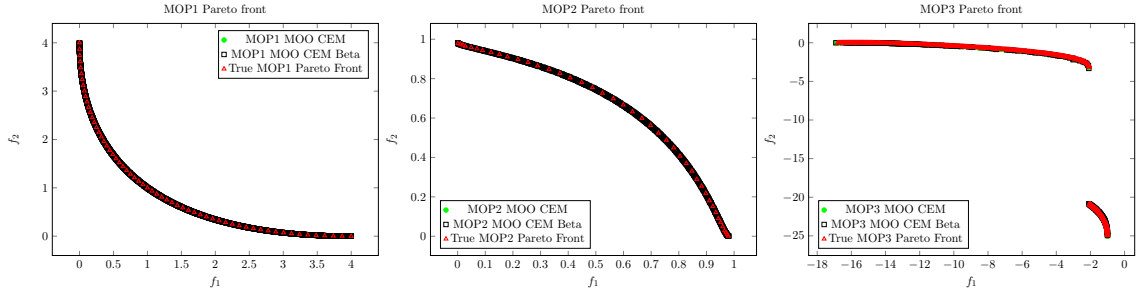

Figure 2: MOO CEM and MOO CEM Beta Pareto sets compared to the true Pareto set for test problems MOP1, MOP2 and MOP3

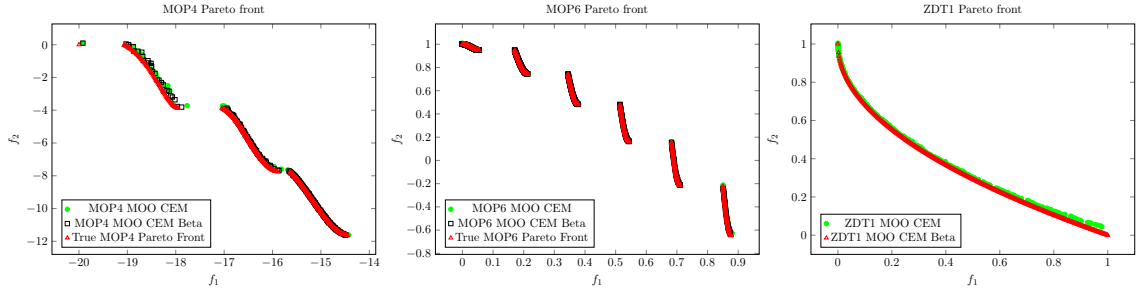

Figure 3: MOO CEM and MOO CEM Beta Pareto sets compared to the true Pareto set for test problems MOP4, MOP6 and ZDT1

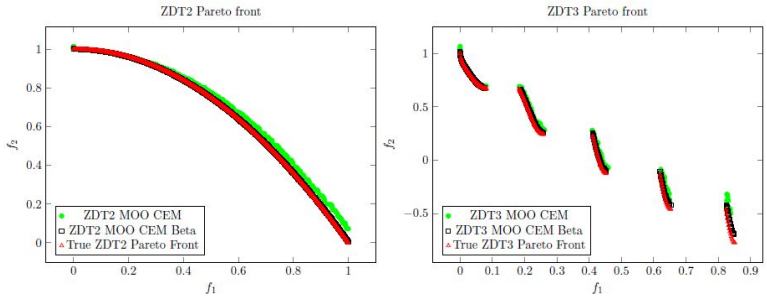

Figure 4: MOO CEM and MOO CEM Beta Pareto sets compared to the true Pareto set for test problems ZDT2 and ZDT3

objective functions, which cannot be solved by the MOO CEM-Constraint algorithm, as MOO CEM-Constraint was based on MOO CEM which was not implemented with the functionality to solve problems with more than two objectives. MOO CEM-ENS which gives MOO CEM the ability to solve problems with three objectives, was developed subsequent to MOO CEM-Constraint, giving it the functionality to solve MOP-C3 having three objectives and constraints.

The method of discarding infeasible solutions was applied to the algorithm first, 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 were required. The result was a long runtime and an insufficient Pareto solution set.

Thereafter, the dynamic penalty method was applied to the MOO CEM-Beta algorithm according to Algorithm 8 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 3. Figure 5 shows the solution sets generated by the enhanced algorithm compared to the true Pareto solution set for problems MOP-C1, MOP-C2 and MOP-C4.

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 the generated solution set and true Pareto solution set is seen (15%). Considering Figure 5, it is observed that the generated solutions closely approximate the true Pareto set for larger values of f_1, but deviate from the true set 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

Table 3: 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

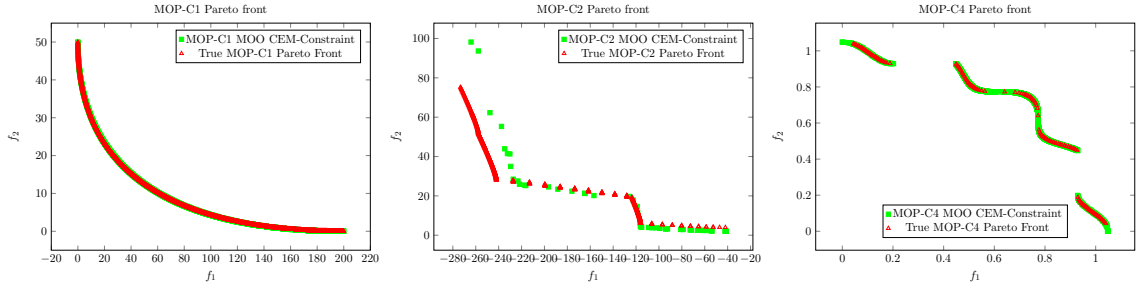

Figure 5: MOO CEM-Constraint Pareto sets compared to the true Pareto set for test problems MOP-C1, MOP-C2 and MOP-C4

for which the algorithm is run. Since the runtime of this problem is currently short, this may be a feasible approach.

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-C4, the generated solution set could possibly be improved as suggested above for MOP-C2.

4.4 MOO CEM-ENS algorithm results

The MOO CEM-Constraint algorithm was enhanced by including the ENS-SS ranking algorithm. 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. Figure 6 shows the solution sets generated by the enhanced algorithm as well as the true Pareto solution set.

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 the MOPs on which MOO CEM-Beta was tested, 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 6, 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 6, 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.

Table 4: MOO CEM-ENS results on some standard benchmark problems with three 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

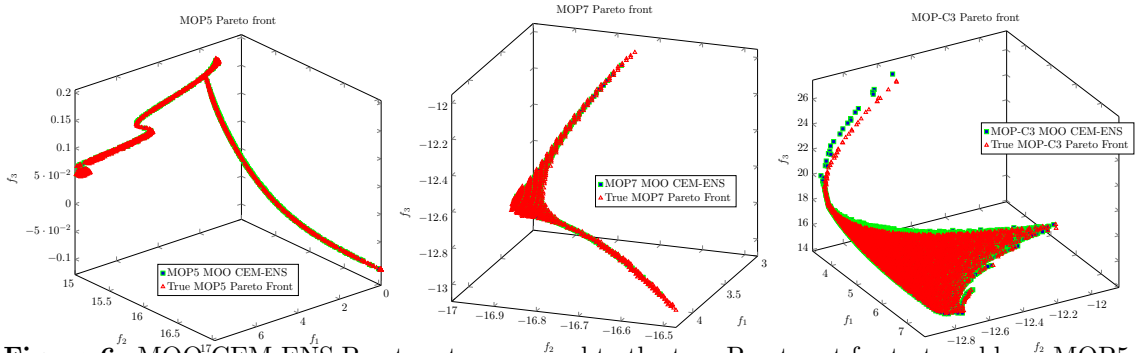

Figure 6: MOO CEM-ENS Pareto sets compared to the true Pareto set for test problems MOP5, MOP7 and MOP-C3

5 Proposed algorithm

Taking into account the results presented in the previous sections, 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).

This final algorithm pseudo code is presented in Algorithm 11. The algorithm follows the same logic as the original MOO CEM algorithm, with the addition of the proposed enhancements.

6 Research Summary

This research explored methods of enhancing and improving the MOO CEM algorithm. A method specific to improving the sampling method of enhancement was identified, namely using the beta distribution instead of truncated normal distributions. Two methods of increasing the functionality of the algorithm were also identified: adding functionality to solve problems with constraints, and implementing a non-dominated sorting algorithm with the ability to solve problems with three objectives. These three areas were researched and appropriate methods of application to the MOO CEM algorithm were identified.

Three algorithms, each addressing one method of enhancement, were developed and tested on a number of standard benchmark problems. The results indicated that replacing the truncated normal distribution with the beta distribution, could improve sampling. When the algorithms with additional functionality were tested on new problems on which the original MOO CEM algorithm could not be tested, positive results were achieved.

Based on the achieved results, a final enhanced MOO CEM algorithm was presented. This algorithm uses the beta distribution when sampling, has the ability to solve constrained problems, and also possesses the functionality to solve problems with three objectives.

Algorithm 11 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 $\mathbf{C}_i = \{c_{i1}, \ldots, , c_{i(r+2)+1}\}$ and histogram frequency vector $\mathbf{R}_i = \{\tau_{i1}, \ldots, \tau_{i(r+2)}\}$.
4: Set $i = 1$.
5: Set $\kappa = 0$.
6: Increment κ.
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 $Beta(\alpha_i,\beta_i)$ distribution.
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 in **q do**
25: Calculate the distance measure d_i of each solution according to (8).
26: Calculate the penalty p_i of each solution according to (12).
27: Calculate the final modified objective value of each solution using (7).
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 (6).
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 $\boldsymbol{\beta}_i$ exceed ϵ_c, or less than the allowable number of evaluations have been done, increment t and reiterate from Step 4.
35: For a constrained problem, rank **Elite** on the final modified objective values, otherwise rank 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.

7 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 large numbers of decision variables. The results of the MOO CEM-Beta tests indicated that the algorithm is especially effective when considering problems with a large 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. Other sorting algorithms should be investigated to decrease the runtime of problems with a large number of objectives.
4. The performance of the algorithm should be compared to other industry leading optimisation algorithms.

References

[1] Bekker JF, 2012, *Applying the cross-entropy method in multi-objective optimisation of dynamic stochastic systems*, Ph.D. thesis, University of Stellenbosch, Stellenbosch.

[2] Burkardt J, 2014, *The Truncated Normal Distribution*, [Online], [Cited August 23rd, 2020], Available from `http://people.sc.fsu.edu/jburkardt/presentations/truncatednormal.pdf`.

[3] Coello Coello CA, 2002, *Theoretical and Numerical Constrain-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art*, Computer Methods in Applied Mechanics and Engineering.

[4] Coello Coello CA, 2006, *Evolutionary Multi-Objective optimization: A Historical View of the Field*, IEEE Computational Intelligence Magazine.

[5] Coello Coello CA & Lamont GB & van Veldhuizen DA, 2007, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Springer.

[6] 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.

[7] Goldberg DE, 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company Inc.

[8] Johnson P & Beverlin M, 2013, *Beta Distributionn*, [Online], [Cited September 1st, 2020], Available from `https://pj.freefaculty.org/guides/stat/Distributions/DistributionWriteups/Beta/Beta.pdf`.

[9] Johnson NL & Kotz S & Balakrishnan N, 1995, *Chapter 25: Beta Distributions in Continuous Univariate Distributions Vol. 2 (2nd ed.)*, Wiley.

[10] Quiza Sardiñas R & Rivas Santana M & Roth S & Alfonso Brindis E, 2006, *Genetic algorithm-based multi-objective optimization of cutting parameters in turning processes*, Engineering Applications of Artificial Intelligence.

[11] Rice JA, 2007, *Mathematical Statistics and Data Analysis*, Thomson Brooks/Cole.

[12] Rubinstein RY & Kroese DP, 2004, *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*, Springer.

[13] Savic D, 2012, *Single-objective vs Multiobjective Optimisation for Integrated Decision Support*, Biennial Meeting of the International Environmental Modelling and Software Society.

[14] Tessema B & Yen GG, 2012, *A self-adaptive constrained evolutionary algorithm*, IEEE International Conference on Evolutionary Computation.

[15] Woldesenbet YG & Tessema BG & Yen GG, 2007, *Constraint handling in multi-objective evolutionary optimization*, 2007 IEEE Congress on Evolutionary Computation, CEC 2007.

[16] Zhang X & Tian Y & Cheng R & Jin Y, 2015, *An Efficient Approach to Nondominated Sorting for Evolutionary Multi-objective*, IEEE Transactions on Evolutionary Computation.

[17] Zitzler E, 1999, *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, TIK-Schriftenreihe.