|  |
| --- |
| **COVENTRY**  UNIVERSITY |
| Faculty of Engineering, Environment and Computing |
| School of Computing, Electronics and Mathematics |
| Data Science and Computational Intelligence |
| M08CDE Individual Project |
| Differential Evolutionary Algorithm for Optimisation and Neuroevolution |
| Author: Alexander Moses |
| SID: 6277932 |
| Supervisor: Dr Mauro Innocente |
| Submitted in partial fulfilment of the requirements for the Degree of Master of Data Science and Computational Intelligence |
| Academic Year: 2018/19 |

**M08CDE Declaration of Originality**

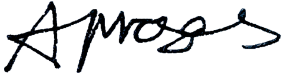
I declare that this project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet etc.) has been acknowledged by citation within the main report to an item in the References or Bibliography lists. I also agree that an electronic copy of this project may be stored and used for the purposes of plagiarism prevention and detection.

**Statement of copyright**

I acknowledge that the copyright of this project report, and any product developed as part of the project, belong to Coventry University. Support, including funding, is available to commercialise products and services developed by staff and students. Any revenue that is generated is split with the inventor/s of the product or service. For further information please see [www.coventry.ac.uk/ipr](http://www.coventry.ac.uk/ipr) or contact [ipr@coventry.ac.uk](mailto:ipr@coventry.ac.uk).

**Statement of ethical engagement**

I declare that a proposal for this project has been submitted to the Coventry University ethics monitoring website (https://ethics.coventry.ac.uk/) and that the application number is listed below (Note: Projects without an ethical application number will be rejected for marking)

Signed:  Date: 19/08/2019

Please complete all fields.

|  |  |
| --- | --- |
| First Name: | Alexander |
| Last Name: | Moses |
| Student ID number | 6277932 |
| Ethics Application Number | P90585 |
| 1st Supervisor Name | Mauro Innocente |
| 2nd Supervisor Name | Kamal Bentahar |

**This form must be completed, scanned and included with your project submission to Turnitin. Failure to append these declarations may result in your project being rejected for marking.**

Abstract

In the field of optimization, there is a family of algorithms known as evolutionary algorithms (EA) which use the concept of biological evolution for global optimization. Differential Evolution (DE) is a type of EA that has proved effective at optimizing multidimensional real-valued functions. This project covers both the implementation of such an algorithm as well as the analysis of it when used for Neuroevolution, the optimization of neural networks. While the algorithm is a couple of decades old, many researchers have contributed to it over the years with many different variations each with their own strengths. This project aims to cover several changes and analyse them to determine their effectiveness when used to optimise neural networks. A number of different DE configurations were created and tested, which proved capable of outperforming the traditional neural network training algorithm, backpropagation while being able to be performed in a similar time frame. In conjunction with this, the paper also contains recommendations for the various aspects and settings of the DE so others could recreate their own.

Table of Contents

[Abstract 2](#_Toc17124531)

[Table of Contents 3](#_Toc17124532)

[Acknowledgements 4](#_Toc17124533)

[1 Introduction 5](#_Toc17124534)

[1.1 Background to the Project 5](#_Toc17124535)

[1.2 Project Objectives 5](#_Toc17124536)

[1.3 Methodology 6](#_Toc17124537)

[1.4 Overview of This Report 6](#_Toc17124538)

[2 Literature Review 7](#_Toc17124539)

[2.1 Basic Differential Evolution Algorithm 7](#_Toc17124540)

[2.1.1 Population Initialization 7](#_Toc17124541)

[2.1.2 Mutation Operation 7](#_Toc17124542)

[2.1.3 Crossover Operation 8](#_Toc17124543)

[2.1.4 Selection Operation 8](#_Toc17124544)

[2.2 Differential Evolution Improvements 9](#_Toc17124545)

[2.2.1 Additional Mutation Strategies 9](#_Toc17124546)

[2.2.2 Strategy Selection 9](#_Toc17124547)

[2.2.3 Control Parameter Adaptation 10](#_Toc17124548)

[2.2.4 Boundary Constraint Handling Techniques 10](#_Toc17124549)

[2.2.5 Population Size Adaptation 11](#_Toc17124550)

[2.2.6 Population Initialization Techniques 14](#_Toc17124551)

[2.2.7 Restart Mechanism 14](#_Toc17124552)

[3 Testing 16](#_Toc17124553)

[4 Implementation 18](#_Toc17124554)

[4.1 Creation of the Basic Differential Evolution 18](#_Toc17124555)

[4.2 Neuroevolution 18](#_Toc17124556)

[4.3 Investigation into Control Parameters 19](#_Toc17124557)

[4.3.1 Scalar F 19](#_Toc17124558)

[4.3.2 Crossover Rate CR 20](#_Toc17124559)

[4.3.3 Control Parameter Recommendations 21](#_Toc17124560)

[4.3.4 Adaptive Control Parameters 22](#_Toc17124561)

[4.3.5 Self-Optimising Control Parameters 23](#_Toc17124562)

[4.4 Mutation Strategies 24](#_Toc17124563)

[4.4.1 Mutation Strategy Recommendation 25](#_Toc17124564)

[4.4.2 Adaptive Mutation Strategy Selection 25](#_Toc17124565)

[4.5 Population Size 26](#_Toc17124566)

[4.5.1 Population Size Recommendation 27](#_Toc17124567)

[4.5.2 Adaptive Population Size 27](#_Toc17124568)

[4.6 Population Initialization 28](#_Toc17124569)

[4.7 Boundary Constraint Handling 29](#_Toc17124570)

[4.8 Restart Mechanism 31](#_Toc17124571)

[4.9 Final Results 31](#_Toc17124572)

[5 Project Management 36](#_Toc17124573)

[5.1 Project Schedule 36](#_Toc17124574)

[5.2 Risk Management 37](#_Toc17124575)

[5.3 Quality Management 38](#_Toc17124576)

[5.4 Social, Legal, Ethical and Professional Considerations 38](#_Toc17124577)

[6 Critical Appraisal 39](#_Toc17124578)

[7 Conclusions 40](#_Toc17124579)

[7.1 Achievements 40](#_Toc17124580)

[7.2 Future Work 41](#_Toc17124581)

[8 Student Reflections 42](#_Toc17124582)

[References 43](#_Toc17124583)

[Appendix A – Control Parameter Analysis Results 1](#_Toc17124584)

[Appendix B – Project Brief and Meeting Records 1](#_Toc17124587)

Acknowledgements

The area of Differential Evolution was new to me when starting this project and so I would like to acknowledge the help and guidance provided to me by my supervisor while completing this project. He not only raised insightful questions about my work but also gave useful pointers on what to do going forward.

# Introduction

## Background to the Project

Both the fields of computer science and engineering contain many real-world problems of which solving them would be a complex task and so the area of Mathematical Optimization is used to attempt to solve these problems. Mathematical Optimization is a method of maximising or minimising a real function to create the best solution to the problem at hand. While the area of mathematical optimization is wide and vast, there is a collection of algorithms known as an evolutionary computation which use concepts taken from biological evolution for global optimization.

One type of evolutionary computing algorithm is the Differential Evolution (DE) algorithm, an algorithm effective for its ability to escape poor local optima and optimization of nondifferentiable problems through the use of difference vectors between individuals in the population.

The DE algorithm can not only optimize real-world mathematical problems, but it can also be used to train neural networks which entail optimizing the weights biases and in some cases the topology. While the DE algorithm has been around for years, several different variations have been proposed over the years, each with its advantages.

Whilst a lot of research has been conducted on using DE on neural networks, these researches lack investigation into the effects the different aspects and variations of DE have on neural networks, and so this paper researches the effects of the control parameters, mutation strategies, and adaptive function on neural network training.

## Project Objectives

The objective is the implementation of the differential evolution algorithm with adaptations from the original algorithm, one capable of performing on several CEC unconstrained optimization benchmarks (such as CEC 2005) to test the capability of the algorithm. Once developed and tested, the algorithm is to be adapted to optimize aspects of an artificial neural network, such as the weights and or topology.

* Creation of the basic Differential evolution algorithm in MATLAB, with the testing of the algorithm on CEC unconstrained optimization benchmarks to get a baseline reading for the algorithm.
* The adaptation of the Differential evolution algorithm with improvements proposed by researchers over the years such as custom fitness functions, crossover functions, and mutation functions. With it being tested against the same CEC benchmarks to evaluate the improvements.
* Adapting the Differential Evolution algorithm to optimize the weights and biases of a neural network
* Adapting the Differential Evolution Algorithm to optimize the number of neurons in a single layer
* Adapting the Differential Evolution Algorithm to optimize the number of layers in the network
* Evaluate the DE algorithm with the different improvements on the ANN to determine the best parameters and operators for the algorithm.

A technical report of both the DE algorithm and the ANN, with evidence of accurate predictions against the original thermal model, and with a basic user manual.

## Methodology

This project attempts to test and analyse the effectiveness of the Differential Evolution algorithm when used for Neuroevolution, the training of neural networks. To do this the different aspects (such as the control parameters) of the network had to be tested separately to determine their impact on the algorithm. In addition to that, many new features which mainly consisted of adaptive functions were added and evaluated. With that in mind, the creation of the basic DE algorithm was the start with the new features being added and tested following an incremental approach. This was deemed best as each feature was to be evaluated independently from each other and so didn’t require the other features to be added yet for one to be analysed.

## Overview of This Report

* The literature review goes through the literature used to make this project which included the work from several different researchers.
* The testing section lays out the structure of all the tests conducted in the project, as testing was conducted alongside the project rather than just at the end, it was described before the implementation.
* The implementation section goes through the work conducted as well as the analysis of the different features and the settings for them.

# Literature Review

## Basic Differential Evolution Algorithm

In 1995 Storn and Price proposed the Differential Evolution algorithm, an algorithm capable of finding the global optimum of non-linear, non-convex, multi-modal and non-differential functions in continuous parameter space (Das, Recent Advances in Differential Evolution - An Updated Survey, 2016). The algorithm starts by creating a population of uniformly random candidates with the search space, which each population being a candidate solution to the problem like other evolution algorithms. Whilst each generation performs the same steps used by the standard evolution algorithm, the creation of the offspring candidates differs from the others by mutating base vectors with scaled differences of other members of the population. The standard DE algorithm follows four steps: Initialization, mutation, crossover, and selection. The three last steps are repeated each generation, with the algorithm continuing until it reaches a set termination criterion such as maximum iterations. Unlike other EAs, with DE being a simple algorithm it only has three control parameters: a difference vector scalar, a crossover rate, and a population size.

### Population Initialization

The algorithm searches for a global optimum point in a d-dimensional search space . The first step is population Initialization, it begins by randomly initialising a population of Np d-dimensional vectors (genomes), with each vector being a candidate solution to the problem at hand. We may use the following notation to represent the *ith*candidate of the population at the current generation *(t)*:

*xi(t) = (xi,1(t), xi,2(t), …, xi,d(t))*

For each variable of the problem, there is likely to be a restrictive boundary, so the variable placed by the problem. With this, the initial population is initialised between the bounds and distributed to cover the range of the bounds as much as possible. The bounds are described as *xmin = (xmin,1, xmin,2, …, xmin,d)* and *xmax = (xmax,1, xmax,2, ..., xmax,d) for* the minimum and maximum bounds respectively. So, each candidate is initialized with the following equation:

*Xi,j(0) = xmin,j + randi,j[0,1](xmax,j – xmin,j),*

Where *jth* is the component for the *ith* candidate and *randi,j*[0,1] is a uniformly distributed random number between 0 and 1.

### Mutation Operation

After the initialization stage, follows the mutation stage, which creates a donor vector for each population member also known as the target vector for the current iteration. Whilst there are several different mutation strategies, there are five that were proposed with the original algorithm:

DE/rand/1: *= + F( - )*

DE/best/1: *= + F()*

DE/current-to-best/1: *= F() + F()*

DE/best/2: *= + F() + F()*

DE/rand/2: *= + F( - ) + F()*

Where *R1i*, *R2i*, *R3i*, *R4i* and *R5i* are mutually exclusive integers randomly chosen within the population and are all different from the target index *i*. These numbers are randomly generated differently for each donor vector. The scaling factor *F* is a positive control parameter for scaling the different vectors, usually between 0.1 and 0.9. is the candidate vector with the best fitness in the population at the current iteration *t*. The naming convention used for mutation strategies is DE*/x/y/z*, where *x* represents the base vector (such as a random one, best one or the combination of the current target and best), *y* represents the number of difference vectors used, and *z* represents the type of crossover strategy used (*exp* being exponential and *bin* being binomial).

### Crossover Operation

During the Crossover stage, an offspring vector is created, also known as a trial vector , by combining the target vector and the donor vector . There are two commonly used crossover strategies used if DE algorithms: exponential (two-point modulo) and binomial (one-point crossover). In Binomial crossover, a random number between 0 and 1 is generated for each of the d parameters, if the random number is less than or equal to the crossover rate control parameter *Cr* then the parameter is taken from the donor vector otherwise it is taken from the trial vector. The strategy is expressed as:

Where is a randomly chosen parameter in the genome used to ensure that the trial vector has at least one component from the donor vector , and is a uniform random number between 0 and 1 generated for each parameter in the genome.

In the exponential crossover, a random integer *n* is generated between 1 and the number of dimensions *d*. This integer is the first of the two pointers used in the crossover in which it is the starting point along the target vector where the crossover operation is performed. Then another integer *L* is created between 1 and the remaining number of parameters after the first pointer, this number determines the number of components is taken from the donor vector. *L* is determined using the following pseudo-code:

*L* = 0;

Do {

*L* = *L* + 1;

} WHILE *rand*[0,1] < *Cr* AND *L* < *d*

With *Cr* also being the *crossover rate* control parameter. After the creation of *n* and *L*, the trial vector is obtained as followed:

Where the angular brackets <.>*d* denote a modulo function with modular *d* (e.g. <x>*d*  = *x* mod *d*). For each donor vector, a new *n* and *L* are generated. Like the binomial crossover, at least one component is taken from the donor vector. Exponential crossover is only effective when neighbouring components influence each other due to its two-point selection, however due to this binomial crossover is the more frequently used crossover strategy.

### Selection Operation

The selection stage determines whether the target (parent) or the trial (offspring) vector survives to the next iteration *t* = *t* + 1, the selection operation is performed as follows:

Where *f*(.) is the objective function to be minimized. With that, if the cost of the new trial vector is equal or lower than its corresponding target vector then the trial vector replaces it. The equality in the helps the population to navigate any flat portions of a fitness landscape and to reduce the possibility of stagnation. Due to the crossover operation between only the target and the donor vectors, only the target and trial vectors tend to have the same values for components. However, if multiple population candidates have the same values for components, then the difference vectors components will be zero. The parent-offspring competition during the selection operation help reduces the possibility of two distinct candidates inheriting the same values for components in the early stages of evolution.

The selection operation can be performed in two different ways, synchronous and asynchronous. In synchronous population update, the method used in the original DE, performs the selection process once all trial vectors are created, this method holds the trial vectors separate from the population until the selection process, so they don’t influence the mutation process. In asynchronous population update, the selection operation is performed as soon as a trial vector is created, allowing the trial vector (if successful) to be added to the population immediately and can influence the mutation processes which in turn can allow faster convergence.

## Differential Evolution Improvements

### Additional Mutation Strategies

Compared to random based mutation strategies (like DE/*rand*/*k*), greedy ones that use the best candidate (such as DE/*best*/*k* and DE/*current-to-best*/*k*) usually have a faster convergence rate. However, due to their usage of the best candidate, there tend to be problems such as premature convergence due to a decrease in population diversity. In the JADE algorithm proposed by Zhang and Sanderson a new mutation strategy called *current-to-p-best* was proposed to take advantage of the fast convergence speed of the greedy strategies but with less premature convergence problems (Sanderson, 2007). The mutation strategy is as follows:

*+* *F*( - ) + *F*( *-* ),

Where is uniformly chosen as one of the top 100*p*% individuals in the current population with . With any of the top 100*p*% solutions being legible to be randomly chosen, it reduces the premature convergence problems caused by the default greedy solutions.

### Strategy Selection

As different mutation functions offer different benefits, such as greedy strategies having faster convergence speed whilst random based strategies help avoid premature convergence, the use of multiple mutation strategies during the evolution can combine all the benefits for an effective algorithm. There are multiple ways to implement this such as dividing the population into sub-populations each with their own mutation strategy. However, an effective method is randomly assigning each member of the population with a mutation strategy from a pool and if the trial vector created from the mutation and crossover stages are unsuccessful then a new mutation strategy is randomly selected from the pool. Using this method, the candidate vector would be as follows:

SaDE, an algorithm proposed by Qin and Suganthan, further developed this idea. Instead of each mutation strategy having an equal chance of being randomly selected, the probabilities are changed based on previous successes (Suganthan, 2005). At the start, each mutation strategy is given an equal probability and is keep the same for a specified number of generations (usually 50) called the learning period. Any time the selection process is conducted, the success or failure of the corresponding mutation strategy is kept track across all the population and across all the generations during the learning period. At the end of the learning period the probabilities of the mutation strategies are updated using the following equation:

with being the cumulated successes of the mutation strategy, being the cumulated failures of the mutation strategy. The numerator being successes of the current mutation strategy multiplied by the sum of the successes and failures of all strategies excluding the current one, while the denominator is the sum all successes and failures multiped by the successes (like the numerator of the equation).

### Control Parameter Adaptation

Much like the mutation strategy, the algorithm can be improved by assigning each individual its own F and CR value, with the vector being as so:

However, while this vector design can be beneficial with having standard static *F* and *CR*, it’s especially effective when combined with a self-adaptive function to adapt the values of both *F* and *CR* over the evolution. The JADE algorithm mentioned in 2.2.1 introduced its own self-adaptive function for CR and F (Sanderson, 2007). It uses two parameters ( and to randomly generate the F and CR values for each individual in the population. CR is generated with a random value along a normal distribution with a mean of , and a standard deviation of 0.1:

The mean is updated using the following formula:

Where *c* is a positive constant between 0 and 1, is a set of all successful crossover probabilities, and mean being an arithmetic mean operation. F is created in a similar manner, with a mixture of uniform distribution between 0 and 1.2 and a normal distribution with a mean of and a standard deviation of 0.1 (). The following formula is used to generate F for an individual:

Where denotes a random collection of one-third of the individuals from the population. The mean is updated using the following formula:

With being the set of all successful mutation factors, and *c* being the same positive constant between 0 and 1. Where L is the Lehmer mean:

The basis of using the previous successful factors is that the better values of CR and F tend to create trial vectors that are more likely to suppose the target vector and so their values are re-used again. As for the mean operation, larger F values are more optimal so the Lehmer mean is used instead of the arithmetic mean to place more weight on larger successful mutation factors.

### Boundary Constraint Handling Techniques

Sometimes the mutation operator may create some individuals where one or more components fall outside the bounds for the problem at hand. When this occurs, the out-of-bound component can either be repaired or substituted so the individual is within the boundaries, and this is done with several techniques. The most common method of boundary constraint is called “Projection”, of which the out-of-bound component becomes the boundary it violated, using the method as follows:

With being the upper boundary of the component and being the lower boundary of the component. In addition to projection, there are several different boundary handling methods (Kreischer, 2017). “Rand Base” replaces the out-of-bounds component with a new value between the base vector’s component and the violated bound:

“Reinitialization” replaces the out-of-bounds component with a randomly generated value inside the boundaries:

“Midpoint Base” replaces the out-of-bounds component with a value midway between the base vector’s component and the violated boundary:

“Midpoint Target” replaces the out-of-bounds component with a value midway between the target vector’s component and the violated boundary:

“Reflection” replaces the out-of-bounds component with a value representing the reflection of the violation with respect to the violated boundary:

“Conservatism” replaces the trial vector with the base vector is one or more components are outside the boundaries:

### Population Size Adaptation

Zhu, Tang, Fang, and Zhang proposed a unique way to handling the population size, by allowing it to be shrunk and grown to both speeds up convergence and to improve population diversity (Zhu, 2013). Their method involved trimming the population when it deemed that some solutions were redundant to the evolutionary process, allowing it to conserve some fitness function evaluations, and increasing the population when it deems that it lacks diversity. It does this by first implementing a status monitor, used to keep track of the solutions and the population. Then an inferior-based population trim with a ranking system is used to remove redundant solutions from the population. Lastly, an elite-based population increase strategy is used to add new solutions to the population. The following pseudocode is the tuning scheme, which executed at the end of each generation.



The status monitor tracks the improvement in fitness in successive generations, if so then the redundancy monitor (RM) variable is incremented. If there is a lack of improvement in the fitness, then the stagnation monitor variable (NM) is incremented. Finally, LM and UM are used to ensure that the population size remains within a pre-determined range.



With min(f(x)) being the minimum value of the fitness function for the current population and MinAll being the minimum value of the fitness function for all previous generations. Once the population has been assessed then if necessary, the cut Strategy function is executed, the pseudocode is as follows:



With IA being the inferior archive, used to keep hold of individuals to be removed. Lastly is the Incremental Strategy function is called to increase the population whenever the population is deemed not diverse enough, the pseudocode is as follows:



With EA being the Elite-archive, a repository to hold individuals before they are changed by the incremental strategy function.



With Maxδ and Minδ being the maximum and minimum values for δ2 set to 5 and 1 respectively. Gmax is the maximum number of generations. Gaussian(0, 1/9) being a random number with a normal distribution with a mean of 0 and a standard deviation of 1/9. C1and C2 are predetermined coefficients set to 0.35 and 0.1 respectively. By using this method, the population can be increased or decreased to fit its current needs.

### Population Initialization Techniques

While the default method of population initialization is to randomly distribute the population over the search space, there are other methods to initialize the population with one such method is opposition-based (S. Rahnamayan, 2008). Once the population is randomly initialized like normal, the opposites of each individual are created using the following equation:

With *l* and *u* being the minimum and maximum bounds of the problem. Due to probability theory, the probability that the solution with being in the interval [*l,x*] or [,*u*] is . Once the opposite population is created it is evaluated against the objective function. With the two populations, the top *NP* candidates are taken between the two populations to create the starting population. This allows the population to start closer to the global optimum and speed up convergence.

With method can be utilized later during the evolution process to help speed up the convergence, this is known as generation-skipping. At the end of each generation, a random number is generated between the values of 0 and 1, and if this value is lower than a predetermined jumping rate (0.3) then generation-skipping is performed. Much like before, the opposite of the population is created, however, instead of being the opposite using the boundaries, it uses the minimum and maximum values of the dimensions:

With min and max being the minimum and maximum values of each dimension from the current population. This is done to ensure that the opposite points do not leave the already narrowed search space and thus doesn’t lose any knowledge obtained so far during the evolutionary process.

### Restart Mechanism

In the paper by Mohamen, a restart mechanism was introduced to mutate a vector if the difference between two successive objective function values is less than a predetermined level δ (E-06) for a predetermined number of generations *K* (25) using one of two mutations (Wagdy, 2014). The mechanism can be expressed as follows:

lse Apply Modified BGA Mutation

Where represent the current and previous objective function values. Which mutation applied is determined by a random value between 0 and 1, allowing each mutation to be used with a 50% probability and while only one mutation can be applied to a vector in the same generation, both can be used on two different vectors. The first mutation is random which replaces the component with a random value within the boundaries using the following equation:

Where is a component randomly chosen from the vector. To perform BGA mutation a scalar value α must first be created using the following equation:

Before mutation, is set to 0, then is mutated to 1 with a probability of 1/16th. This means that on average there will only be one with a value of 1 and so α is only affected by that one . Once α is created then the BGA mutation is given as follows:

The + or – sign is chosen with a probability of 0.5. This is the default BGA mutation formula, the paper introduced a modified version using a random value between 0 and 1 instead of the static value of 0.1, and so the modified formula is as follows:

# Testing

To test the effectiveness of the different aspects and features, they were tested against a number of different optimization functions and problems. With such a large range of different problems, an in-depth analysis can be conducted at the different stages of the project. Functions 1 to 25 are taken from the CEC 2005 real-parameter optimization benchmark suite; these functions are designed to test the effectiveness of an optimization algorithm so that they can be compared against each other and thus were deemed effective to be used for this project. As for the optimization of neural networks, 3 different problems were used (function 26 to 28). These functions are restricted to optimizing only the weights and biases for the neural network, while functions 29 to 31 are the same three problems with this restriction removed which means that the algorithm also optimises the topology of the network as well as the weights.

|  |  |  |  |
| --- | --- | --- | --- |
| Functions | Name | Search Space | fBias |
| f1(x) | Shifted Sphere Function | [-100, 100]D | -450 |
| f2(x) | Shifted Schwefel’s Problem 1.2 | [-100, 100]D | -450 |
| f3(x) | Shifted Rotated High Conditioned Elliptic Function | [-100, 100]D | -450 |
| f4(x) | Shifted Schwefel’s Problem 1.2 with Noise in Fitness | [-100, 100]D | -450 |
| f5(x) | Schwefel’s Problem 2.6 with Global Optimum on Bounds | [-100, 100]D | -310 |
| f6(x) | Shifted Rosenbrock’s Function | [-100, 100]D | 390 |
| f7(x) | Shifted Rotated Griewank’s Function without Bound | [0, 600]D | -180 |
| f8(x) | Shifted Rotated Ackley’s Function with Global Optimum on Bounds | [-32, 32]D | -140 |
| f9(x) | Shifted Rastrigin’s Function | [-100, 100]D | -330 |
| f10(x) | Shifted Rotated Rastrigin’s Function | [-5, 5]D | -330 |
| f11(x) | Shifted Rotated Weierstrass Function | [-0.5, 0.5]D | 90 |
| f12(x) | Schwefel’s Problem 2.13 | [-pi, pi]D | -460 |
| f13(x) | Expanded Extended Griewank’s plus Rosenbrock’s Function (F8F2) | [-3, 1]D | -130 |
| f14(x) | Shifted Rotated Expanded Scaffer’s F6 | [-100, 100]D | -300 |
| f15(x) | Hybrid Composition Function | [-5, 5]D | 120 |
| f16(x) | Rotated Hybrid Composition Function | [-5, 5]D | 120 |
| f17(x) | Rotated Hybrid Composition Function with Noise in Fitness | [-5, 5]D | 120 |
| f18(x) | Rotated Hybrid Composition Function | [-5, 5]D | 10 |
| f19(x) | Rotated Hybrid Composition Function with a Narrow Basin for the Global Optimum | [-5, 5]D | 10 |
| f20(x) | Rotated Hybrid Composition Function with the Global Optimum on the Bounds | [-5, 5]D | 10 |
| f21(x) | Rotated Hybrid Composition Function | [-5, 5]D | 360 |
| f22(x) | Rotated Hybrid Composition Function with High Condition Number Matrix | [-5, 5]D | 360 |
| f23(x) | Non-Continuous Rotated Hybrid Composition Function | [-5, 5]D | 360 |
| f24(x) | Rotated Hybrid Composition Function | [-5, 5]D | 260 |
| f25(x) | Rotated Hybrid Composition Function without Bounds | [-2, 5]D | 260 |
| f26(x) | XOR Problem with weight optimization | [-10, 10]D | 0 |
| f27(x) | Bit Parity Problem with weight optimization | [-10, 10]D | 0 |
| f28(x) | 4-Bit Encoder-Decoder Problem with weight optimization | [-50, 50]D | 0 |
| f29(x) | XOR Problem with topology optimization | [-10, 10]D | 0 |
| f30(x) | Bit Parity Problem with topology optimization | [-10, 10]D | 0 |
| f31(x) | 4-Bit Encoder-Decoder Problem with topology optimization | [-50, 50]D | 0 |

During most of the tests, a for the CEC functions (1 to 25) a dimension size of 10 was used, whereas for the neural networks the number of dimensions varied depending on the number of neurons in the network. The maximum number of evaluations is capped at 1000 times the number of dimensions, with a population size of 5 times the number of dimensions. The default settings for the algorithm are the Rand mutation strategy with the projection boundary constraint with an F value of 0.5 and a CR value of 0.5. For each test, 30 independent runs were used and the average of them was used for the results. All these settings are the default for the algorithm, however, during the different sections, some settings were changed to analyse the different aspects of which when this is done then it is stated so in the report.

For the neural networks, the topology of the network determines the number of dimensions, and so the layout of the network for the three problems is as follows: 2-2-1 for the XOR problem, 3-3-1 for the Bit parity problem and 4-2-4 for the 4-bit encoder-decoder problem. This, of course, excludes the topology optimization problems, which the number of hidden neurons and hidden layers vary.

# Implementation

## Creation of the Basic Differential Evolution

The basic DE algorithm, as detailed in the literature review, is a simple algorithm that uses the concept of natural evolution to create a population of candidate solutions and have them convergence on an optimum. This algorithm, while simple, does have many different aspects of it that need to be considered when creating it such as the control parameters, mutation strategy, bound constraints and population size. Over the years, many researchers have helped further this algorithm by contributing to these different aspects with their own work thus these sections needed an investigation into ways they could be improved and analysed to determine their impact. Below consists of the pseudocode detailing the algorithm.



## Neuroevolution

For basic neural network weight training, all that is required is the creation of the fitness function, while the rest of the algorithm remains untouched. As for the topology optimization, two dimensions are added to the genome, one being the number of neurons in a hidden layer and the other being the number of hidden layers. These two dimensions were given the bounds of 2-4 and 1-3 respectively for the majority of tests. As the increase in the number of neurons means the increase in the number of dimensions, an individual solution contains the number of dimensions same as the maximum number of neurons possible, and so when the topology changes, the weights no longer needed are simply left along and unused in the fitness function, this is done so if the topology was to change again, the weights previously created still remain.

To gain an understanding of whether the DE algorithm is effective in optimizing neural networks, below are scores from when backpropagation is run on the network and when DE is run on it. As for when DE optimise the topology, as there is a lot of combinations to test the with the backpropagation, it was tested with the maximum number of layers and the maximum number of neurons per layer that the DE algorithm could output (e.g. 3 layers and 4 neurons). Unlike most of the tests in this report, the maximum number of evaluations for the DE was set to 20 times the number of dimensions, this is due to backpropagation being able to do it in a few iterations and thus wouldn’t be a fair comparison unless DE also used a low number of generations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Optimization Problem | BP Normal Layout | BP Maximum Layout | DE Weights | DE Topology |
| XOr Problem | 0.289 | 0.500 | 0.223 | 0.152 |
| Bit Parity Problem | 0.492 | 0.305 | 0.27 | 0.221 |
| 4-bit Encoder-Decoder Problem | 0.507 | 0.509 | 0.64 | 0.595 |

As can be seen, with even the default DE, it has the possibility of outperforming backpropagation, with the exception of the 4-bit encoder-decoder problem. This could be due to the difficulty of such a problem of which DE would struggle when capped at a low number of evaluations.

## Investigation into Control Parameters

In the basic DE algorithm, during the creation of the trial vectors, two control parameters are used F and CR, which in turn scale the difference vector and determines the number of components taken respectively. While these values lie in the range of 0.1 and 0.9, their effects change drastically even when changed by the slightest bit.

### Scalar F

The scalar F is used to scale the difference vector created during the mutation strategy, this is used to determine the influence the difference vector has in the creation of donor vector. As can be seen in the graph below, an F value of 0.9 (left) and 0.1 (right) have very different effects on the creation of the donor vector



With this in mind, tests were conducted using the different values of F ranging from 0.1 to 0.9 to determine their effects. These tests were run with a population size of 5 times the number of dimensions, for the CEC benchmark marks, it was 10 dimensions, as for the neural network the number of dimensions varies for the problems as the number of input and output neurons differ. The mean squared errors were collected and normalised so different problems could be compared against each other. However, as CR is used to determine how many components are taken from the donor, different CR values have an impact on the effectiveness of different F values and so the tests were run on every combination of F with CR of increments of 0.1 for a total of 81 tests per problem, and so the following values are the average for F across the different CR values.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Scalar F | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| CEC Optimization | 0.507 | 0.439 | 0.411 | 0.422 | 0.461 | 0.515 | 0.555 | 0.603 | 0.651 |
| Neural Network Weights Only | 0.314 | 0.368 | 0.440 | 0.507 | 0.576 | 0.639 | 0.669 | 0.680 | 0.699 |
| Neural Network Weights and Topology | 0.347 | 0.374 | 0.415 | 0.476 | 0.554 | 0.589 | 0.585 | 0.598 | 0.596 |

As can be seen, by both the table and graph, the scalar F has an important impact on the effectiveness of the differential evolution. However, as real-life problems, there is not a single value of F that always performs better than the others, instead, the best value of F varies on the problem at hand. For CEC benchmarking the best value lies at 0.3 and worsens as it gets larger, however for the neural networks the best value is 0.1.

### Crossover Rate CR

Much like the scalar F, CR has an important impact on the creation of the trial vector and thus the effectiveness of the algorithm. The graph shows the crossover of xi and vi, when one component is swapped. It shows the two different trial vectors when they swap different components, while it doesn’t show the direct effects of CR as its only 2 dimensions, it shows the effects of the crossover which is affected by the crossover rate.

Like the values for F, there are averages obtained from the large benchmark which tested the different values of F and CR and so the CR values are averages from the different F values.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Crossover rate Cr | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| CEC Optimization | 0.558 | 0.555 | 0.549 | 0.536 | 0.499 | 0.476 | 0.454 | 0.464 | 0.475 |
| Neural Network Weights Only | 0.151 | 0.293 | 0.449 | 0.616 | 0.702 | 0.744 | 0.705 | 0.642 | 0.586 |
| Neural Network Weights and Topology | 0.082 | 0.153 | 0.319 | 0.503 | 0.645 | 0.736 | 0.749 | 0.691 | 0.657 |

As can be seen by the graph, for the CEC benchmarks, the crossover rate has less of an impact on its own, as these values are averages across the different F values. However, when looking at the impact of different CR when different F values are used, then the impact of CR can be seen (refer to a table in the appendix-A for full test results). An example would be with a scalar value of 0.4, a CR value of 0.1 gives an error of 0.431 whereas a CR value of 0.9 gives an error of 0.7. However, the impact of CR when training neural networks is much more substantial, ranging from a normalised error rate of 0.1 to 0.75.

### Control Parameter Recommendations

The previous tests conducted helped high light the effective static values of F and CR for the different problems, from the large scale test an optimum F and CR value for each of the three areas was obtained.

|  |  |  |
| --- | --- | --- |
| Control Parameter | F | Cr |
| CEC Optimization | 0.3 | 0.8 |
| Neural Network Weights Only | 0.2 | 0.1 |
| Neural Network Weights and Topology | 0.1 | 0.1 |

However, if the intention of the differential evolution algorithm is to optimise varying different problems rather than separating them into different areas, then the following results highlight the changes in the error when F and CR changes.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Control Parameter | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| F | 0.264 | 0.334 | 0.439 | 0.551 | 0.615 | 0.652 | 0.636 | 0.599 | 0.573 |
| CR | 0.389 | 0.394 | 0.422 | 0.468 | 0.530 | 0.581 | 0.603 | 0.627 | 0.649 |

Like before, the values of F and CR are averages across the different values of its counterpart. While the results above are for different values of F and CR separately, the tests showed that the optimum value of F and CR of 0.2 and 0.1 respectively.

### Adaptive Control Parameters

As described in the related work, the JADE algorithm introduced a method to adapt the control parameters during the evolution process by using previous successful F and CR values to generate new ones each generation. In this adaptive function, much like others, each individual is given its own F and CR value instead of a singular value affecting the population. Smaller values of F allows individuals to exploit the parent when creating the donor, whereas larger values allow better exploration, and so different values of F are most effective at different stages of the evolution process and so an adaptive function allows the value to change to accommodate the different stages. For most functions, the difficulty is determining a suitable equation to adapt the control parameters, and so some functions are more effective based on that. However, JADE instead uses the previous successful values to adapt it, which in turn uses the information obtained so far to adapt the parameters, allowing it to be fine-tuned to the problem at hand and the current stage of the evolution process.

While JADE adapts both control parameters at the same time, they are adapted independently from each other, and so below are the results of different static F values when CR is adapted to further highlight the impact of F without the influence of CR much like before when averages were taken for every possible combination of CR with each value of F.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Scalar F | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| CEC Optimization | 0.381 | 0.358 | 0.306 | 0.367 | 0.375 | 0.456 | 0.483 | 0.515 | 0.55 |
| Neural Network Weights Only | 0.206 | 0.255 | 0.196 | 0.244 | 0.254 | 0.248 | 0.235 | 0.238 | 0.278 |
| Neural Network Weights and Topology | 0.264 | 0.235 | 0.246 | 0.248 | 0.24 | 0.236 | 0.236 | 0.248 | 0.23 |

As JADE generates its CR values based on previous successful CR values, it adapts the CR to an optimum for the respective F value, and so the difference between the error values for the different F values is much lower compared to before. Much for F, the following results are obtained when F is adapted using JADE’s method while CR is static for each value between 0.1 and 0.9 of increments of 0.1.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Crossover rate Cr | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| CEC Optimization | 0.588 | 0.583 | 0.586 | 0.593 | 0.54 | 0.441 | 0.369 | 0.323 | 0.235 |
| Neural Network Weights Only | 0.232 | 0.322 | 0.486 | 0.594 | 0.792 | 0.779 | 0.709 | 0.617 | 0.535 |
| Neural Network Weights and Topology | 0.166 | 0.301 | 0.436 | 0.539 | 0.577 | 0.604 | 0.607 | 0.612 | 0.597 |

As can be seen, by the table, the values of CR do not differ much compared to the static tests when F is self-adapted, this in conjunction to the adaptive CR test showing the precedence that the crossover rate CR has over the scalar value F and that F has little impact when CR is self-adaptive. The final test of the adaptive function is to adapt both the F and CR control parameters, below is a table which shows the results of the adaptive function against the scores of the optimum static values stated in the control parameter recommendations.

|  |  |  |
| --- | --- | --- |
| Control | Optimum Static | Adaptive |
| CEC Optimization | 0.319 | 0.416 |
| Neural Network Weights Only | 0.130 | 0.214 |
| Neural Network Weights and Topology | 0.080 | 0.212 |
| General Optimization | 0.244 | 0.280 |

While it didn’t beat it outright, those best results for the static F and CR were obtained from 81 tests which takes a lot of time, and the purpose of control parameter adaption is to remove the need to trial and error the parameters until the best one is obtained. On top of that, further tests revealed that having the control parameters adapt over time showed better results when used for optimizations over a long time, due to it allowing itself to fine-tune itself to the problem at hand, having more time allows it to adapt to the current stage in the evolutionary process and thus perform better. The difference between the optimum static and adaptive for general optimization is much less compared to the other areas, and while it’s still worse, that small difference means that although worse, adaptive F and CR is the better option as it can adapt to the problem at hand in case a new optimization problem tested is different to the ones used in this paper.

### Self-Optimising Control Parameters

All adaptive control parameter functions work externally from the evolution process, they adjust the control parameter at the end of a generation once the mutation-crossover-selection stage is completed. An alternative is to include the control parameters into the optimisation process much like the topology of the neural network, by having components (or dimensions) of the genome dedicated to the control parameters.

|  |  |
| --- | --- |
| Control | Self-Optimising |
| CEC Optimization | 0.432 |
| Neural Network Weights Only | 0.419 |
| Neural Network Weights and Topology | 0.393 |
| General Optimization | 0.415 |

While this method didn’t beat the adaptive function outright, it did manage to come close whilst also beating some of the static configurations. This highlights the potential for such a method if further work was put into it.

## Mutation Strategies

The control parameters F and CR are not the only aspects of the algorithm that need to be considered when applying the differential evolution algorithm to a problem, one big aspect is the mutation strategy used to create the donor vector. While there are many different ones proposed over the years, only four are considered in this paper: Rand, Best, Current-To-Best, and Current-To-P-Best, with the first three being from the original differential evolution algorithm and the fourth is from the JADE algorithm. The fifth mutation strategy (P-Best) takes the concept of P-Best from JADE’s Current-To-P-Best and applies it to the best mutation strategy. Lastly, the sixth mutation strategy (Current-To-P-Worst) follows a similar concept to Current-To-P-Best but instead of selecting one randomly from the top p% it selects one randomly from the population excluding the top p%, this is to allow the mutation strategy to be a kind of opposite to Current-To-P-Best as the use of the best individuals can cause premature convergence. While some mutations strategies like Rand is explorative while others like Best are exploitative and so they perform differently based on the problem at hand and so further investigation is needed. For the following tests, a population size of 5 times dimension size, F value of 0.5, CR value of 0.5, a maximum evaluation of 1000 times dimension size, and a total of 30 runs per variation.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Mutation strategy | Best | Current-To- Best | Rand | Current-To-P-Best | p-best | Current-To-p-worst |
| CEC Optimization | 0.5233 | 0.4807 | 0.7668 | 0.3938 | 0.3927 | 0.7151 |
| Neural Network Weights Only | 0.3895 | 0.2483 | 0.6918 | 0.5062 | 0.6019 | 0.7623 |
| Neural Network Weights and Topology | 0.0536 | 0.2590 | 0.9389 | 0.6012 | 0.6383 | 0.7043 |

Based on its nature, Rand performed the worst as while it is effective at exploring, it is slow to converge and so would require more evaluations compared to the greedier solutions of Best and Current-To-Best. While Best is considered the greediest of solutions, it doesn’t allows perform the best as it has a habit of converging to local minima’s and so Current-To-Best tends to perform better as it’s a nice midpoint between explorative and exploitative. However, similar to Best, Current-To-Best uses the best individual in the mutation strategy and so still converges to local minimum’s, although slower, and so Current-To-P-Best improves on it by using one of the top 10% individuals which makes sure that the donor vectors are not too similar to each other thus allowing it to perform better in most cases, the same can be applied to P-Best. While overall the Current-To-P-Worst performed badly, it is due to its nature to avoid premature convergence, much like Rand, however, it did prove to perform better in a couple of the problems than the others although not enough to show any real difference in the overall score. However, for training neural networks, greedier strategies work best due to the incredibly high number of dimensions and the simplicity of the problem, it is only weighted whereas the CEC benchmarks are complex mathematical functions.

### Mutation Strategy Recommendation

The previous tests highlighted which mutation strategies are more effective for the different optimization problem areas, with P-Best being the optimum choice for mathematical optimization, Current-To-Best for Neural Network Weight optimization and Best for Neural Network Topology optimization. Rand is never the optimum because like mentioned before, due to its nature, it has an incredibly slow converge speed. However, if the intent is to create one algorithm for all problems, general optimization, then the following data shows the effectiveness of the mutation strategies which results in Current-To-P-Best being optimal.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Mutation strategy | Best | Current-To- Best | Rand | Current-To-P-Best | p-best | Current-To-p-worst |
| General Optimization | 0.4609 | 0.4337 | 0.7768 | 0.4269 | 0.4397 | 0.7189 |

### Adaptive Mutation Strategy Selection

The mutation strategy selection method proposed by SaDE uses two or more mutation strategies, assigns an individual one of the two strategies. At the start and for the first 50 generations, each mutation strategy has an equal chance of being selected, during this period it keeps track of the successes and failures of the strategies and after the 50 generations it changes the percentage based on the success rate, of which it erases the number of successes and failures obtained during the learning period. At the end of each generation, the process repeats, updating the percentage, erasing the stored data, and assigning a new mutation strategy to each individual. This allows the use of two strategies while allows one strategy to take precedence over the over during different stages of the evolution process, for example, a greedy strategy at the start to allow it to converge quickly and a less greedy strategy once the population is close together.

SaDE uses Rand and Current-To-Best in its mutation strategy pool, tests were conducted to different combinations of the four mutation strategies selected from a pool size of 2, using the same parameter settings as the previous mutation tests. (CT standing for Current-To). Due to its poor performance, Current-To-P-Worst was used in the assessment of adaptive mutation strategy selection.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Mutation strategy | Best & ct-Best | Best & Rand | Best & ct-P-Best | ct-best & Rand | P-Best & CT-p-best | rand & ct-p-best | P-Best & Rand |
| CEC Optimization | 0.4451 | 0.4522 | 0.3859 | 0.4615 | 0.3438 | 0.3548 | 0.4486 |
| Neural Network Weights Only | 0.3251 | 0.3301 | 0.2948 | 0.5317 | 0.4709 | 0.6303 | 0.5774 |
| Neural Network Weights and Topology | 0.0789 | 0.4510 | 0.2052 | 0.4839 | 0.5969 | 0.7919 | 0.7845 |
| General Optimization | 0.3948 | 0.4394 | 0.3578 | 0.4711 | 0.3831 | 0.4285 | 0.4966 |

As for using a static strategy, the optimum strategy depends on the area of optimization and so the same applies to the adaptive strategy selection which the optimum pool of strategies changes. CEC optimization benefits from a pool of Current-To-P-Best and P-Best, Neural Network Weight optimization benefits from Best and Current-To-P-Best, and Neural Network Topology optimization benefit from Best and Current-To-Best. However, if general optimization is the goal, then the pool of Best and Current-To-P-Best is optimal.

## Population Size

In the default Differential Evolution, the third and final control parameter is the population size NP. Unlike the other aspects, the population size has less research conducted on it, as well as very little solutions to improve the control parameter. In the original paper by Storn and Price, they suggest using a population size of 5 times the number of dimensions (5D) or 10 times the number of dimensions (10D). With this in mind, tests were conducted for different values of NP ranging from 1D to 10D.

Population size is an important control parameter as it determines the diversity of the population, by having a larger population then there is a smaller chance of the population prematurely converging to a local minimum. However, as most benchmarks are based on a maximum number of fitness evaluations rather than a maximum number of generations, having a larger population, in turn, results in fewer generations before the maximum evaluations are met.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Population size | 1D | 2D | 3D | 4D | 5D | 6D | 7D | 8D | 9D | 10D |
| CEC Optimization | 0.805 | 0.521 | 0.396 | 0.443 | 0.423 | 0.435 | 0.455 | 0.457 | 0.487 | 0.474 |
| Neural Network Weights Only | 0.369 | 0.319 | 0.487 | 0.512 | 0.508 | 0.562 | 0.615 | 0.607 | 0.622 | 0.694 |
| Neural Network Weights and Topology | 0.389 | 0.505 | 0.538 | 0.557 | 0.620 | 0.582 | 0.574 | 0.621 | 0.606 | 0.644 |

For CEC optimization, the population size has to be large enough to prevent premature convergence as most of the benchmarks are complex and have multiple local minimums as mentioned before. However, for training neural networks, smaller population size is optimum, to allow it to have more generations.

### Population Size Recommendation

The tests above highlight the effects of changing population size when the number of evaluations is limited, with CEC optimization benefitting from a population size of 3 times the number of dimensions, neural network weight optimization benefitting from 2 times the number of dimensions, and neural network topology optimization benefitting from the size being the number of dimensions. If general optimization is the goal of the algorithm, then the following results are the scores for the different sizes for all optimization, with 2 times the number of dimensions being the most optimum.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Population size | 1D | 2D | 3D | 4D | 5D | 6D | 7D | 8D | 9D | 10D |
| General Optimization | 0.521 | 0.448 | 0.474 | 0.504 | 0.517 | 0.526 | 0.548 | 0.562 | 0.572 | 0.604 |

### Adaptive Population Size

As population size has a massive impact on the effectiveness of the algorithm as it determines both the diversity of the population and the maximum number of generations, adaptive population size can help boost performance. The main premise of the adaptation is to increase the population size when the population is deemed not diverse enough and decrease to the size when the population is performing adequately to reduce the number of evaluations it uses up. As mentioned in the literature review, … proposed a way to achieve this which if the best solution of the population beats the previous best for a number of subsequent generations then it cuts some of the worst from population as it deems them redundant, and if the best solution doesn’t improve for a number of generations then it adds in new members which are created as variations from the current population. This is all done while keeping within a predetermined minimum and maximum for the population size, and while the paper uses an initial population size of 5D, a minimum of 2.5D and a maximum of 10 D, as the previous section highlighted that neural networks benefit from a smaller population size different values were also tested. The other values were an initial population size of 2D, minimum of 1D and maximum of 3D, the results of both tests are compared below against the previously determined optimums for the different areas.

|  |  |  |  |
| --- | --- | --- | --- |
| Population size | Optimum | Adapt: 2.5D – 10D | ADapt: 1d – 3d |
| CEC Optimization | 0.396 | 0.572 | 0.263 |
| Neural Network Weights Only | 0.319 | 0.585 | 0.346 |
| Neural Network Weights and Topology | 0.389 | 0.372 | 0.094 |
| General Optimization | 0.448 | 0.510 | 0.234 |

As seen from the results, having an adaptive population can have an adverse effect on the population if the range of the population isn’t set correctly for the problems at hand, however, when it is set correctly, it can improve the performance considerably especially for neural network topology optimization.

## Population Initialization

Population initialisation can affect the entire process, with it being the starting point if the population, how it’s initialised can affect the rate of convergence. In the basic DE algorithm and most variations, the initial population is randomly generated between the initialisation range given for the problem at hand, however, there are a few alternatives to give the DE algorithm a better starting point. One such method is known as opposition-based initialisation which uses the theory of opposition to create an opposite population after the initial creation and selects the best from the two separate populations.

The graph below depicts the process, the blue and red lines in the left graph are the lines that separate a dimension in two, with these two lines the opposite of an individual represented by the white individuals to create the opposite individuals represented by the black individuals with the black cross representing the optimum. The NP best individuals are picked from the two populations, represented by the right graph, of which the 4 remaining closest individuals to the optimum. By applying this method, the initial population starts closer to the optimum and so can help speed up convergence.



The same method can be applied later in the evolution process, also known as generation-skipping, which creates another opposite population however instead of reflecting the points on the initial bounds of the problem, as the population slowly converges to the optimum and so it reflects them on the narrowed search space so it doesn’t lose the important information obtained from the parents. At the end of each generation a random number between 0 and 1 is created, if less than a predetermined value of 0.3, then generation-skipping is performed.



|  |  |  |  |
| --- | --- | --- | --- |
| Population Initilization | Normal | Opposition | Opposition with Gen Skip |
| CEC Optimization | 0.666753 | 0.580278 | 0.406466 |
| Neural Network Weights Only | 0.674729 | 0.445792 | 0.338514 |
| Neural Network Weights and Topology | 0.65274 | 0.72642 | 0.207442 |
| General Optimization | 0.664741 | 0.584163 | 0.317474 |

When the number of evaluations is limited, starting closer to the optimum can allow the algorithm to perform better, evident by the results for CEC, neural network weight and general optimization. However, for topology optimization, it has an adverse effect. As for generation-skipping, by performing the process approximately every 3-4 generations it allows the evolutionary process to skip a lot of the early convergence stages thus allowing it to converge much sooner when the number of evaluations is limited.

## Boundary Constraint Handling

A lot of optimization problems have some form of bounds, restrictions on the data so they don’t exceed certain values. For example, if the problem at hand has something to do with mass, the numbers shouldn’t be negative as mass cannot be negative and so boundary constraints allow the restriction of the data. The default method of handling boundary violations is to replace the violating data with the boundary its violating, this is called projection. This method, while simple, also means that some of the data is lost which can affect performance. If a component violates the boundary be a small amount or large amount the resulting value after projection is the same and so other methods were created to handle boundary constraints in a potentially better way.

Below depicts the outcomes of the different boundary constrain handling methods. With the donor vector violating the bound (black square) by one dimension the boundary constraint handling method is performed and the 7 different vectors that are created from them. In order from 1 to 7, Projection, Reinitialization, Rand Base, Midpoint Base, Midpoint Target, Reflection, and Conservatism with them labelling by their respective number. As some methods use a random value, the same random value was used between them of 0.4.



As can be seen from the illustration above, the different methods produce very different vectors, all of these are only when one dimension is violated hence why the Y value for all of them (except Conservatism) being the same so when more than one dimension is violated the amending vector is very different depending on which method is used. Conservatism varies from the others as instead of changing just the violating dimension, it replaces the entire vector with the Base vector whenever there’s a violation thus making it the only method to result in a vector with a different Y value. The results below are from 8 different types of bound constraints, run on the usual setting for the tests, a population size of 5D with the rand mutation strategy and a maximum number of evaluations of 1000D.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Boundary Constraint | Conservatism | Projection | Reflection | Rand Base |
| CEC Optimization | 0.453 | 0.467 | 0.465 | 0.539 |
| Neural Network Weights Only | 0.087 | 0.573 | 0.596 | 0.559 |
| Neural Network Weights and Topology | 0.022 | 0.445 | 0.777 | 0.700 |
| General Optimization | 0.371 | 0.476 | 0.511 | 0.558 |

|  |  |  |  |
| --- | --- | --- | --- |
| Boundary Constraint | Midpoint Target | Reinitialization | Midpoint Base |
| CEC Optimization | 0.392 | 0.657 | 0.469 |
| Neural Network Weights Only | 0.797 | 0.501 | 0.551 |
| Neural Network Weights and Topology | 0.740 | 0.460 | 0.547 |
| General Optimization | 0.470 | 0.621 | 0.485 |

As shown by the results, for CEC optimization the optimum method of handling boundary constraints is Midpoint Target, which assigns the violating dimension with a value that lies in the middle of the target vector and the violated boundary. However, for both types of neural network optimization conservatism performed several times better than the other methods, where if one or more of the dimensions violate a boundary the vector becomes a copy of the base vector. Due to this high performance, it meant that even for general optimization, Conservatism is the optimum method for handling boundary constraints.

## Restart Mechanism

In an attempt to make the algorithm better at avoiding local minimums, a restart mechanism was developed. Any time the selection process is performed, if the target vector (parent) remains or if the trial vector (offspring) only performs better by a small predetermined value (10e-6), if this happens for 25 generations in a row then one component from that vector is replaced using the restart mechanism described in the literature review. This is used to attempt to move a vector out of a minimum in case the minimum it is stuck in is the local minima rather than the global. The following results show the outcome of using the restart mechanism with the default DE algorithm.

|  |  |  |
| --- | --- | --- |
| Restart mechanism with rand mutation | Without | With |
| CEC Optimization | 0.615028 | 0.680684 |
| Neural Network Weights Only | 0.586582 | 0.882509 |
| Neural Network Weights and Topology | 0.689149 | 0.852197 |
| General Optimization | 0.619448 | 0.716813 |

As can be seen, the restart mechanism can have an adverse effect on the outcome of the algorithm, this is due to the already slow convergence of using the Rand mutation strategy. The restart mechanism can indeed improve the performance of the algorithm but at the cost of convergence speed thus pairing it with the Rand mutation strategy can have an adverse effect, however, if paired with a greedy mutation strategy such as Best then the restart mechanism can improve the performance. However, as can be seen, it still has a negative effect on topology optimization, this is due to the fact that topology optimization benefits from convergence speed more than the other problems.

|  |  |  |
| --- | --- | --- |
| Restart mechanism with best mutation | Without | With |
| CEC Optimization | 0.382558 | 0.34082 |
| Neural Network Weights Only | 0.433487 | 0.107227 |
| Neural Network Weights and Topology | 0.09181 | 0.399792 |
| General Optimization | 0.359349 | 0.323922 |

## Final Results

To test the full extent of these features and how they interact with each other, a final set of benchmarks were conducts with a number of features enabled with different configurations. 3 configurations were tested to see the effects of the different implemented features.

|  |  |  |  |
| --- | --- | --- | --- |
| Features | Config #1 | Config #2 | Config #3 |
| Mutation | Adaptive (P-Best & Current-To-P-Best) | P-Best | Current-To-P-Best |
| Boundary Constraints | Conservatism | Conservatism | Projection |
| Population Size | Adaptive (1D to 3D) | Adaptive (1D to 3D) | Adaptive (1D to 3D) |
| POpulation Initialisation | Opposition-Based | Opposition-Based | Opposition-Based |
| Scalar F | 0.1 | JADE Adaptive | Self-Optimising |
| Crossover Rate CR | 0.1 | JADE Adaptive | Self-Optimising |
| Restart MEchanism | Enabled | Enabled | Enabled |

However, for these tests, a higher dimensionality was used for the CEC benchmarks (30) and the maximum number of fitness function evaluations was increased to 10000 times the number of dimensions. To just an understanding of their performance, the same benchmarks were conducted on the basic DE algorithm and JADE, with the best out of the five made bold for each problem. The following is the mean and standard deviation of 30 runs.





With that in mind, as these values were obtained from a maximum number of evaluations of 10000 times the number of dimensions, another set of benchmarks needed to be conducted so the algorithms could be compared to backpropagation and so the cap was reduced to 20 times the number of dimensions.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Optimization Problem | DE | Jade | Config #1 | Config #2 | COnfig #3 |
| XOr Problem – Weights | 0.223 | 0.209 | 0.208 | 0.199 | **0.194** |
| Bit Parity Problem – Weights | 0.27 | 0.278 | 0.255 | **0.253** | 0.259 |
| 4-bit Encoder-Decoder Problem – Weights | 0.64 | 0.626 | 0.608 | 0.61 | **0.599** |
| XOr Problem – Topology | 0.152 | 0.149 | **0.104** | 0.138 | 0.129 |
| Bit Parity Problem – Topology | 0.221 | 0.226 | 0.21 | **0.208** | 0.209 |
| 4-bit Encoder-Decoder Problem - Topology | 0.595 | 0.587 | **0.573** | 0.601 | 0.584 |

While the three configurations were successful in beating backpropagation, excluding the encoder-decoder problem, the features implemented into the configurations were designed for a much larger number of generations and so performance could be improved if they were adapted to work efficiently at a smaller number of generations. For example, adaptive mutation strategy selection changes the probability for the mutation strategies after 50 generations however the algorithm doesn’t perform 50 generations when the number of evaluations is this limited. The performance problems with the encoder-decoder problem highlight the fact that backpropagation is an effective algorithm because it can be performed quickly, while DE can outperform it but only if given a reasonable amount of generations thus taking longer.

# Project Management

## Project Schedule

The project work breakdown is split into four sections which represent four different types of tasks: research being any necessary research into any area needed for the project; Implementation being the creation of any feature discovered during the research stage; analysis being the in-depth assessment into the effects of a feature; and the report being the write up for that area. For the analysis stages, a large amount of the time spent on it was allowing the algorithm to run the necessary benchmarks and so while no work could be done on the algorithm itself during this time, time was able to be spent on writing up the results from previous benchmarks thus resulting in some overlapping in the work breakdown timeframes.

|  |  |  |  |
| --- | --- | --- | --- |
| Section | Name | Start | Finish |
| Research | Differential Evolution | 23rd May | 9th June |
| Report | Project Brief | 29th May | 31st May |
| Implementation | Differential Evolution | 10th June | 13th June |
| Research | Mutation Strategies | 15th June | 16th June |
| Implementation | Mutation Strategies | 15th June | 16th June |
| Research | Adaptive Mutation Strategy Selection | 17th June | 18th June |
| Implementation | Adaptive Mutation Strategy Selection | 17th June | 18th June |
| Research | Control Parameters and Adaptive functions | 20nd June | 21rd June |
| Implementation | Adaptive Control Parameters | 22nd June | 23rd June |
| Research | Population Size and adaptive population size | 24th June | 25th June |
| Implementation | Population Size and adaptive population size | 24th June | 25th June |
| Implementation | Neural Network Weight Optimization | 26th June | 26th June |
| Implementation | Neural Network Topology Optimization | 26th June | 27th June |
| Research | Population Initialisation Methods | 29th June | 1st July |
| Implementation | Opposition-Based Initialisation | 1st July | 2nd July |
| Analysis | Initial Analysis of Control Parameters | 4th July | 10th July |
| Implementation | Self-Optimizing Control Parameters | 12th July | 13th July |
| Research | Boundary Constraint Handling Methods | 14th July | 15th July |
| Implementation | Boundary Constraint Handling Methods | 15th July | 17th July |
| Research | Local Minima Avoidance Methods | 19th July | 20th July |
| Implementation | Restart Mechanism | 19th July | 20th July |
| Report | Literature Review | 25th July | 4th Aug |
| Implementation | Code Revision to Improve Efficiency | 1st Aug | 3rd Aug |
| Analysis | Control Parameters | 5th Aug | 7th Aug |
| Report | Control Parameters | 7th Aug | 8th Aug |
| Analysis | Adaptive Control Parameters and Self-Optimizing Control Parameters | 7th Aug | 8th Aug |
| Report | Adaptive Control Parameters and Self-Optimizing Control Parameters | 9th Aug | 10th Aug |
| Analysis | Mutation Strategies | 9th Aug | 10th Aug |
| Report | Presentation | 10th Aug | 11th Aug |
| Report | Mutation Strategies | 12th Aug | 13th Aug |
| Analysis | Adaptive Mutation Strategies | 13th Aug | 14th Aug |
| Report | Adaptive Mutation Strategies | 14th Aug | 14th Aug |
| Analysis | Population Size and Adaptive Population Size | 14th Aug | 15th Aug |
| Report | Population Size and Adaptive Population Size | 15th Aug | 15th Aug |
| Analysis | Population Initialisation | 15th Aug | 15th Aug |
| Report | Population Initialisation | 15th Aug | 15th Aug |
| Analysis | Boundary Constraint Handling Methods | 15th Aug | 16th Aug |
| Report | Boundary Constraint Handling Methods | 16th Aug | 16th Aug |
| Analysis | Restart Mechanism | 16th Aug | 16th Aug |
| Report | Restart Mechanism | 16th Aug | 16th Aug |
| Report | Final Sections (Conclusion, Critical Appraisal, Student Reflection etc.) | 16th Aug | 19th Aug |

As can be seen by the breakdown, the order of which tasks were completed varied from the original plan outlined in the project brief. Originally it was decided that the algorithm would be constructed with all its additional features before adapting it to work with a neural network. However, this plan was later changed to include the neural network earlier in the development stage due to its simplicity to be implemented, this, in turn, meant that when implementing a feature, it simply only had to be designed to work with the algorithm at the time rather than the initial plan to adapt any new features to work with the neural network.

In addition to this change in order, was a slight change into the intended outcomes. When the project brief was written, the research into the differential evolution algorithm was not finished and so at the time, the potential improvements to the algorithm were not stated. As research was conducted, the areas that needed further research and investigation came to light which in turn changed the plan of events.

The largest discrepancy between the original plan and the breakdown is the estimated number of hours. While the total number was fairly accurate, the distribution of those hours between the sections was wrong. This was mainly for two reasons. The first being the implementation of the neural network with the DE, as this stage itself, was fairly straight forward and didn’t require a lot of time, the section for it in the plan was completely overestimated, while the number of hours needed for the algorithm improvement was underestimated. The second reason was due to the amount of time it took to run the benchmarks, as these tests would take up a large amount of time and had the potential to delay aspects of the project. Without building the project, it was difficult to determine how long it would take to computationally run the benchmarks and so the estimated number of hours was wrong. Before the code revision, it would have taken a lot longer than planned to run these benchmarks with some tests taking a couple of days. This was resolved when the code was revised to be more time and computationally efficient.

## Risk Management

Due to the nature of the project being purely software, one large risk to such a project is the loss of data, which includes the project code itself, benchmark results and the written report. If any of the data was lost then that could result in lost time attempting to either retrieve it or recreate it, which in the case of either the code or the report, could result in a large amount of time lost. To avoid such a risk, several copies of all the data, code and report was made across different storage devices including the cloud. This means that in the case of the primary drive being lost then there are copies so nothing is lost thus removing (or at least limiting) any need to redo any work. With this in mind, the primary drive was not lost and thus nothing was lost, and nothing needed to be redone.

The schedule for the project was created at the start of the project before any research or development had occurred, due to this creating an accurate estimation for the schedule is difficult and so the project had the risk of overrunning the allotted time frame for it. If this was to occur, then some aspects and or sections of the project would have to been removed to allow the project to be completed in time. To help avoid this, weekly or bi-weekly meetings with the supervisor were conducted to not only relay progress to him but also get feedback on the work. With each meeting, further work was planned for the next meeting in mind, this ensures that the project was completed within the time frame.

## Quality Management

To be able to run the differential evolution algorithm, a fitness function must be created for it. This fitness function is tested with the solutions within the population and a cost is created. This can be used to not only conduct the algorithm itself but to compare the variations against one another, with the outcome of the variations being a cost of said fitness function. While this allows the comparison of algorithms within this project, it cannot be used to compare them against other peoples work unless the same optimization problem is attempted. To help identify the effectiveness of the algorithm and the aspects tested, the fitness functions chosen for mathematical optimization were selected from the CEC (conference on evolutionary computation) 2005 benchmark suite. These benchmarks were designed to test the extent of an optimization algorithm in a variety of different problems to allow accurate comparison between algorithms and evaluation of said algorithms.

As for neural network optimization, to create a fitness function for neural networks the data must have an intended output to compare against the output of the network, also known as supervised learning. With that, once a solution from the DE is attempted, the output of that solution is compared against the intended output and a mean squared error can be calculated, this allows for the comparisons of algorithms and allows for the evaluation of the algorithm.

## Social, Legal, Ethical and Professional Considerations

This project does not use any form of personal data nor does it use any proprietary data, therefore, the data protection and privacy laws do not affect this project, this is further supported by the fact that this project isn’t a business project nor is it commissioned as a product. Any work used from other researchers has been mentioned and fully referenced with the intent that none of their work is considered stolen or passed off as work of this paper’s author. Finally, as this project is simply a mathematical optimization project for the purpose of guiding others to creating a suitable algorithm for their purposes, the social implications are limited to just that, helping others create a differential evolution algorithm.

# Critical Appraisal

The creation of the basic differential evolution algorithm in MATLAB proved successful without hindrance. The algorithm was capable of optimizing all the CEC benchmarks and the neural network weights. Extra work was required to allow the algorithm to optimise the topology of the network. As the algorithm was the main premise of the project, simple topology optimisation was developed to prove that the DE algorithm could be used. This simple optimisation was to have each hidden layer have the same number of neurons and it simply optimises the number of neurons as well as the number of layers. This solution sufficed as a proof of concept that the DE algorithm could be used to optimise the topology while optimising the weights at the same time.

Originally to test the fitness of a candidate solution when optimising the weights of a neural network, the weights had to be inputted into a neural network created through the MATLAB Neural Net toolbox. This was done originally due to its simplicity, however, it proved to be too inefficient as this process had to be conducted several hundreds of times. So, a function was constructed that performed a simple forward pass with the weights, biases and input data to simulate a neural network. This performed the exact same process as the toolbox did but in a much quicker timeframe.

Once all of this was done, the algorithm was to be improved with a number of features to be tested for their effectiveness. The first stage was mutation strategies, of which had included half a dozen new mutation strategies from various papers. However, a lot of them were outdone by others and proved to be ineffective which meant they were removed. An attempt to create a unique mutation strategy was conducted but it proved unsuccessful and was also removed due to its poor performance. Once several mutation strategies were added, an investigation into potential adaptation methods was conducted to find a way to improve the mutation process. This resulted in an adaptive mutation strategy selection method being added into the program which was tested with varying pools of mutation strategies.

After that was an investigation into possible ways to improve the control parameters F and CR, which entailed researching different adaptive functions to be used. While a couple were considered, only one was selected for the project. On top of that, self-optimisation of the control parameters was attempted, and while it performed poorly even compared to the static values, it is possible that further work into it could allow it to surpass other methods.

Next was the research into methods that could adapt the population size during the evolution process, where one such method was selected and implemented. On top of this, research was conducted to find alternative methods to initialise the population. A number of methods were found however most of them were designed for certain types of problems in mind (excluding neural networks) and so only two methods were selected (opposition and quasi-opposition) due to them being designed for general optimization rather than specific. Both methods were implemented into the algorithm including their generation-skipping feature which wasn’t originally considered.

All but 2 optimization problems used in the project had some form of boundaries set on the problem, and so 6 boundary constraint handling methods excluding the original one was implemented to test their effectiveness. After research into ways to avoid local minimums, a restart mechanism was implemented into the program to potentially improve performance on some optimization problems.

Once everything was added, several different tests were conducted on each newly added feature to analyse their effectiveness and determine their impact on the algorithm, with all results and findings documented in this report. With these results, recommendations for the different configurations could be made. Afterwards, a final set of tests were conducted on a varying number of maximum fitness function evaluations to test different final configurations. In addition, a number of low evaluation tests were conducted to show the effectiveness of the DE algorithm when compared to backpropagation, and these tests outperformed backpropagation in 2 out of the 3 problems.

# Conclusions

## Achievements

The original project objective stated in the project brief is as followed:

* Creation of the basic Differential evolution algorithm in MATLAB, with the testing of the algorithm on CEC unconstrained optimization benchmarks to get a baseline reading for the algorithm.
* The adaptation of the Differential evolution algorithm with improvements proposed by researchers over the years such as custom fitness functions, crossover functions, and mutation functions. With it being tested against the same CEC benchmarks to evaluate the improvements.
* Creation of an artificial neural network in MATLAB to act as a surrogate model for the thermal modelling of a DC-DC converter, for the DE algorithm to optimize the weights of said network.
* The adaptation of the ANN to have more layers and for the DE to optimize the topology of the network
* Evaluate the DE algorithm with the different improvements on the ANN to determine the best parameters and operators for the algorithm.
* If time allows it, the creation of a GUI for the developed system to allow for easier use.

The main difference between the original objective and the achieved ones was the removal of the DC-DC converter thermal modelling. Originally it was intended that an artificial neural network would be created for the thermal modelling of a DC-DC power converter and have the differential evolution algorithm to optimize that neural network. The use of the thermal modelling was simply to be one an optimization problem to test the DE algorithm on neural networks and so it wasn’t necessary if alternative optimization problems were used in its place. The other minor difference was the order in which the objectives were accomplished. With that, the following is the achieved objectives all completed in MATLAB:

* Creation of the basic Differential Evolution Algorithm capable of optimizing the problems from the CEC 2005 benchmarks and Neural Networks weights, biases, number of neurons in a hidden layer, and the number of hidden layers.
* The analysis of the control parameters scalar F and crossover rate CR, determining the impact and the effects of different control parameters have on both normal optimization and neural network optimization. This includes the use of adaptive F and CR functions as well as the self-optimization of F and CR.
* The analysis of different mutation strategies, determining the impact and the effects of different mutation strategies have on both normal optimization and neural network optimization. This includes the use of an adaptive mutation strategy function with the use of different mutation strategy pools.
* The analysis of the effects of using different population sizes, especially when the maximum number of evaluations is capped. This includes the use of an adaptive population size function.
* The analysis of the effects of using a different method of handling boundary constraints.
* The analysis of a few different methods to initialising the population, as well as using these methods to perform generation-skipping.
* The analysis of a restart mechanism to help avoid local minima.

## Future Work

While this project covered many aspects of the differential evolution algorithm, these aspects were analysed individually to determine their effects on the algorithm. However, these aspects affect the others and so the analysis on how they affect each other could be conducted. One such example is the effects of using different control parameters when different mutation strategies are used. With that in mind, all aspects analysed used work from other researchers such as adaptive functions, and while this project covered a number of them, there is, in fact, many more and so more adaptive functions could be analysed to determine the more successful one. While the implementation of self-optimizing control parameters fell short, the concept is promising and with some further work, it could potentially rival the adaptive functions created.

While the differential evolution algorithm can be applied to optimize simple neural networks without much adjustment, further investigation could be conducted into ways to adjust the algorithm to work more effectively with neural networks, although this might render the algorithm only capable of working with neural networks and no longer being about to do normal mathematical optimization. One such example would be the inclusion of validation data in the selection process, a paper by Piotrowski, introduces two ways that this can be done. This method would allow the algorithm to create a neural network for the data without overfitting it to the training data, much like normal neural network training. In addition, the DE algorithm was performed a simple multi-layer perceptron to determine which aspects affect weight optimization for that type of network, however, there are more complex types of networks, such as recurrent or convolutional, and so further work could be conducted to allow the DE algorithm to perform on them.

Lastly, while the main premise of the paper and project was to be a guide to help others create a similar custom DE algorithm for a problem. The algorithm created could be altered to allow someone to simply input the options they desire (such as adaptive population size) so that they can use the algorithm coded and therefore removing the need for programming. This can be extended by the creation of a graphical interface to make it easier to use, and well as give more visual feedback from the algorithm.

# Student Reflections

When this project started, I possessed knowledge of evolutionary computing but not specifically the differential evolution algorithm. During this project, I had to learn the basics of the algorithm, how it worked and learn any extra work researchers have conducted on the algorithm over the years. While conducting the research necessary for this project, I was able to grasp an understanding on the concept of differential evolution which in turn helped with the research, allowing me to determine whether a piece of research could be useful for the project.

I started this project with an already existing proficiency in MATLAB programming both for mathematical projects and data science projects, which helped me to take the concepts from the research and implement them in MATLAB to be tested. It also helped me to adapt pseudocode from these research papers so that they could be pieced together as more papers are independent of each other.

While I did possess this proficiency in MATLAB programming, I hadn’t done much work inefficiency programming in said language. As stated earlier in the report, the first version of the code was inefficient in terms of performance speed, it resulted in tests taking several times longer then they needed to, and while I had done some work in making code more efficient in other programming languages, I had not done anything of the sort in MATLAB. This resulted in the need to learn tricks and tips to help reduce the number of operations performed and the time it takes to perform them, this mainly entailed removing for loops in exchange for matrix manipulation of which MATLAB excels at but the other programming languages I’ve used have not. Once the problem of inefficiency was resolved, which resulted in a newly structured code, any form of tests and feature analysis was able to be conducted within a timely fashion, which was not previously possible. This stage was necessary and in hindsight should have been conducted early then it was to save time when conducting the tests.

This problem was extended when it came to neural network optimization. Originally when a solution was to be tested it was inputted into a neural network created by the MATLAB Neural Net toolbox, while this proved simply it was also slow. A simple forward pass function had to be created to replace the neural net toolbox function that was used, while this task did not prove to be quick and straight forward, it did take a while to make it work for when the topology of each solution varies. After all of this was done, the speed of the algorithm was quick enough to conduct any test or analysis.

Aside from the code inefficiency, there only consisted of the main problems. These problems mainly consisted of coding errors when converting pseudocode to MATLAB, mainly caused by either my misunderstanding of the pseudocode or lack of explanation. However, these problems only caused minor delays, which mainly consisted of having to repeat some benchmarks. Other than that, these problems didn’t cause many delays and resolving them were able to be done quickly.

When I started implementing the algorithm, I focused more on creating it and attempting to add as much as I could to the algorithm to improve it. However, with help from my supervisor, I soon learnt the importance of questioning certain decision instead of simply accepting them, this helped when I started conducting my analysis of the different aspects.

References

Das, S. &. (2011). Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, 4 - 31.

Das, S. &. (2016). Recent Advances in Differential Evolution - An Updated Survey. *Swarm and Evolutionary Computation*.

Kreischer, V. &. (2017). Evaluation of Bound Constraints Handling Methods in Differential Evolution using the CEC2017 Benchmark.

S. Rahnamayan, H. R. (2008). "Opposition-Based Differential Evolution". *IEEE Transactions on Evolutionary Computation vol. 12*, pp. 64-79.

Sanderson, J. Z. (2007). "JADE: Self-adaptive differential evolution with fast and reliable convergence performance". *2007 IEEE Congress on Evolutionary Computation*, pp. 2251-2258.

Suganthan, A. K. (2005). "Self-adaptive differential evolution algorithm for numerical optimization". *IEEE Congress on Evolutionary Computation*, pp. 1785-1791.

Wagdy, A. (2014). RDEL: Restart Differential Evolution algorithm with Local Search Mutation for global numerical optimization. *Egyptian Informatics Journal*, 175-188.

Zhu, W. &.-a. (2013). Adaptive population tuning scheme for differential evolution. *Information Sciences 223*, 164–191.

Appendix A – Control Parameter Analysis Results







Appendix B – Project Brief and Meeting Records

# MASTER PROJECT BRIEF FORM 2018-19-20

1. Your details:

|  |
| --- |
| Full Name: Alexander Moses  Student ID: 6277932  E-mail: MosesA4@coventry.ac.uk  Module Code: M08CDE  Course of Study: Data Science and Computational Intelligence  Project Supervisor: Mauro S. Innocente |

2. Project title (provisional) [Meaningful, relevant and concise]

|  |
| --- |
| Differential Evolutionary Algorithm for Optimisation and Neuroevolution |

3. Outline (synopsis) of your project.[ What are the aim and objectives of the project?]

|  |
| --- |
| Evolutionary Algorithms (EAs) are methods inspired by biological evolution, mimicking the process of natural selection for optimization purposes. Differential Evolution (DE) is one of the more successful EAs, due to its nature to optimize nondifferentiable problems and its ability to escape poor local optima.  Artificial Neural Networks (ANN) is a system inspired by the biological neural network found in the human brain, an interconnection of neurons to allow the ability to learn. While ANNs come in many different forms to be applied to a wide range of applications, for this project the interest is with a feedforward network for the use as a surrogate data-driven model.  One important aspect of using ANNs is the training of its coefficients to match known data and therefore perform future predictions. Traditional training algorithms are gradient based, which while efficient, are prone to converge towards local optima instead of global. When coupled with an EA (in this case DE), thecoefficients of the network can be optimised to a global optimum. The EA can also be used to optimise the topology of the network to further improve the network.  This project aims to implement a simple DE algorithm to train and optimise ANNs, for which some prior optimisation benchmarking will be required. MATLAB’s ANN toolbox will also be used for reference. |

4. Intended user or group of users and their requirements. [ a) Who is the intended user or group of users? b) Why you think there is need for this project? c) What are the needs of the intended user that your product should satisfy?]

|  |
| --- |
| The differential evolution algorithm is a global search optimisation algorithm that can be used to optimize most problems, while the algorithm can be used to help optimize problems in most fields (engineering, medicine etc) the most likely people to use it is data scientists tasked with optimizing problems for people in those fields. The same can be said about the use of the DE in neuroevolution. With this in mind, the findings in the project need to allow others to easily implement the system themselves and to be able to reproduce the results.  The thermal modelling of a DC-DC converter in real time allows the project to be targeted to electrical engineers and automotive engineers. The thermal modelling of the converter needs to be in real time with a high degree of accuracy whilst maintaining a low computational cost. |

5. Systems requirements and project deliverables. [ a) What are the characteristics/properties that the final product should possess? b) What are the process stages and the corresponding deliverables that will enable you to create the final product?]

|  |
| --- |
| An implementation of the differential evolution algorithm with adaptations from the original algorithm, one capable of performing on several CEC unconstrainted optimization benchmarks (such as CEC 2005) to test the capability of the algorithm. Once developed and tested, the algorithm is to be adapted to optimize aspects of an artificial neural network, such as the weights and or topology while it acts as a surrogate model for the thermal modelling of a DC-DC converter.   * Creation of the basic Differential evolution algorithm in MATLAB, with the testing of the algorithm on CEC unconstrainted optimization benchmarks to get a baseline reading for the algorithm. * The adaptation of the Differential evolution algorithm with improvements proposed by researchers over the years such as custom fitness functions, crossover functions, and mutation functions. With it being tested against the same CEC benchmarks to evaluate the improvements. * Creation of an artificial neural network in MATLAB to act as a surrogate model for the thermal modelling of a DC-DC converter, for the DE algorithm to optimize the weights of said network. * The adaptation of the ANN to have more layers and for the DE to optimize the topology of the network * Evaluate the DE algorithm with the different improvements on the ANN to determine the best parameters and operators for the algorithm. * If time allows it, the creation of a GUI for the developed system to allow for easier use. * A technical report of both the DE algorithm and the ANN, with evidence of accurate predictions against the original thermal model, and with a basic user manual. |

6. Research [ a) How will you investigate/identify in detail the needs of the specified user in (3) b) How will you investigate the background of the project?]

|  |
| --- |
| With access to the paper by Storn and Price which introduced the differential evolution algorithm, a basic understanding of the base algorithm can be obtained. This will allow for an easier understanding of the other papers published by researchers over the years with their improvements, to help identify which aspects could be useful and warrant further investigation.  Many attempts for Neuroevolution has been attempted over the years, covering different evolutionary algorithms and evolving different aspects of the network. With the large range and variety of papers, a basic understanding on how to use DE to optimize the ANN can be achieved easily.  While the ANN is to act only as a surrogate for the thermal modelling of the DC-DC converter, a basic knowledge of thermal modelling and of DC-DC power converters would allow for easier implementation of the surrogate model. |

7. Evaluation. [ a) What makes a product successful? b) How will you demonstrate that your product fulfils the needs of the user in (3)? c) How will you evaluate the product?]

|  |
| --- |
| The CEC benchmarks are designed to test the effectiveness of evolutionary based optimization algorithms, by testing the algorithm against several of these benchmarks, a baseline of the effectiveness of the algorithm can be obtained. When a potential improvement is implemented, it can be tested against the same benchmarks and compared to the original results, which in turn allows for the evaluation of the effectiveness of said potential improvement.  When the neural network is developed, it will be run with a basic topology to get a baseline reading of the network, with its accuracy score and how long it takes to train. As the implementation of the DE on the NN will allow for higher accuracy (if it optimizes the topology) and quicker training time (if it optimizes the weights). So, the different configurations of the DE testing previously can be tested with the NN and compared to the original results and evaluated at whether it improves the accuracy and or the training time. |

8. Development skills. [ a) What information and resources do you need to complete the project successfully? b) Which of these do you need to acquire yourself? ]

|  |
| --- |
| * Skills   + Ability to programme algorithms in MATLAB   + Basic knowledge of Evolutionary Algorithms, Artificial Neural Networks and Optimisation.   + GUI development skills if GUI development is pursued   + Knowledge of mathematical modelling   + Knowledge of the differential evolution algorithm * Resources   + Computer with the computational needs to run the algorithm in a timely fashion to test the different variations of the algorithm and to evaluate them.   + Access to researchers’ materials on differential evolution algorithm and other related text. |

9.Skill acquisition. [How do you intend to gain the skills, information and resources specified in (7)?]

|  |
| --- |
| * Skills   + I am already proficient in programming in the MATLAB programming language.   + I already have a basic knowledge of Evolutionary Algorithms, Artificial Neural Networks and Optimisation.   + I am experienced in GUI development outside of MATLAB. With said experience and the already existing proficiency in MATLAB, learning to develop a GUI shouldn’t take a large amount of time. In addition, MATLAB has a built-in app developer, which should speed up the process.   + I already have knowledge of mathematical modelling with a background in both computer science and electrical engineering.   + While I have no past experience in differential evolution algorithm, with experience in the area of evolutionary algorithms, access to researchers’ papers on the topic should suffice. * Resources   + I already possess a computer with the means of running the algorithm, both personally and at the university   + Access to the researchers’ materials through the library and the university |

10. Estimate the number of hours you are planning to spend on each of the following tasks:

|  |  |
| --- | --- |
| Background research | 75 |
| Learning new skills | 10 |
| Base DE Algorithm development + CEC Testing | 50 |
| DE Algorithm Improvement + CEC Testing | 125 |
| ANN development | 50 |
| DE and ANN Combination + Evaluation | 115 |
| GUI Development (If pursued) | 25 |
| Final Report Preparation | 150 |
| **Total Number of Hours** | **600** |

**Important note:** Ethical approval must be obtained for ALL research projects and BEFORE any data are collected.

All students MUST get ethics approval on projects. Any project which has not received approval will be failed.

You MUST complete the Ethics Online Procedure. You CANNOT start with data collection until your ethics application has been approved. Failure to comply will result in you not continuing with your project hence automatically fail your project

See CU Ethics About at < <https://ethics.coventry.ac.uk/about/default.aspx>>

The grade for the Project Brief may only be awarded if the Ethics Online Procedure has been completed for approval by supervisor.

This form must be submitted electronically on your Moodle project web ***before*** 18:00 on the due date.

**I will complete my Ethics application within one (1) week from the Project Brief Deadline: NO**

Signature:  Date: 31/05/2019



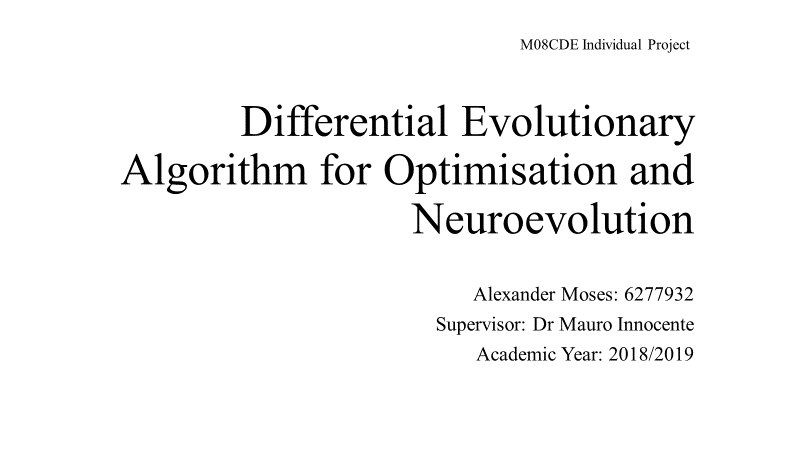


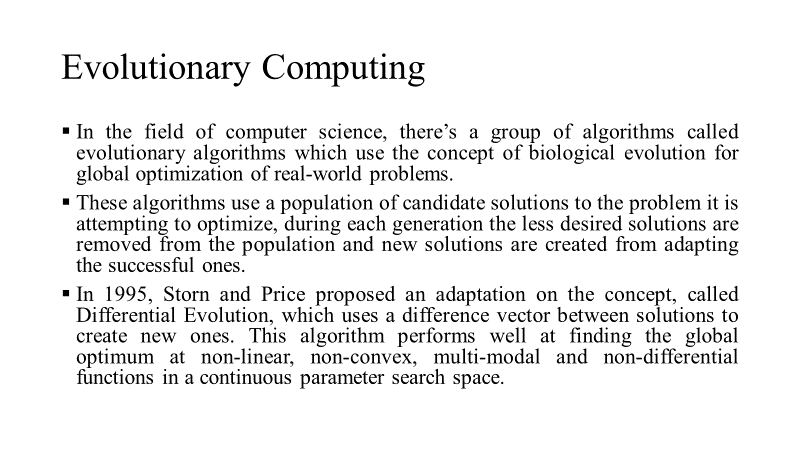


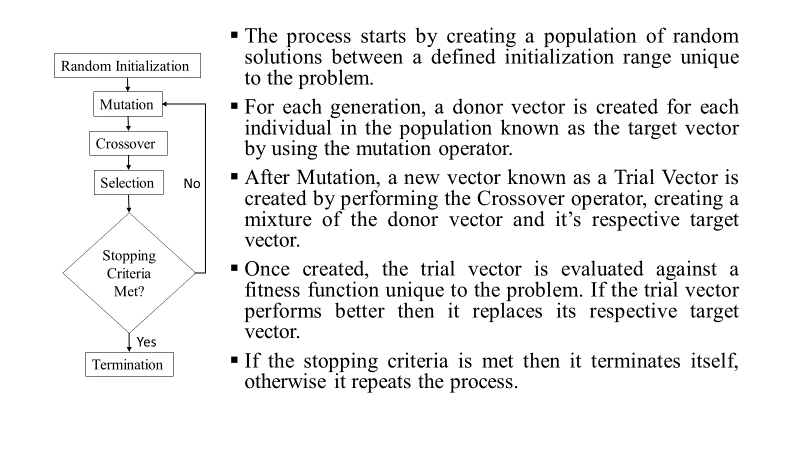


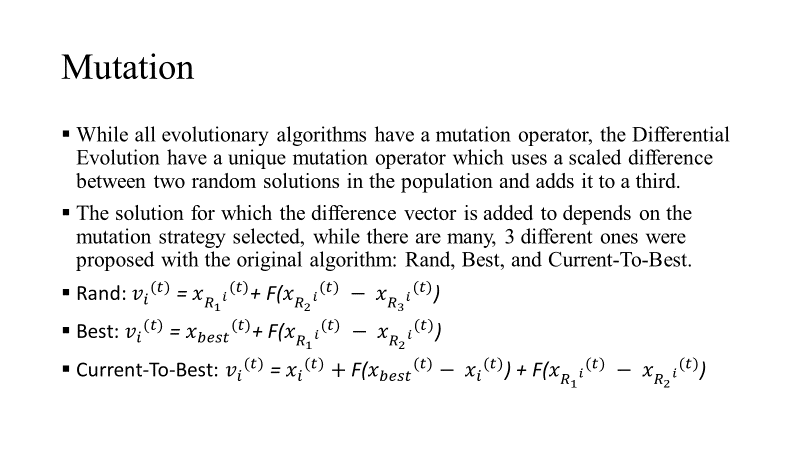


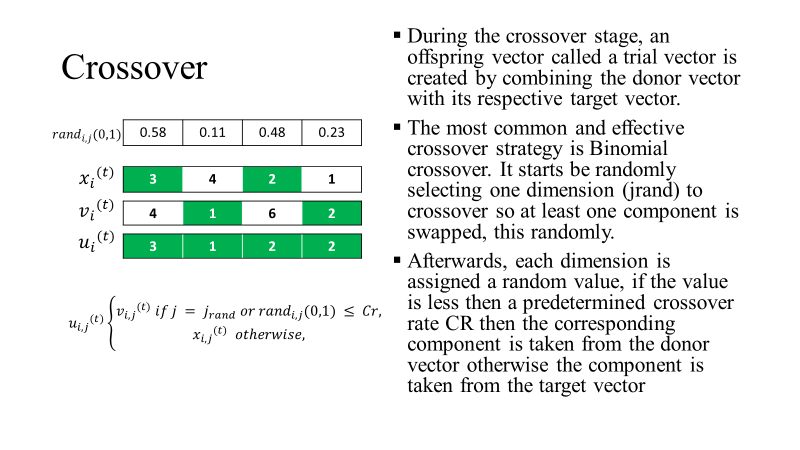
Appendix C – Project Presentation

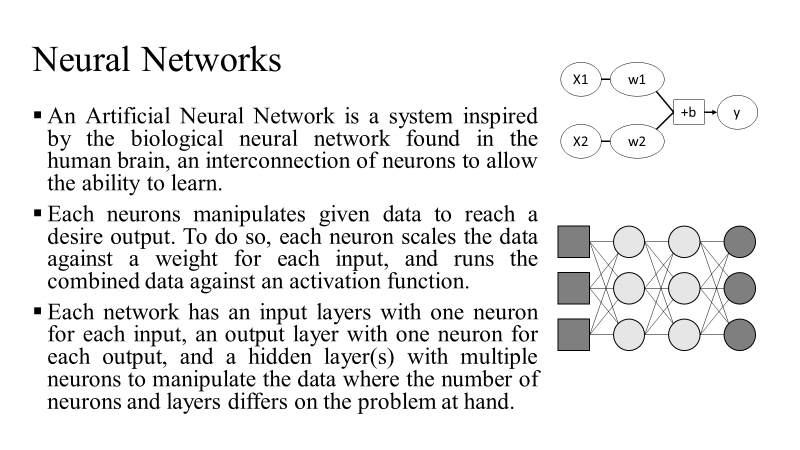


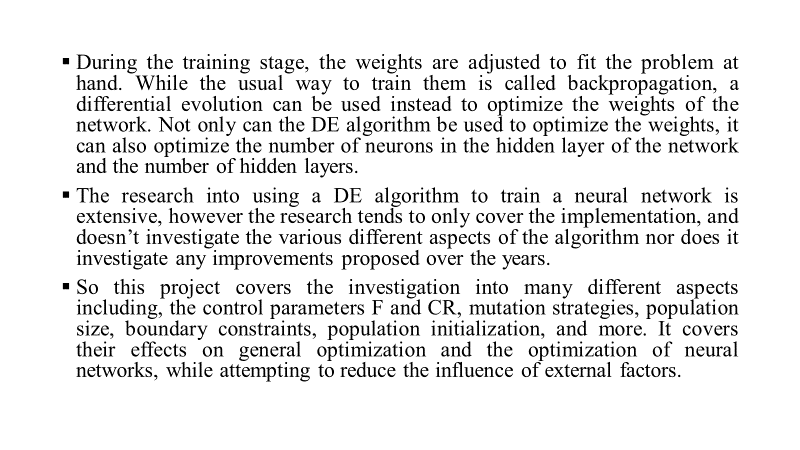


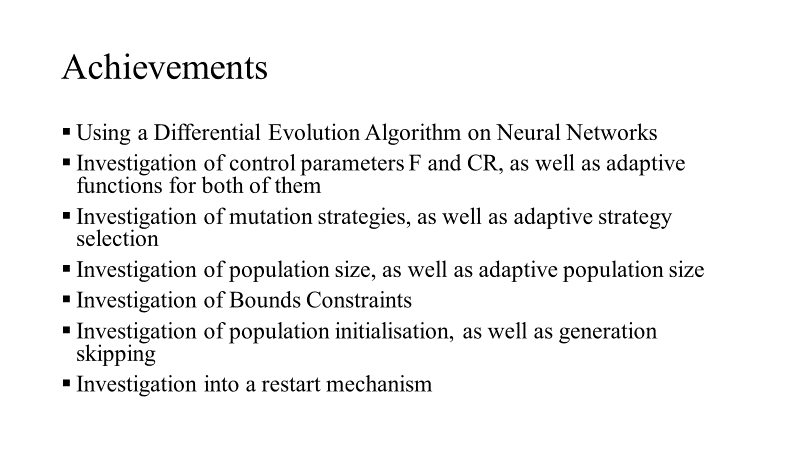


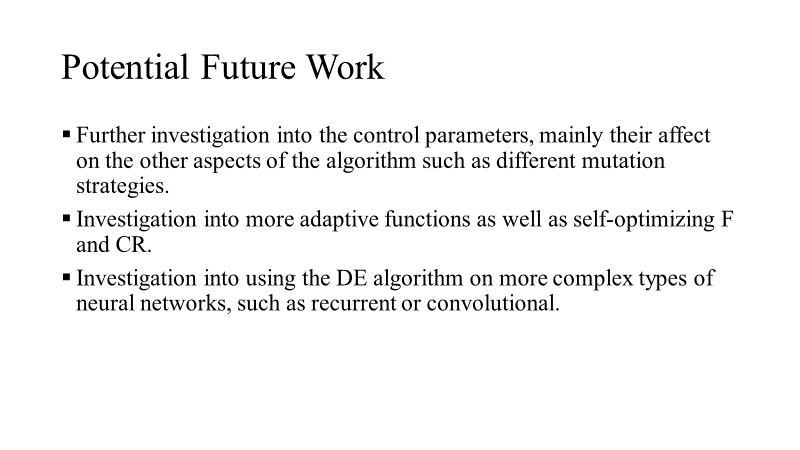


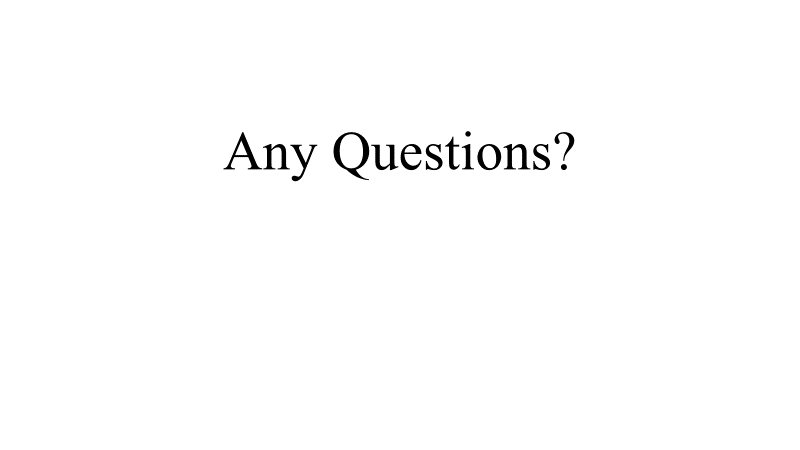


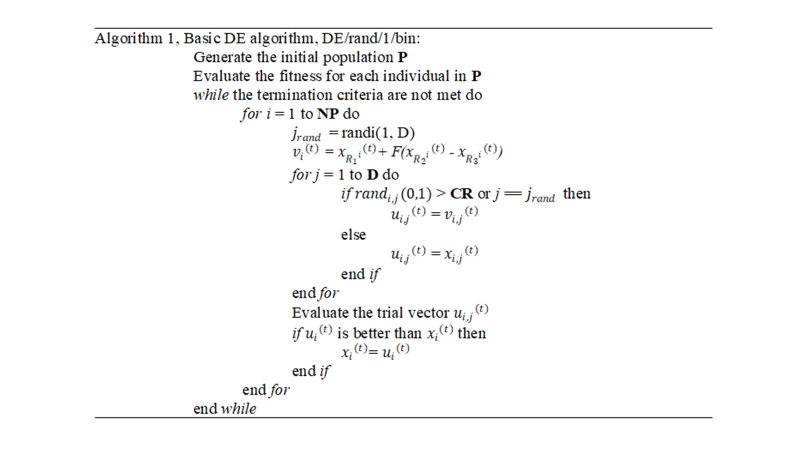


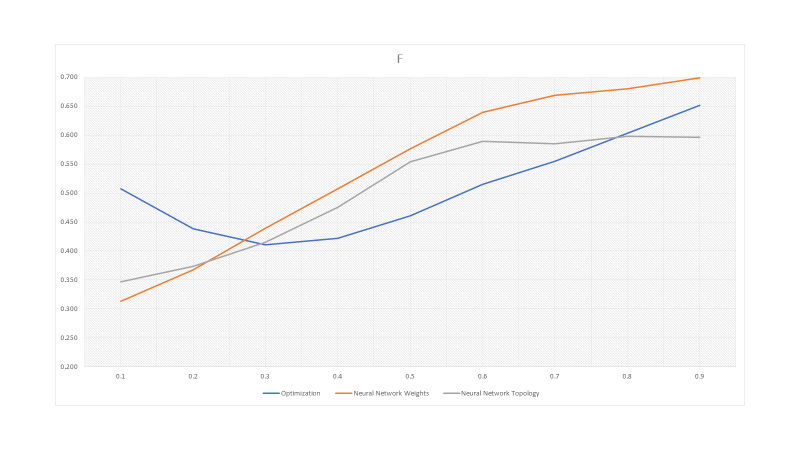












Appendix D – Certificate of Ethics Approval

