Title: Advances in evolutionary neural networks with applications in energy systems and the environment

Author(s): Mason, Karl

Publication Date: 2018-08-29

Publisher: NUI Galway

Item record: http://hdl.handle.net/10379/10017

Some rights reserved. For more information, please see the item record link above.
Advances in Evolutionary Neural Networks with Applications in Energy Systems and the Environment

Karl Mason

Submitted for the degree of Doctor of Philosophy

August 2018
Abstract

Evolutionary neural networks combine two of the most powerful areas of computing, evolutionary algorithms and neural networks. There are a number of benefits of using evolutionary algorithms to train a network over traditional methods. These include: no need for target outputs and error gradients, applicable to both supervised learning and reinforcement learning problems, and robustness to noise and local optima.

This thesis presents both a number of novel applications of evolutionary neural networks to energy systems and also the development of algorithms for evolving neural networks for these problems. The applications of evolutionary neural networks to energy systems and the environment include: Watershed Management, Dynamic Economic Emission Dispatch, forecasting Ireland’s power demand, wind power generation and carbon dioxide levels, and also forecasting CPU utilization in data centers. Each of these problems are of critical importance. The work described in this thesis demonstrates how evolutionary neural networks can have a positive impact on these problems.

Aside from applications to real world problems, this thesis also makes fundamental contributions to evolutionary neural networks. A meta optimisation analysis of the largest collection of Particle Swarm Optimisation velocity update equations is conducted, revealing that many uncommon parameter settings can enhance performance. A Neuro Differential Evolution algorithm is proposed that combines Genetic Algorithms with Differential Evolution to evolve the size, topology and weights of a neural network. This thesis also presents a Multi-Objective Neural Network trained with Differential Evolution that can successfully produce Pareto fronts for dynamic multi-objective optimisation problems.
Acknowledgements

I would like to thank the following people for their support and encouragement throughout my PhD:

- My supervisors Enda Howley and Jim Duggan. I greatly appreciate all the time you have given me to help me carry out my PhD research. Your advice, feedback and support has been instrumental to the completion of this thesis.

- I would like to also thank all of the researchers in Room 307 for making the PhD process much more enjoyable. All of our entertaining discussions while procrastinating from doing our research will be fondly remembered.

- Next, I would like to thank Niamh for being there for me and supporting me. Our trips and adventures were often needed to take my mind away from research. These were some of the highlights of my PhD and I am excited for them to continue into the future.

- Finally, I would like to thank my family for their love and unwavering support for me and my friends which I am deeply grateful to have.
Contents

Abstract i

Acknowledgements iii

1 Introduction 1

1.1 Motivation .......................................................... 1

1.2 Hypothesis ............................................................ 2

1.3 Contributions ......................................................... 2

1.4 Statement of Originality ............................................. 3

1.5 Publications .......................................................... 3

1.5.1 Journal Papers .................................................... 3

1.5.2 Conference and Workshop Proceedings ......................... 4

1.6 Thesis Overview ...................................................... 4

1.7 Background Literature ................................................. 6

1.7.1 Computing and Artificial Intelligence ......................... 7

1.7.2 Evolutionary Algorithms ...................................... 8

1.7.3 Swarm Intelligence Algorithms ............................... 13

1.7.4 Neural Networks ................................................ 16

1.7.5 Evolutionary Neural Networks ............................... 18

1.7.6 Energy and The Environment ................................. 25
2 A Meta Optimisation Analysis of Particle Swarm Optimisation 31

2.1 Introduction .................................................. 31

2.1.1 Velocity Update Equations ................................. 32

2.1.2 Meta Optimisation & Parameter Selection .......................... 33

2.1.3 Watershed Management .................................. 34

2.1.4 Contribution & Structure .................................. 35

2.2 Particle Swarm Optimisation .................................. 35

2.2.1 Velocity Update Equation Variations .......................... 36

2.3 Meta Optimisation .............................................. 39

2.3.1 Application of Meta Optimisation ............................. 39

2.3.2 Problem Definition ......................................... 41

2.4 Machine Learning .............................................. 41

2.4.1 Artificial Neural Networks .................................. 42

2.4.2 Related Work ............................................... 44

2.4.3 Application to Watershed Management ......................... 44

2.5 Watershed Problem ............................................ 45

2.5.1 Boundary Definition ....................................... 47

2.5.2 Constraint Handling ....................................... 48

2.5.3 Fitness Function ............................................ 48

2.6 Experimental Results .......................................... 49

2.6.1 Constriction vs Inertia ...................................... 49

2.6.2 Avoidance Strategies ....................................... 50

2.6.3 Non-Standard Velocity Update Equations ....................... 50

2.6.4 Hybrid Velocity Update Equations ........................... 51

2.6.5 Overall Comparison ....................................... 53
2.6.6 Optimum PSO Parameters ................................. 55
2.6.7 Individual Problem Performance .......................... 59
2.6.8 Watershed Test Data ........................................ 60
2.7 Conclusion ....................................................... 63
2.7.1 Future Work .................................................. 64

3 A Combined Genetic Algorithm - Differential Evolution Strategy 65

3.1 Introduction ...................................................... 65
3.2 Background ....................................................... 67
3.2.1 Neuroevolution ............................................... 67
3.2.2 NEAT Algorithm .............................................. 68
3.2.3 ESP Algorithm ............................................... 69
3.2.4 Differential Evolution Algorithm ............................... 69
3.3 Neuro Differential Evolution (NDE) ......................... 70
3.3.1 Encoding ..................................................... 71
3.3.2 Speciation .................................................... 71
3.3.3 Network Growth ............................................. 72
3.3.4 NDE Algorithm ............................................. 74
3.4 Pole Balancing Experiment .................................... 75
3.4.1 Pole Balancing Problem .................................... 77
3.4.2 Double Pole Problem ..................................... 77
3.4.3 Non Markovian Variant .................................... 78
3.4.4 Fitness Function and Generalization ......................... 78
3.5 Watershed Management Experiment .......................... 79
3.5.1 Boundary Definition ...................................... 81
3.5.2 Constraint Handling ...................................... 82
3.5.3 Fitness Function ........................................ 83
3.6 Results ................................................... 83
  3.6.1 Pole Balancing Experiments ........................... 84
  3.6.2 Watershed Management Experiment .................... 86
  3.6.3 Summary ............................................. 87
3.7 Conclusion .............................................. 89
  3.7.1 Future Work .......................................... 90

4 A Multi-Objective Neural Network Trained with Differential Evolu-
tion .......................................................... 91
  4.1 Introduction ............................................. 91
  4.2 Background ............................................. 95
    4.2.1 Multi-Objective Optimisation ...................... 95
    4.2.2 Neural Networks .................................. 96
    4.2.3 Partially Connected Neural Networks .............. 97
    4.2.4 Differential Evolution ................................ 98
    4.2.5 Neural Network Topology and Weight Optimisation 99
  4.3 Dynamic Economic Emission Dispatch .................... 102
    4.3.1 Problem Definition .................................. 104
    4.3.2 Constraint Handling .................................. 106
  4.4 MONNDE Algorithm ..................................... 107
    4.4.1 Standard Fitness Function ......................... 111
    4.4.2 Proposed Fitness Function ......................... 111
    4.4.3 Parameter Selection .................................. 113
  4.5 Experimental Procedure .................................. 113
    4.5.1 Experiment 1 - Bench Marking MONNDE ............ 114
4.5.2 Experiment 2 - Proposed Fitness Function .......................... 114
4.5.3 Experiment 3 - Partially Connected Neural Networks .............. 115
4.5.4 Experiment 4 - Generator Failure ................................. 115
4.5.5 Experiment 5 - Scalability ........................................ 115
4.5.6 Experiment 6 - New Power Demands ............................... 116
4.6 Results ........................................................................... 116
4.6.1 Standard Fitness Function ............................................. 117
4.6.2 Fitness Function with Pareto Penalty ................................. 120
4.6.3 Partially Connected Neural Networks ................................. 122
4.6.4 Generator Failure ....................................................... 124
4.6.5 5 and 15 Generator Systems .......................................... 126
4.6.6 New Power Demands Evaluation .................................... 128
4.7 Conclusion ....................................................................... 133

5 Forecasting in Ireland’s Energy Sector .................................. 135
5.1 Introduction ...................................................................... 135
5.2 Energy Forecasting ......................................................... 137
5.2.1 Energy Demand Forecasting .......................................... 138
5.2.2 Wind Generation Forecasting ....................................... 139
5.2.3 Atmospheric Pollutant Forecasting ................................. 140
5.3 Neural Networks ............................................................ 141
5.3.1 Backpropagation ......................................................... 144
5.4 Swarm and Evolutionary Methods for Energy Forecasting ....... 145
5.4.1 Covariance Matrix Adaptation Evolutionary Strategy .......... 145
5.4.2 Particle Swarm Optimisation ........................................ 147
5.4.3 Differential Evolution .................................................. 149
5.5 Experimental Methods ........................................ 150
  5.5.1 Forecast Data ........................................ 150
  5.5.2 Network Parameter Selection .......................... 150
  5.5.3 Network Training ...................................... 151
  5.5.4 Comparative Forecasting Methods .................... 151
  5.5.5 Forecast Accuracy Evaluation ........................ 152
  5.5.6 Experiments Conducted ................................ 153

5.6 Results ..................................................... 154
  5.6.1 Convergence ........................................... 154
  5.6.2 Accuracy ............................................... 155
  5.6.3 Predictions ............................................ 157
  5.6.4 Multi-step Ahead Prediction .......................... 161
  5.6.5 Discussion ............................................. 162

5.7 Conclusion ................................................ 166
  5.7.1 Future Work ........................................... 167

6 Forecasting In The Cloud .................................... 168
  6.1 Introduction ............................................. 168
  6.2 Related Work ............................................ 171
    6.2.1 Forecasting in Cloud Computing .................... 171
    6.2.2 State of the Art .................................... 174
    6.2.3 Neural Networks .................................... 174
    6.2.4 Swarm and Evolutionary Algorithms ............... 176
  6.3 Experimental Setup ..................................... 180
    6.3.1 Network Parameter Selection ....................... 180
    6.3.2 Network Training ................................... 181
7 Conclusion 194

7.1 Summary of Thesis Achievements 195

7.2 State of the Art 197

7.3 Limitations 197

7.3.1 Computational Cost of Meta Optimisation 197

7.3.2 Choice of Meta Optimisation Algorithm 197

7.3.3 Neuro Differential Evolution and Large Networks 198

7.3.4 MONNDE and Large Networks 198

7.3.5 Diversity of Pareto fronts produced by MONNDE 198

7.3.6 Univariate Forecasting of Ireland’s Energy Sector 199

7.3.7 Implementing Resource Predictions in Data Center Management 199

7.4 Future Work 200

7.4.1 PSO Parameters 200

7.4.2 Scaling Up NDE 200
List of Tables

2.1 Velocity Update Equations ......................................... 38
2.2 Benchmark Maximisation Functions ................................. 42
2.3 Watershed Constants .................................................. 46
2.4 Modified Velocity Update Equations ............................... 52
2.5 Optimum Parameter Settings & Fitness Results .................. 56
2.6 Individual Problem Average Fitness & Standard Deviation. This table presents the average fitness and fitness standard deviation of each velocity update equation for all 8 problem spaces. .................. 61
2.7 Test Data Neural Network Results .................................. 62
3.1 Watershed Constants .................................................. 80
3.2 Average Evaluations Taken for 4 Variants of Non Markovian Double Pole Balancing Problem. This table displays the average evaluations taken and standard deviation over 10 runs for the proposed NDE, NEAT [347] and ESP [142] algorithms. .............................. 85
3.3 Number of Failed Runs. This table presents the number of failed runs (out of 10) for each algorithm on each experiment. A failed run on the cart pole problem refers to failing to pass the generalization test. ... 88
4.1 Locations of Hourly Pareto Fronts For 24 Hours (w = 0.5) - 10 Generator System .................................................. 118
4.2 Average 24 Hour Cost and Emissions (w = 0.5) - 10 Unit System ... 118
List of Figures

1.1 Thesis Structure .................................................... 5

1.2 Evolutionary Algorithm Blueprint ............................... 11


1.4 Neural Network ...................................................... 18


2.1 Meta Optimisation Process. The overlaying PSO meta optimiser is used to find the optimal parameters for the sub optimisation PSO. The parameters of each velocity update equation of the sub optimiser PSO will be optimised for 8 problems .............................. 41

2.2 Multi-Layer Artificial Neural Network ........................... 43

2.3 Watershed Illustration [394] ........................................ 47
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4</td>
<td>Convergence of Velocity Update Equations using Constriction &amp; Inertia. This graph illustrates the average cumulative fitness convergence of the standard constricted PSO, PSO with linear inertia and adaptive inertia PSO over all 8 problems. 49</td>
</tr>
<tr>
<td>2.5</td>
<td>Convergence of Avoidance Velocity Update Equations. This graph illustrates the average cumulative fitness convergence of the 6 PSO variants that utilize avoidance strategies over all 8 problems. 51</td>
</tr>
<tr>
<td>2.6</td>
<td>Convergence of Non-Standard Velocity Update Equations. This graph illustrates the average cumulative fitness convergence of the 6 non-standard PSO velocity equations over all 8 problems. 52</td>
</tr>
<tr>
<td>2.7</td>
<td>Convergence of Hybrid Velocity Update Equations. This graph illustrates the average cumulative fitness convergence of the 5 hybrid PSO velocity update equations over all 8 problems. 53</td>
</tr>
<tr>
<td>2.8</td>
<td>Convergence of Best 8 Velocity Update Equations. This graph illustrates the average cumulative fitness convergence of the 8 best performing PSO velocity update equations over all 8 problems. 55</td>
</tr>
<tr>
<td>3.1</td>
<td>Neuro Differential Evolution Encoding. A direct encoding scheme is implemented to map the genotype to the network phenotype. A 0 implies no connection while a 1 implies that a connection exists. 72</td>
</tr>
<tr>
<td>3.2</td>
<td>Neuro Differential Evolution Crossover. This image illustrates how crossover occurs between genes within the same species and how this relates to the network topologies. 73</td>
</tr>
<tr>
<td>3.3</td>
<td>Neuro Differential Evolution Species Mutation. When $\text{rand()} &lt; sm$, a new species is created. This new species consists of a network with an additional neuron. The weights and connectivity of this new neuron are initialized at random as illustrated above. 74</td>
</tr>
<tr>
<td>3.4</td>
<td>Snapshot of Double Pole Balancing Problem 3D Simulation [142]. The evolved neural network controller must continuously balance both poles by applying a force to the cart. Balancing two poles simultaneously without velocity information is a difficult task. 77</td>
</tr>
<tr>
<td>3.5</td>
<td>Watershed Illustration [394]. 82</td>
</tr>
</tbody>
</table>
3.6 Short Pole Length Increases vs Average Number of Evaluations Taken. This figure graphs the increase in the number of evaluations taken as the length of the short pole is iteratively increased. This illustrates the scalability of the proposed NDE, NEAT [347] and ESP [142] algorithms. 85

3.7 Short Pole Length Increases vs Number of Failed Runs. This figure graphs the number of runs that failed to find a solution that passes the generalization test against the length of the short pole. Neither NDE or ESP fail once in any of the experiments. This demonstrates the robustness of each algorithm. 86

3.8 Fitness Convergence for Watershed Management Problem. This graph displays the average fitness convergence of the NEAT, ESP and NDE algorithms for the watershed problem. Each algorithm was able to find a solution with no violation on every run. 87

4.1 Recurrent Neural Network. This figure depicts the structure of a fully connected recurrent neural network. 97

4.2 Neural Network Encoding. The network connections are encoded into a string of bits that the genetic algorithm can optimise. A 0 (red) implies no connection while a 1 (black) implies that a connection exists. 101

4.3 Crossover. This image illustrates how crossover occurs between chromosomes. 102

4.4 MONNDE Algorithm Flowchart 110

4.5 Average Pareto Front Over 24 Hours. 117

4.6 Hour 12 Pareto Front - 10 Unit System 121

4.7 Hour 14 Pareto Front - 10 Unit System 121

4.8 Convergence of Partially vs Fully Connected Neural Network- 10 Unit System 122

4.9 Best (a) and Average (b) 24 Hour Pareto Front PCNN vs FCNN - 10 Unit System 123

4.10 Fitness Convergence with Power Generator Failure - 10 Unit System 125
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.11</td>
<td>Best Pareto Front Over 24 Hours - 5 and 15 Units.</td>
<td>128</td>
</tr>
<tr>
<td>4.12</td>
<td>Pareto Front Over 24 Hours- Test Power Demand.</td>
<td>130</td>
</tr>
<tr>
<td>5.1</td>
<td>Recurrent Neural Network [240]. This figure depicts a fully connected recurrent neural network. Neurons are connected by weighted connections (known as synapses). These connections pass signals between neurons. The recurrent connections can be seen in the hidden layer of neurons. These recurrent connections give the network memory from the previous forward pass.</td>
<td>142</td>
</tr>
<tr>
<td>5.2</td>
<td>Convergence of Each Neural Network Training Algorithm. This figure illustrates the convergence of each evolutionary neural network training algorithm as each network learns to predict the training data. Graphs a, b and c depict the wind power, power demand and CO$_2$ levels respectively.</td>
<td>155</td>
</tr>
<tr>
<td>5.3</td>
<td>Forecast Accuracy Spread of Each Algorithm. This figure illustrates the spread in accuracy of each forecasting algorithm. Graphs a, b and c depict the wind power, power demand and CO$_2$ levels respectively.</td>
<td>158</td>
</tr>
<tr>
<td>5.4</td>
<td>Wind Power Generation Predictions for Training and Test Data. This figure illustrates the predicted wind power generated at each time step using the evolved neural networks. Graph (a) illustrates the predictions for the training data while graph (b) presents the test data predictions.</td>
<td>160</td>
</tr>
<tr>
<td>5.5</td>
<td>Power Demand Predictions for Training and Test Data. This figure illustrates the predicted power demand at each time step using the evolved neural networks. Graph (a) illustrates the predictions for the training data while graph (b) presents the test data predictions.</td>
<td>160</td>
</tr>
<tr>
<td>5.6</td>
<td>CO$_2$ Intensity Predictions for Training and Test Data. This figure illustrates the predicted CO$_2$ level at each time step using the evolved neural networks. Graph (a) illustrates the predictions for the training data while graph (b) presents the test data predictions.</td>
<td>161</td>
</tr>
<tr>
<td>5.7</td>
<td>MAE of CMA-ES Neural Network When Forecasting Further Into the Future. This figure illustrates how the forecasting accuracy of the neural network degrades as it predicts further into the future for each problem.</td>
<td>162</td>
</tr>
</tbody>
</table>
6.1 Recurrent Neural Network [240]. This figure illustrates a fully connected recurrent neural network. Neurons are connected by weighted synapses (or connections) that pass signals between neurons. The recurrent connections can be seen in the hidden layer of neurons. This gives the system memory. Image taken with permission from Mason et al. [240] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 176

6.2 Convergence of Each Neural Network Training Algorithm. This figure illustrates the convergence of each neural network training algorithm as each network learns to predict the training data. . . . . . . . . . . . 184

6.3 Host Utilization Predictions for First 500 Steps of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for the first 500 training data time steps. The predictions for the remaining training data can be found in the appendix. . . . . . . 185

6.4 Forecasting Accuracy Spread. These figures depict box plots for each of the algorithms implemented. These graphs highlight the statistical differences in the performance of every algorithm. . . . . . . . . . . . . 186

6.5 Host Utilization Predictions for Test Data. This figure illustrates the predicted host utilization of each evolved neural network on unseen test data. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 187

6.6 Host Utilization Predictions for First 500 Steps of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for the first 500 training data time steps. The predictions for the remaining training data can be found in the appendix. . . . . . . 189

1 Wind Power Generation Error Per Time Step. This figure illustrates the wind power generated along with the error of the Covariance Matrix Adaptation - Evolutionary Strategy trained neural network at each time step. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 209

2 Power Demand Error Per Time Step. This figure illustrates the power demand along with the error of the Covariance Matrix Adaptation - Evolutionary Strategy trained neural network at each time step. . . 209
3 Carbon Dioxide Level Error Per Time Step. This figure illustrates the carbon dioxide level along with the error of the Covariance Matrix Adaptation - Evolutionary Strategy trained neural network at each time step. .......................................................... 210

4 Host Utilization Predictions for Time Steps 500 - 1000 of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for time steps 500 - 1000 of training data. ....... 211

5 Host Utilization Predictions for Time Steps 1000 - 1500 of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for time steps 1000 - 1500 of training data. .. 211

6 Host Utilization Predictions for Time Steps 1500 - 2000 of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for time steps 1500 - 2000 of training data. . 212

7 Host Utilization Predictions for Time Steps 2000 - 2300 of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for time steps 2000 - 2300 of training data. . 212
Chapter 1

Introduction

1.1 Motivation

Machine learning is a field of study within computer science that is concerned with creating computer programs that can automatically learn to complete a task without being explicitly programmed how to do so. In order to achieve this ambitious goal, researchers in computing have looked to the natural world for inspiration. This has led to the development of neural networks. Neural networks are computational models that are inspired by the biological brain. Neural networks are capable of approximating functions that allow them to successfully complete tasks by producing the correct output for a given input [47].

The function that the network can approximate is determined by the parameters of the network. In order to develop neural networks capable of producing the correct outputs, the optimum parameters of the network must first be selected. In order to achieve this, researchers have once again turned to nature. All living organisms have evolved by Darwinian Evolution to successfully adapt to their environment. The field of evolutionary algorithms mimics this evolutionary process and applies it to optimisation and machine learning problems. Algorithms such as Genetic Algorithms mimic this evolutionary process by replicating evolutionary processes such as selection, crossover and mutation [169].

By applying evolutionary algorithms to the neural network parameters, it is possible to evolve a neural network capable of completing machine learning tasks. This process of evolving neural networks is often referred to as neuroevolution. Many swarm
intelligence optimisation algorithms have also been developed by taking inspiration from the collective behaviour exhibited by a decentralized group of individuals, such as flocks of birds [205], or colonies of ants [99].

There are numerous successful applications of evolutionary neural networks to many machine learning tasks, from control to forecasting. These evolutionary neural networks have proven to provide great benefits to a large number of fields from physics [1] to medicine [232]. With environmental change by global warming listed as one of the greatest threats to human existence [357] and an increasing need for energy worldwide [16], it is hoped that evolutionary neural networks can provide further benefits to problems relating to energy systems and the environment.

1.2 Hypothesis

The primary aim of this thesis is to demonstrate that evolutionary neural networks perform competitively with state of the art methods when applied to control and forecasting problems in the areas of energy systems and the environment.

1.3 Contributions

The most significant contributions of the research presented in this these can be summarized as follows.

1. The application of a neural network trained with Particle Swarm Optimisation (PSO) variants to the environmental task of watershed management. It is shown that the trained neural network can successfully complete this task. A meta optimisation analysis is also conducted of the most prominent PSO velocity update equations to explore their performance [245] (Chapter 2).

2. A combined evolutionary strategy, using a Genetic Algorithm (GA) and Differential Evolution (DE), is proposed to optimise the topology and weights of a neural network. This approach is validated by comparing it to state of the art neuroevolution algorithms on the reinforcement learning task, the double pole balancing problem, and the watershed management problem [247, 242] (Chapter 3).
3. A method for evolving neural networks for dynamic multi-objective optimisation problems is proposed. This Multi-Objective Neural Network trained with Differential Evolution (MONNDE) algorithm is applied to the task of Dynamic Economic Emissions Dispatch (DEED) where it performs competitively with state of the art multi-objective optimisation algorithms. A fitness function with Pareto penalty is proposed that places evolutionary pressure on the MONNDE algorithm to produce viable Pareto fronts [240, 246] (Chapter 4).

4. Evolutionary neural networks are applied to the task of forecasting in Ireland’s energy sector. The evolved network performs better than many state of the art forecasting approaches to predict Ireland’s power demand, wind power generation and carbon dioxide emissions in the short term. This is vital for unit commitment of power generators and policy makers regarding clean energy production [243] (Chapter 5).

5. Evolutionary neural networks are also applied to the task of predicting host CPU utilization in data centers. Evolutionary neural networks again outperform state of the art forecasting approaches when predicting CPU utilization. This is crucial for managing host migration and energy consumption in data centers [248, 111] (Chapter 6).

1.4 Statement of Originality

I declare that the research described in this thesis is original work, which I undertook at the National University of Ireland, Galway during the years 2015 - 2018. This work has not previously been presented for an award at this or any other university.

1.5 Publications

Many parts of this thesis have been published in journals and conference proceedings. For each of the publications listed below, the first author is the primary author.

1.5.1 Journal Papers

1. **Mason, K.**, Duggan, J. and Howley, E., 2018. A meta optimisation analysis of particle swarm optimisation velocity update equations for watershed manage-


### 1.5.2 Conference and Workshop Proceedings


### 1.6 Thesis Overview

The diagram in Figure 1.1 outlines the broad structure of this thesis. This diagram outlines which algorithms, problems and problem types are examined in each chapter.
1.6. Thesis Overview

The acronyms in this diagram are: PSO - Particle Swarm Optimisation, DE - Differential Evolution, CMA-ES - Covariance Matrix Adaptation Evolutionary Strategy and NN - Neural Network.

A detailed outline of the structure of this thesis is as follows:

- Chapter 2 will explore the environmental problem of watershed management. A number of PSO variants will be applied to train a neural network to complete this task. A meta optimisation analysis will be conducted to evaluate the parameter settings of each velocity update equation.

- The idea of neural network topology and weight optimisation will be investigated in Chapter 3. A combined evolutionary strategy “Neuro Differential Evolution”...
(NDE) is proposed to optimise the topology and weight of a neural network. This method uses a GA to optimise the topology of the network and DE to optimise the weights. NDE is compared to state of the art algorithms for the double pole balancing problem and watershed management.

- The focus of Chapter 4 is energy systems. A Multi-Objective Neural Network trained with Differential Evolution (MONNDE) algorithm is proposed to address the Dynamic Economic Emission Dispatch (DEED) problem. MONNDE performs well when compared to state of the art multi-objective optimisation algorithms, and scales well for DEED variants with 5, 10 and 15 power generators.

- Chapter 5 applies evolutionary neural networks to forecasting problems in Ireland’s energy sector. Neural networks trained with a number of evolutionary algorithms are evaluated along with state of the art forecasting methods to predict Ireland’s power demand, wind power production and carbon dioxide emissions.

- The final set of experiments are presented in Chapter 6. This chapter applies evolutionary neural networks to the problem of host CPU utilization in data centers. A variety of evolutionary algorithms are applied to train a neural network to predict CPU utilization levels.

- Chapter 7 will conclude the research outlined in the thesis with a summary of contributions, and potential areas for future work.

- Appendices .1 - .4 can be found in Chapter 8, which contains all data sets, coefficients and graphs from experiments outlined in Chapters 2 - 6.

1.7 Background Literature

This section will give a comprehensive review of the literature relating to computing, artificial intelligence, evolutionary algorithms, neural networks and evolutionary neural networks (or neuroevolution) and their applications to both energy systems and the environment.
1.7. Background Literature

1.7.1 Computing and Artificial Intelligence

**Background**

The field of computing dates back to the 1800s. Charles Babbage created the first mechanical computer named the “Analytical Engine” capable of basic arithmetic in 1838 [56]. Ada Lovelace is thought to have written the first ever algorithm to run on the Analytical Engine in 1843 [130]. John von Neumann was another early computer science pioneer who invented the merge sort algorithm [215] and contributed to the invention of the Monte Carlo method [231], among many other contributions in the early to mid 1900s. George Boole proposed Boolean logic in 1954 which forms the basis of computing [50]. Claude Shannon is widely regarded as the father of information theory. His work on information theory in 1951 laid the foundations for the field of natural language processing [331]. Alan Turing was a key figure who made many early contributions to the field of computer science. His 1937 work proposed a universal computing machine that could compute any mathematical problem represented in algorithmic form [361]. Alan Turing is also regarded as the founder of Artificial Intelligence. In 1950, Alan Turing first asked the question “Can machines think?” and proposed what became known as the Turing test to determine if a machine is intelligent [360]. In short, the test determines that a computer is intelligent if a human interacting with the machine does not know that they interacting with a machine. John Searle proposed another test in 1980 called the Chinese room where a computer would read in Chinese characters and, by following a complex set of rules, produce the appropriate Chinese character as a response [325]. If the computer could do this in such a way that the Chinese speaker interacting with the machine thought that they were interacting with a human Chinese speaker, the machine would pass the test. These early researchers provided the foundation for today’s computer science and artificial intelligence research.

**Terminology and Methods**

When speaking about Artificial Intelligence (AI), the distinction is often made between narrow and general AI. The vast majority of AI research today would be considered narrow AI, meaning that it is capable of performing one particular task extremely well but nothing else. General AI refers to an all round machine that can perform many different tasks. General AI could be interpreted as the combinations of the sub
fields which are considered to be narrow AI. These include: Planning [381], Learning [261], Natural Language Processing (NLP) [236], Perception [126], Robotics [85] and Social Interaction [384]. A large part of early AI research in the 1970s and 1980s was concerned with developing expert systems using symbolic logic [158]. These were often developed using the programming language LISP, invented by John McCarthy between 1956 and 1958 [253, 254]. Some of the methods that are explored in AI research today include: evolutionary computing [113], neural networks [47], Bayesian networks [129], learning and classification methods [319]. Evolutionary computing and neural networks will be explained in Sections 1.7.2 and 1.7.4. Bayesian networks are concerned with modelling the probabilistic dependencies between variables. Classifiers are concerned with categorizing sets of data. Classification algorithms include: neural networks [47], support vector machines [86] and k-nearest neighbour [84].

There have been many achievements in the field of AI since Turing first asked if a computer can think. Some of the main achievements in the field include:

- Gerald Tesauro applied temporal difference learning to play backgammon in 1995 [356].
- IBM developed the Deep blue computer program to beat the world champion Garry Kasparov at chess in 1997 [62].
- Chellapilla and Fogel evolved a neural network to learn to play checkers in 1999 [71].
- Bongard et al. developed a robot that can adapt to damage and relearn to walk in 2006 [49].
- IBM developed the computer Watson that learned to play the word game Jeopardy better than the best human players in 2011 [121].
- Google Deepmind developed a computer program, called AlphaGo, that beat the world champion Go player Lee Sedol in 2016 [337].

1.7.2 Evolutionary Algorithms

Background

The field of evolutionary computing dates back to the late 1950s [21]. Some of the first pioneers in the area include George Box [52], Richard Friedberg [128], Hans
Bremermann [55], John Holland [168] and David Goldberg [140]. Lawrence Fogel proposed the evolutionary programming algorithm in the 1960s [123, 124]. John Holland introduced the first genetic algorithm in the 1970s [169, 170]. In the late 1980s, John Koza developed the genetic programming algorithm [218]. The 1990s and 2000s lead to the proposal of many new evolutionary algorithms. The most well known of these algorithms include: Covariance Matrix Adaptation Evolutionary Strategy by Nikolaus Hansen in 1996 [156, 157], Differential Evolution by Storn and Price in 1997 [348], Cartesian Genetic Programming in 2000 by Julian Miller [265], Neuroevolution of Augmenting Topologies by Stanley and Miikkulainen in 2002 [347], Nondominated Sorting Genetic Algorithm (NSGA) in 1994 by Srinivas and Deb [343] and NSGA-II by Deb in 2002 [93].

Inspiration

Evolutionary algorithms take inspiration from the fundamental driving force of the natural world: evolution by natural selection, first proposed by Charles Darwin in 1859 [68]. This idea says that in nature, biological organisms that are best adapted to their environment will be most likely to survive and reproduce to create the next generation. This is commonly referred to as survival of the fittest. A biological organism is defined by its genotype, which defines the organisms genetic makeup. The fittest individuals in the population will pass on their genes to the next generation. Evolution in the natural world can be viewed as the genes of a particular species being optimised to maximize the fitness of the species, i.e its ability to survive and reproduce. An example of this is the hunting abilities of predatory animals. Members of the population that are better at hunting would be better able to catch food which aids their survival. These individuals then go on to reproduce and pass on their genes for hunting to their offspring, such as eyesight, sense of smell, speed, etc. Over many generations, evolution will optimise the genes of predatory animals so that they become better at hunting. This is of course an over simplified account of what is happening. In reality, the ability of an organism to survive is not solely determined by one particular trait such as hunting ability. Many other factors also determine the fitness of an animal, e.g. the immune system. The same evolutionary process also applies to the prey animals being hunted, so that they evolve to become better at detecting predators and escaping. This is commonly referred to as an evolutionary arms race [90]. The idea of an evolutionary arms race has since made its way into the
field of evolutionary computing, referred to as coevolution [282].

It is this process of survival of the fittest that inspires evolutionary algorithms. A general blueprint for these algorithms is as follows:

1. Create a population of individuals. These individuals represent a solution to some problem and are analogous to genes in natural evolution.

2. Evaluate the fitness of each of the individuals, i.e. how good are they at solving a particular problem.

3. Select the best (fittest) individuals for reproduction to form the next generation.

4. Perform reproduction on the parent solutions to form new child solutions.

5. Perform mutations on the child solutions.

6. Repeat steps 2. - 6. until a predetermined number of generations have passed or a suitable solution has been found.

Figure 1.2 outlines a general blueprint of an evolutionary algorithm. This process can be applied to problems such as optimisation problems where a number of variables have to be selected in such a way that maximizes or minimizes an objective function. An example of such a problem is, how to invest capital so that profit is maximized. Algorithms such as genetic algorithms and differential evolution were designed for this class of problems. This evolutionary process can also be applied to function approximation. In genetic programming the solution represents a tree structure where each leaf on the tree represents a constant or problem variable and each non-leaf node represents a mathematical operator. Here the evolutionary process is applied to find the best tree that approximates some function with the highest accuracy. To use the same problem of investing capital, genetic programming can be applied to approximate a function that represents how the value of a particular stock changes over time. This is useful for forecasting how the price of that stock will change.

Motivation

Optimisation problems occur everywhere in the world. Designing race cars [350], locations of wind turbines [118] and vehicle routing [301] are all examples of optimisation problems. The goal of each of these problems is to maximize speed, maximize power output and minimize travel distance and emissions respectively. In order to
achieve these goals, the problem variable must be optimally selected. There are many examples of traditional mathematical optimisation algorithms, e.g. gradient descent [316] and Newton’s method [289]. The advantage of these algorithms is that they are rooted in mathematics, meaning that their convergence to the optimal solution can be guaranteed in many cases. The disadvantage of these methods is that they are not applicable to certain problem types, e.g. noisy, multi-modal, non separable and very large problems [400]. Noisy problems are problems where there is no smooth gradient, instead the problem landscape is rugged. Multi-modal problems are problems that contain many local optima, rather than a single global optimal point. Multi-modal problems have areas where the gradient leads towards a local rather than global optima. Many traditional mathematical approaches rely on problem gradients and therefore do not perform well on noisy or multi-modal problems. Separable problems, are problems where the variables are independent of one another. These are much
Chapter 1. Introduction

easier to solve as each variable can be optimised in isolation. Non separable problems are more difficult as the variables cannot be solved separately [187]. Very large problems refer to problems with a high number of problem variables. Mathematical approaches have difficulties with both non separable and large problems because of the time taken to find the optimum solution. There is a class of problems referred to as NP-Hard problems (non-deterministic in polynomial time). The complexity of these problems grows exponentially with the number of variables [81]. A solution to these problems can be evaluated quickly, however finding the optimum solution is a very computationally expensive task. It is these problems that motivate the use of evolutionary optimisation algorithms. Evolutionary algorithms make no assumptions about the search space of the problem and are therefore applicable to problems with a range of challenging properties such as: noisy, multi-modal, non separable and very large problems [400]. They are also capable of approximating NP-Hard problems.

Applications

Since the 1950s, evolutionary algorithms have been applied to a vast range of problem areas. A small sample of these applications include:

- In 2005 NASA designed the antenna of their Space Technology 5 spacecraft using a genetic algorithm [229].
- Genetic algorithms have been applied to health care by optimising the operation of an emergency department of a hospital in 2007 [398].
- The Multi-Objective Genetic Algorithm (MOGA) was applied to optimise the design of an aircraft wing in 1998 [284].
- The Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) algorithm was applied to the task of Funnel and gate systems design optimisation for the treatment of groundwater in 2007 [57].
- A multi-objective GA was applied to task allocation during the construction stage of software development in 2004 [105].
1.7.3 Swarm Intelligence Algorithms

Inspiration

Similar to evolutionary algorithms, swarm intelligence algorithms take inspiration from the natural world, in particular, from the collective intelligent behaviour that emerges from a decentralized group of individuals. There are many examples of this emergent behaviour that occurs in groups of individuals in nature: the schooling behaviour of fish [202], the flocking behaviour of birds [308], colonies of ants [276] and colonies of bees [175]. Each of these species exhibit seemingly intelligent behaviour when observing the group as a whole. However at an individual level, each member of the group is following very simple rules. A study by Craig Reynolds in 1987 demonstrates this by simulating flocking behaviour in birds [308]. Each bird observes its environment locally and obeys the following simple rules to update its position:

1. Avoid colliding with nearby individuals.
2. Travel at a similar velocity to nearby individuals.
3. Maintain a close proximity to nearby individuals.

This results in flocking behaviour by the group as a collective. Swarm intelligence algorithms use this principal of distributed individuals obeying simple rules to achieve a more complex goal.

Research Areas

Swarm intelligence methods have proven to be applicable to a wide range of research areas. One such problem that swarm intelligence has been applied to is crowd simulation and crowd control. The Notting Hill Carnival is one of the largest music festivals in London and attracts over a million festival goers each year. In 2003, Batty et al. used swarm intelligence methods to model the movement of crowds with the aim of developing safer strategies of crowd control [33]. Another area that swarm intelligence has been applied to is swarm robotics. In 2009, Waharte et al. proposed coordination strategies from swarms of unmanned aerial vehicles (UAVs) for the task of search and rescue [373]. Swarming behaviour has also inspired research in Multi-Agent Systems (MAS). In 2009, Xiao et al. proposed a method for controlling large groups of agents...
that was inspired by swarm behaviour [388]. In 2002, Verbeeck and Nowe explored the relationship between swarm intelligence and learning for the task of foraging [369]. A 2008 study by Lemmens et al. demonstrated that a bee inspired multi agent system is more efficient at collecting food and more scalable than other approaches [226]. Ducatelle et al. explored the relationship between swarms of robots and self organizing systems (SOS) in 2010 by simulating two swarms of different types of robots that adapt to the task of foraging for food [102]. The image in Figure 1.3 illustrates one of the largest swarm of physical robots to date by Rubenstein et al. in 2014 [315]. This study demonstrates self assembly in 1024 robots, named kilobots. This robot swarm was able to create complex two dimensional shapes through local interactions with one another only. Finally, one of the most well established application areas of swarm intelligence is optimisation. This will be discussed in the next section.


Optimisation

As highlighted in Section 1.7.2, optimisation problems feature in a diverse range of domains. While there have been many optimisation algorithms that have been proposed based on the evolutionary survival of the fittest philosophy, there have also been a number of optimisation algorithms proposed that take inspiration from the emergent intelligence that is observed from a collection of individuals. The most popular of
these are: Particle Swarm Optimisation (PSO) first proposed by Kennedy and Eberhart in 1995 [205] and Ant Colony Optimisation (ACO) proposed by Dorigo et al. in 1992 [98, 99]. PSO is an optimisation algorithm designed for global optimisation problems with real valued variables. The algorithm was originally inspired by the flight of birds. It operates by simulating a number of individuals (named particles), that traverse the problem space and evaluate candidate solutions to the optimisation problem. The particles each follow simple rules to update their position (which represents a solution) based on the position found with the highest fitness. Eventually the particles converge on the best known position. ACO was designed for discrete optimisation problems. The algorithm operates by simulating the behaviour of a colony of ants. The route an ant takes is analogous to a solution to the optimisation problem. If the solution has poor fitness, the ant is said to have taken more time to traverse the route. As the ant traverses a route, it lays down a pheromone trail. When the ant takes longer to traverse a route with poor fitness, more of the pheromone will evaporate. This makes it less likely for ants to traverse this route again. Conversely they are more likely to traverse routes that have a strong pheromone trail as they have a higher fitness.

Applications

The motivation for using these swarm intelligence optimisation algorithms is the same as for evolutionary optimisation algorithms mentioned in Section 1.7.2. They make no assumptions about the problem space and are resilient to both noise and multiple local optima, and are also applicable to large problems with many variables. Both PSO and ACO have been applied to similar problems as evolutionary optimisation algorithms. These include:

- ACO has been applied to the travelling salesman problem [300]. This is an NP-Hard problem where the optimisation algorithm must find the fastest route that visits each city and returns to the original city.

- PSO has been applied to scheduling work flows in the cloud in 2010 [291]. Computer applications were scheduled to minimize both computation cost and data transmission cost.

- In 2007, He and Wang applied PSO to a number of engineering design problems, including welded beam design, spring design and pressure vessel design [162].
• Silva et al. applied ACO to supply chain management in 2009 [336]. This study optimised each element in the supply chain to make the overall operation more efficient.

• In 2017, it was shown that PSO aid the search for gravitational waves by reducing the number of evaluations to less than 10% of what grid based search requires [375]. In 2018, PSO was also shown to increase the sensitivity of gravitational wave detectors by 10% by optimising the parameters of the detector [262].

1.7.4 Neural Networks

Background

Neural networks are computational models that are inspired by the biological brain [47, 160]. Neural network research dates back to the mid 1900s. The first computational model of a neural network was created by McCulloch and Pitts in 1943 [255]. This early neural model did not learn [323]. The first neural network that was capable of learning was proposed by Donald Hebb in 1949. Hebb proposed an approach for unsupervised neural network learning that is now referred to as Hebbian Learning [137]. Rochester et al. built on the work of Hebb by conducting experiments on networks with 99 and 512 neurons [310]. These were then referred to as calculators. Rosenblatt proposed the perceptron in 1958 for pattern recognition [311]. Other early contributors to the field include Marvin Minsky [266], Bernard Widrow and Marcian Hoff [379], Aleksei Ivakhnenko and Valentin Lapa [185] and Stephen Grossberg [150].

Training a neural network using backpropagation was introduced in the 1970s by Paul Werbos [377] and was popularized by Rumelhart, Hinton and Williams in 1986 [317]. Other significant advances in neural networks include convolutional neural networks for image processing [39], and long short term memory (LSTM) cells [166] for speech recognition [386]. Juergen Schmidhuber provides a comprehensive overview of the history of neural networks [323].

Neural Model

A typical neural network comprises of an input layer of neurons, one or more hidden layers and an output layer. A deep neural network refers to networks with more
than one hidden layer. A neuron is a basic processing unit that receives one or more signals from other neurons (or the input signal if the neuron is in the input layer). The neuron then produces an output signal based on the input signals and an activation function. Typically the Sigmoid function\(^1\) is used as an activation function. Each neuron in a layer is connected to every neuron in the previous and next layers by weighted synapses in a fully connected network. Modifying the values of these weights is what enables the network to produce a desired output for a particular input. How to optimise the network weights is one of the core challenges of neural network research. Neural networks first read in a signal through the input layer. This signal is then passed forward to the neurons in the next layer by weighted connections. These neurons then produce outputs which are passed forward to the next layer. This process is repeated until the signal reaches the output layer and the network produces an output signal. Figure 1.4 illustrates the architecture of a neural network.

One of the most popular methods to train neural networks is the backpropagation algorithm [317]. This algorithm operates by calculating the error between the desired and observed network outputs, and passing the error signal back through the network to update the weights. This method requires the target outputs to be known and is therefore only applicable to supervised learning problems. For problems where the target outputs are not known (for example game playing), algorithms such as reinforcement learning are often applied. Some of the most popular reinforcement learning algorithms include: Q Learning [374], State Action Reward State Action (SARSA) [318] and Temporal Difference Learning [353]. These methods typically operate as follows: (1) Observe the environment. (2) Take an action. (3) Receive a reward. Evolutionary neural networks are also a suitable method for these kinds of problem. This will be discussed in detail in Section 1.7.5.

Problem Domains

The ability of neural networks to reliably produce the desired outputs for given inputs has proven to be of great benefit to many problem domains. Some of the problem domains where neural networks have had great success include:

- **Control** [135]. This area is concerned with developing control systems for continuous and dynamic environments.

---

\(^1\)Sigmoid function: \( f(x) = \frac{1}{1+\exp(-x)} \)
Chapter 1. Introduction

Figure 1.4: Neural Network

- **Classification** [100]. The task of classification consists of developing methods to categorize data.

- **Regression** [342]. These problems relate to identifying relationships between continuous variables.

- **Robotics** [34]. The multidisciplinary field of robotics is concerned with the simulation, design, creation and operation of robotic systems.

- **Forecasting** [19]. The task of forecasting can be defined as making predictions into the future given past and current information.

- **Image and speech recognition** [338, 147]. These complex problems relate to enabling a computer to identify images and language.

- **Game playing** [267]. This area consists of developing computer programs that can play games.

1.7.5 Evolutionary Neural Networks

Motivation

The most commonly used algorithm to train neural networks is the backpropagation algorithm, as highlighted in the previous section. Backpropagation however requires
that the target outputs of the network are already known. This is the case for cer-
tain problems such as classification. The backpropagation algorithm can use historic
labeled data to calculate error signals. There are many problem types where the tar-
get outputs of the network are not known however, e.g. reinforcement learning and
control problems. Evolutionary methods are well suited to these problems as they
do not need target outputs to train the network weights. Evolutionary algorithms
have the advantage of only needing a fitness function to determine the suitability of a
given network. These are generally readily available. The process of evolving neural
networks is also often referred to as neuroevolution. Advantages of evolving neural
networks include [264]:

1. Easily parallelizable. Since evolutionary algorithms consist of populations of
solutions these can be evaluated in parallel without much difficulty.

2. Less susceptible to local optima. Evolutionary algorithms are global optimi-
sation algorithms which means that they are suitable for problems that have
many local optimal, e.g. neural network weight training.

3. Do not need target outputs, as previously stated.

Some of the disadvantages include:

1. Lack of theoretical guarantees of finding the optimum network. There is no way
of knowing if the network produced by an evolutionary method is in fact the
 optimum networks.

2. Scalability. Until recently it was thought that evolutionary methods do not
scale well to suitably evolve large deep neural networks. Recent results indicate
however that modern evolutionary methods are capable of evolving suitable
deep neural networks [321, 351]. This will be discussed later in this section.

The process of evolving neural networks operates by applying a given evolutionary
algorithms to the network parameters. These always include the network weights, as
these are the primary factor that determines the network output. For example the
differential evolution algorithm can evolve the weights of the network by setting the
weights of the network as the differential evolution agent’s position. Over time, the
algorithm produces new agent positions which are set as the network weights. These
new networks are evaluated on the problem and the fitness of the network is passed
back to the algorithm.
It is also common to evolve other network parameters including: the number of layers, the number of neurons, the activation function, network connections, etc. There are advantages and disadvantages of evolving other network parameters [117]. The advantages include:

1. There is no need for parameter selection when evolving the network size. When evolving just the network weights, it is often necessary to do a lot of parameter tuning before hand to find the network size gives the optimum performance. When dynamically evolving the network size during run time, there is no need to find the optimum size network before hand.

2. Reduced complexity. By evolving the connectivity of the network, it is possible to evolve a partially connected network that has fewer connections.

3. Reduced hardware requirements. A network with fewer connections will have fewer components in physical implementations.

Some of the disadvantages associated with optimising other network parameters include:

1. Increased algorithmic complexity. Increasing the number of network parameters and parameter types to evolve will increase the complexity of the optimisation problem thus making it more difficult for the evolutionary algorithm to produce suitable networks.

2. Increased computational time. Evolving the network size will involve evaluating networks of different size and optimising the weights for these different networks. This increases the computational burden on the evolutionary algorithm.

Terminology

When evolving neural networks, aspects of the neural network design are encoded into a genotype or chromosome. These typically contain network information such as synaptic weight values, number of neurons, connectivity, etc. These network traits form the genotype. For example, when evolving just the network weights. The genotype would be the list of values associated with the network weights. These genotypes are evolved over a series of generations. The phenotype is the expression of the genotype, i.e. the phenotype is the actual neural network.
Neuroevolution methods can be sub divided into direct encoding schemes and indirect encoding schemes. In direct encoding schemes, the bits in the genotype are mapped directly onto the phenotype [272, 143]. Indirect encoding schemes are more abstract. Instead the genotype provides more general rules as to how the phenotype should be formed [235, 212, 372]. These indirect schemes are advantageous in forming large networks as they do not require large genotypes.

**Methods**

There have been numerous approaches to evolving neural networks. In 1994 Gruau proposed the Cellular Encoding (CE) algorithm which utilizes genetic programming to effectively grow a neural network from a single neuron [151]. The network starts from a single neuron. The genotype contains the instructions to grow the network topology and weights. Another early approach includes the GeNeralized Acquisition of Recurrent Links (GNARL) algorithm proposed by Angeline et al. in 1994 [12]. GNARL uses evolutionary programming to evolve both the topology and weights of the network. The weights of the network are mutated using a Gaussian distribution. The Evolutionary Programming Neural Networks (EPNet) algorithm was proposed in 1997 by Yao and Liu [396]. This method stores the weights, connectivity and nodes in three separate matrices. These describe what the weight values of each connection are, whether connections between nodes are activated and whether each node is in use. EPNet then uses evolutionary programming, backpropagation and simulated annealing\(^2\) to evolve the network.

NeuroEvolution of Augmented Topologies (NEAT) is one of the most popular evolutionary neural network algorithms and was first proposed by Stanley and Miikkulainen in 2002 [347, 345]. This method uses a GA to evolve both the network topology and weights. It employs a direct encoding scheme and begins with a minimal network. As the algorithm runs, more complexity is added to the network by adding neurons and connections via mutation. Genes are divided into species based on their innovation number. The innovation number is a counter that records the number of species. NEAT remembers the innovation number of each gene so that crossover can occur between networks with different topologies. The distance between genes can be easily measured by comparing their chromosomes. The terms genotype and chromosome are used interchangeably for methods with direct encoding schemes. When comparing

\(^2\)Simulated annealing is an optimisation algorithm inspired by the cooling of materials [330]
two chromosomes, the genes that do not match are considered to be either disjoint or excess. The distance between genes is therefore just the sum of the number of disjoint and excess genes. Genes are categorized into species based on this distance. Speciation refers to the emergence of a new and distinct species throughout the evolutionary process. This speciation allows the algorithm to protect topological innovation. Later the NEAT algorithm was extended to the Hypercube-based NEAT (hyperNEAT) algorithm [346]. HyperNEAT operates using the same basic principles as the NEAT algorithm. The algorithm uses Compositional Pattern Producing Networks (CPPNs) to evolve large scale neural networks. Unlike NEAT, hyperNEAT utilizes indirect encoding via the CPPNs which determines the networks connectivity.

Enforced SubPopulations (ESP) is another popular method of evolving neural networks [142]. This algorithm was proposed by Gomez and Miikkulainen in 1999 and is an extension of the Symbiotic Adaptive Neuro-Evolution (SANE) algorithm by Moriarty and Miikkulainen, originally proposed in 1996 [271]. Unlike NEAT, ESP uses a fixed neural network size. In ESP, the population is divided into distinct sub-populations. Networks are formed by selecting a neuron from each subpopulation and combining them to form a network. Crossover only occurs between chromosomes within the same subpopulation. No inter subpopulation breeding is allowed. Offspring also remain within the parents’ subpopulation. Cooperative Synapse NeuroEvolution (CoSyNE) was proposed by Gomez et al. in 2008 [141]. The algorithm constructs networks from subpopulations similar to ESP. The population of network weights is represented as a matrix. CoSyNE permutes the weights so that a weight is potentially part of a new network to enable the coevolution process. The idea of coevolution was also a key component of the Cooperative Coevolutionary Neural Network (COVNET) algorithm proposed by Garcia et al. in 2003 [134]. COVNET also operates by evolving subpopulations similar to ESP and CoSyNE.

More recently, Turner and Miller proposed the use of cartesian genetic programming to evolve neural networks [363, 362]. Cartesian genetic programming encodes a graph representation of a computer program, which makes it suitable to evolve neural networks.

For neural network weight optimisation, there are numerous successful applications of real valued global evolutionary optimisation algorithms, such as those outlined in Section 1.7.2. Recent examples include evolving a recurrent neural network to predict the contact area that car tires make with the ground using DE [103], evolving
1.7. Background Literature

a neural network using PSO to predict building structural failure [70], evolving a network using DE and novelty search for maze navigation [244], evolving a neural network for face detection using CMA-ES [177] and playing Atari games using a Natural Evolutionary Strategy (NES) trained neural network [321]. In many cases, simply applying these global evolutionary optimisation algorithms to the weights of neural networks is equally as effective as many of neuroevolution methods highlighted above. In 2003, Igel demonstrated that a network trained with CMA-ES performs better than many other neuroevolution methods at the control task of pole balancing [176].

Performance Comparison with Other Methods

Evolutionary neural networks perform very competitively when compared to other methods on a wide range of problems. When compared to reinforcement learning algorithms at the task of pole balancing, Gomez et al. demonstrated that evolutionary neural networks outperform many state of the art reinforcement learning methods such as Q-Learning, SARSA and actor critic learning [141]. Similarly Whiteson and Stone demonstrated that the NEAT algorithm outperformed Q Learning on the mountain car problem and the task of server job scheduling [378]. Interestingly their results also illustrate that combining NEAT with Q learning provided the best results. Drugan gives a recent (2018) comparison between evolutionary methods and reinforcement learning [101].

Earlier, it was stated that one of the barriers to implementing evolutionary methods for deep neural networks was that they do not scale well. Recent 2017 results by Salimans et al. [321] and Such et al. [351] indicate that evolutionary methods can perform competitively when evolving larger networks using NES for deep reinforcement learning problems. Salimans et al. found that after training, evolving the network performed better in 23 Atari games tested and worse in 28 when compared to the state of the art deep reinforcement learning algorithm Asynchronous Actor-Critic Agents (A3C) [321]. Such et al. reported similar findings on the Atari games environment and demonstrated that evolving deep neural networks can be further enhanced using novelty search [351].

It has also been shown that evolving neural networks can perform competitively when compared to state of the art gradient based methods for supervised learning problems. In 2016, Morse and Stanley demonstrated that the Limited Evaluation Evolutionary
Algorithm (LEEA) can rival Stochastic Gradient Decent (SGD) and RMSProp when tested on function approximation, time series and regression problems [273].

In summary, the purpose of this section was to motivate the use of evolutionary algorithms for neural network training. The studies mentioned above have highlighted that evolving neural networks is an effective approach that performs well on a wide range of problems. Next, the applications of evolutionary neural networks will be explored.

Applications

Evolutionary neural networks is a highly active area of research. The purpose of this section is to demonstrate that neuroevolution is also incredibly useful with many practical real world applications. This section will highlight some of the main application areas of evolutionary neural networks. In terms of problem types evolutionary neural networks are applicable to time series, control, classification, regression, robotics, art generation, electronics, embedded systems, etc. It would be infeasible to list all of the applications of evolutionary neural networks here. Instead the following list contains a diverse range of interesting applications of evolutionary neural networks:

- **Particle physics.** In 2009, neuroevolution was utilized to separate real signals from background noise in order to observe the production of top-quarks at the Fermilab particle collider [1].

- **Medicine.** Magoulas et al. evolved a neural network with DE for medical diagnosis based on colonoscopy images in 2004 [232].

- **Financial markets.** Chang evolved partially connected network using genetic algorithms to forecast stock prices in 2012 [67].

- **Aerial robotics.** In 2010, Salichon and Tumer utilized an evolved neural network controller to control micro aerial vehicles [320].

- **Theorem proving.** Desai and Miikkulainen demonstrate that the SANE algorithm was capable of learning natural deduction in 2000 [94].

- **Video games.** In 2009, Togelius et al. evolved a neural network capable of playing Super Mario [358].
• **Art.** Secretan et al. evolved neural networks that were used in conjunction with human input to generate artistic images [326].

• **Biology.** Grisci and Dorn utilized neuroevolution to predict structural features in protein in 2016 [149].

## 1.7.6 Energy and The Environment

**Energy**

The generation of energy is a fundamental challenge faced by society today. Increasing living standards, industrial development and an increasing global population has resulted in an ever increasing demand for energy. For example, Figure 1.5 illustrates the increase in the population of Ireland from 1990 to 2016 along with an overall increase in residential energy consumption. Similar trends are observed worldwide with a growth of residential energy consumption of 14% from 2000 to 2011 [277]. This increasing energy demand is observed across all sectors [183]. The sources of energy available for use can be categorized into renewable and non-renewable energy resources. Renewable energy resources refers to resources that replenish themselves when consumed. Examples of these include: wind, solar, tidal, wave, geothermal and biomass. Non-renewable resources are resources that do not replenish once they are used, e.g. coal, oil, nuclear and gas. The primary advantage of renewables is that they have less of an environmental impact than non-renables. The primary disadvantage of renewables is that many of them are not reliable. For example, wind speed changes regularly which impacts the energy available from wind turbines. Proposed solutions to this reliability issue include, battery technology [355] and developing larger electrical grids, e.g. a European supergrid [365]. Non-renewable resources are cheap and reliable but have a much worse effect on the environment. Figure 1.6 shows the energy consumption of Ireland in 2016. In this image, energy consumption is broken down into three categories: Heat, Electricity and Transport. Each of these consume approximately a third of the overall 167,592 GWh of energy consumed in Ireland in 2016 [183].
Environmental Impact

The demand for energy worldwide is constantly increasing, as discussed in the previous section. In 2016, only 8% of Ireland’s total energy consumption (heat, transport and electricity) came from renewables, while 91.9% came from fossil fuels [183]. The emission of carbon dioxide (CO$_2$) is one of the primary side effects with burning fossil fuels as an energy source [167]. Carbon dioxide and other greenhouse gases, such as nitrous oxide (NO$_x$) and ozone (O$_3$), have been identified as the leading cause of global warming [87]. The industrialization of developing countries is expected to exacerbate this problem in the coming years [259]. This is due to the high energy demands in many industries, e.g. steel production, data centers, transport, oil and gas [233]. Climate change is considered to be one of the greatest current threats to human existence [357]. Climate change is also responsible for many of the negative changes to our environment, including:

- Bleaching the coral reefs [174]. This has disastrous effects on the surrounding ecosystem.

- Warming the oceans and therefore melting the sea ice [227, 223].

- Increased sea levels [41]. This increases the likelihood of flooding in coastal cities.
Extreme weather events, e.g. hurricanes and drought [312]. This negatively impacts food production and poses a serious risk to both human life and the surrounding ecosystem.

Almost 200 countries around the world have pledged to reduce their emissions of greenhouse gases in the 2015 Paris Climate Conference (COP21) to help address this problem. Figure 1.7 shows that Ireland has managed to reduce its CO$_2$ emissions since 2005.
Greenhouse gas emissions are not the only concern with regards to energy generation. Air pollutants from the exhaust of diesel engines have been found to have negative health effects [197]. The building of oil and gas pipelines often results in habitat destruction of wildlife [73]. Extracting gas by hydraulic fracking can pose risks to surrounding aquatic ecosystems [58]. Extracting oil from the ocean can result in oil spills which have disastrous effects on the surrounding ecosystem [211].

**Environmentally Friendly Energy Production**

Considering the environmental effects listed above, it is therefore of utmost importance to further advance environmentally friendly methods of energy generation. As Figure 1.6 illustrates, Ireland’s transportation is heavily reliant on oil. Electric cars are an alternative to fossil fuel powered vehicles. With advancements in battery technology, it is thought electric cars will become more viable into the future with many companies already producing electric vehicles [367]. These electric cars will then be charged from the grid. In order for these electric vehicles to be charged in an environmentally friendly way, the electrical grid needs to be powered from renewable sources. Ireland gets most of its renewable energy from wind, however as previously stated, wind energy is not reliable. Aside from battery technology, it is also vital to have accurate forecast models to incorporate wind energy to meet the power demand. In order for thermal power generators to be scheduled optimally to minimize cost and emissions, it must be known in advance how much energy is available from wind turbines and what the power demand will be. This is due to the physical ramp up/down limits of power generators [30]. There are similar concerns with generating energy from hydropower sources such as rivers. Forecasting methods are important as the amount of energy available from hydropower in a river is dependant on the volume of water in the river and the amount of rainfall [155]. Building dams also has a severe effect on the surrounding ecosystem and can result in habitat destruction for wildlife [210]. It is therefore crucial that watershed is correctly managed to protect the habitat of wildlife [380]. It is clear that meeting energy demands in an environmentally friendly manner is a complex task, due to the many uncertainties and inter-dependencies. In order to address these issues, effective forecasting, control and optimisation strategies are needed.
Machine Learning and Optimisation

There are numerous examples of successful applications of machine learning and optimisation to energy systems and environmental problems. Power generation is one such area where evolutionary optimisation algorithms have had a significant impact. The problem consists of optimising the power output of a number of generators in order to minimize cost. Swarm and evolutionary algorithms such as differential evolution [31] and particle swarm optimisation [234] have been applied to this problem. Other studies have looked at incorporating wind [165], tidal [275] and solar [17] energy into the power generation scheduling process. The problem can also be stated as a multi-objective optimisation problem\(^3\) where both cost and greenhouse gas emissions must be considered [30]. Feedforward and Hopfield neural networks have also been applied to the economic dispatch problem [352, 349]. Neural networks have been applied to many forecasting problems in energy production. Examples of these forecasting problems include: wind [44], solar [5] and hydropower [75].

Many environmental problems have also benefited from optimisation and machine learning methods. Evolutionary algorithms have been applied to watershed management [274]. Evolutionary neural networks have been applied to predict air pollution [280]. Genetic algorithms and neural networks have been applied to reduce harmful diesel engine emissions [9]. Many industries that consume large amounts of energy have benefited positively from machine learning methods. The application of genetic algorithms has led to more energy efficient data centers [354]. Neural networks have improved the energy efficiency of ships [42], industrial furnaces [131] and hybrid electric cars [270]. Smart buildings is a recent approach to increasing the energy efficiency of buildings, e.g. using smart thermostats [91]. Neural networks have successfully predicted the energy consumption of residential buildings [48] and genetic algorithms have improved energy efficiency in building design [399].

Other machine learning approaches have also been applied to increase energy efficiency. Reinforcement learning was shown to improve the energy efficiency in wireless sensor networks [263] and household devices such as coffee machines and water dispensers [366]. Reinforcement learning has also been applied to reduce energy consumption in cloud computing [110, 332]. Multi agent systems have been shown to reduce reliance on fossil fuels in smart grids using energy storage [306]. Each of these

\(^3\)Multi-objective optimisation consists finding the set of solutions that optimise each objective to varying degrees [78]
previous studies have demonstrated that optimisation and machine learning can have a positive effect in reducing energy consumption and protecting the environment. Generating energy in a responsible and environmentally friendly manner is a complex task. As outlined above, reducing the negative impact of meeting modern societies energy needs can be achieved at many different levels. This ranges from reducing our consumption through increased efficiency to reducing our reliance on fossil fuels and further incorporating renewable energy resources when generating electricity. Machine learning and optimisation methods can help achieve this, which provides the motivation for the research outlined in this thesis.

Summary

This section has outlined the problem of generating energy in an environmentally friendly manner, and also how machine learning and optimisation methods can have a positive impact in addressing this problem. Chapters 2 - 6 of this thesis will present new techniques and applications that further demonstrate how machine learning and optimisation techniques can be beneficial for problems relating to energy systems and the environment. Chapter 2 will apply a PSO trained neural network to the task of watershed management. Next, Chapter 3 will address the problem of watershed management using evolutionary neural networks. The multi-objective problem of power generator scheduling will be addressed in Chapter 4 by evolving a multi-objective neural network. Forecasting Ireland’s power demand, CO$_2$ emissions and wind power generation using evolutionary neural networks will be explored in Chapter 5. Finally, Chapter 6 will investigate the application of evolutionary neural networks to predict CPU utilization. CPU utilization is a crucial factor in managing data centers. It is therefore vital to have accurate predictions that will enable data centers to be managed effectively and therefore reduce energy consumption.
Chapter 2

A Meta Optimisation Analysis of Particle Swarm Optimisation

The work outlined in this chapter was published in:


2.1 Introduction

Particle Swarm Optimisation (PSO) is an optimisation algorithm that consists of a number of particles exploring a problem space and ultimately converging on a solution [205]. The particles evaluate potential solutions and share information with one another. This information is used to direct their movement so that they move towards the best known solutions. The PSO algorithm has been applied to numerous real world problem domains since its first proposal. These include design, scheduling and routing problems across several disciplines and industries ranging from imaging to energy production [8]. The main advantages of meta heuristic optimisation algorithms such as PSO, Differential Evolution and Scatter Search is their robustness, versatility and applicability to a wide range of problems. Traditional optimisation
methods such as gradient decent struggle with certain problems, e.g. problems with noise and problems that are otherwise non-differentiable. Such problems do not pose a problem to PSO. Another class of problems that PSO is well suited to are very large problems. Problems that are classified as NP-complete or NP-Hard (non-deterministic polynomial-time). These problems increase in size at an exponential rate as the number of parameters increase and are therefore much too time consuming for deterministic optimisation algorithms to solve. Algorithms such as PSO can provide good approximations to these problems however. The drawback to PSO is that it is heavily reliant on correct parameter selection and cannot guarantee that it will converge on the optimum solution. This research will aim to address the former of these two issues. There are more comprehensive studies that outline in detail the advantages and disadvantages of heuristic optimisation algorithms [400].

2.1.1 Velocity Update Equations

This research will focus on the area of particle movement, in particular how the particles update their velocity. There has been a wealth of research published in this area, exploring all aspects of how best to update the particles’ velocity. When the algorithm was first proposed in 1995, a particle’s velocity was updated based on its velocity at the previous time step, the particle’s best previous position and the groups best previous position [205]. An inertia term was added at a later date to better enable the particles to converge [333], this was followed shortly by a linearly decreasing inertia term [334]. It was later shown that implementing a constriction term rather than the inertia term is mathematically guaranteed to provide stable convergence [76]. An example of a more recent development on the inertia term is the adaptive inertia weight which changes between the maximum and minimum value based on the success of particles on improving their fitness between iterations [278].

There are many velocity update equations in the literature that implement novel operators within the velocity update equation in an effort to improve performance. The Attractive Repulsive PSO is an example of this [309]. This algorithm uses a diversity measure to determine when the swarm has converged. If the swarm is deemed to have converged, the particles then move away from the best locations found. A similar idea was proposed in the form of the PSO CV where a velocity control parameter was implemented as a means giving the particles’ extra velocity when they slowed down closer to the optimum [171]. The Dissipative PSO is another
PSO variant which gives a particle a random velocity and location if a random number generated is below a crucial value [390].

Incorporating the worst locations of the problem space into the motion of the particles was first proposed in 2005 [392]. This velocity update equation only took the worst locations into account when updating the particles’ velocity and as such only avoided the worst locations rather than converging on the best locations. This issue was addressed in later research that also enabled the particles to converge [329, 188, 249, 241, 250].

In an opposite direction of much of the above research there has also been efforts made to reduce the complexity of the velocity update equation, most notably the Many Optimising Liaisons (MOL) [297] and the Adaptive Velocity PSO (AV PSO) [13]. MOL is a PSO variant that simplifies the velocity update equation by removing the influence of a particle’s personal best position and focuses solely on the group’s best position. The AV PSO aimed to reduce the complexity of the algorithm by removing most of the parameters that have to be tuned for the algorithm to run optimally. Instead this variant utilizes the Euclidean distance of a particle from the best location to update its velocity.

2.1.2 Meta Optimisation & Parameter Selection

When implementing the PSO algorithm, it is important to ensure that adequate parameters are selected. Parameter selection itself can be thought of as an optimisation problem. There are generally two approaches to addressing the issue of parameter selection: 1) A parameter sweep consisting of a brute force search. 2) Meta Optimisation. A parameter sweep is sufficient when only one or two parameters need to be selected, however it becomes very computationally expensive as the number of parameters increases. For this reason meta optimisation will be utilised in this paper to ensure each velocity update equation is tuned to give the best performance. Meta optimisation refers to applying an optimisation algorithm to the parameters of another optimisation algorithm. The field of meta optimisation dates back to 1978 when it was first applied to tune a genetic algorithm [258]. In the years since, meta optimisation has been applied to ant colony optimisation [46], differential evolution [296], COMPLEX-RF [221], particle swarm optimisation [256, 297] and genetic algorithms [148, 20, 204].
There are limitations to the previous studies conducted in applying meta optimisation to PSO. The first study in this area, the Optimised PSO (OPSO), optimises the PSO parameters separately per problem [256]. It is considered desirable within PSO research to find PSO parameters that give the best performance over a range of problems rather than just one. Although the authors of the OPSO presented their algorithm which incorporates meta optimisation as an optimisation algorithm itself rather than using meta optimisation to tune the PSO parameters, as done in later research [297] and in this research. The second limitation with each of the previous applications of meta optimisation to PSO is the size of the experiments conducted. With regards to the OPSO, meta optimisation was only applied to 1 PSO variant for 1 problem at a time [256]. In the later Many Optimising Liaisons (MOL) [297], meta optimisation was applied to 2 PSO variants over 5 problems. The research presented in this paper is much more comprehensive in terms of scale, implementing 20 PSO variants based on their velocity update equations evaluated over 8 problems.

### 2.1.3 Watershed Management

In order to gauge the performance of each velocity update equation, they will be applied to 8 optimisation problems. Of these problems, 7 are standard test optimisation functions. The final problem is a novel application of PSO to a neural network function approximator for the Watershed Management problem. This problem is essentially a resource management problem [394]. The resource in question is water. There are a number of interested parties that wish to withdraw water from the system. These include water to sustain a city, water for farm irrigation, water for hydroelectric power generation and water for the surrounding ecosystem. The problem consists of many constraints and multiple flow scenarios. Previous approaches to addressing this problem include Multi Agent Systems (MAS) [394, 11], Multi Agent Reinforcement Learning (MARL) [252], MAS combined with Genetic Algorithm [24], Multi Population Evolutionary Algorithm [120] and PSO [252]. The single neural network function approximator approach implemented in this research differs from many of these previous methods as it does not implement multiple individual controllers to each problem variable. There are many examples in the literature of training a neural network using PSO, however this is the first application of such approach to the watershed management problem. In the previous application of PSO to the Watershed Management problem, PSO was applied directly to the problem variables to find the
optimum solution. This application of PSO to the watershed management problem is far more novel and interesting as it utilizes PSO in an offline learning manner to train a neural network to be able to find the optimum solution to the problem.

### 2.1.4 Contribution & Structure

The contributions of this paper are as follows:

1. Provides a comprehensive comparison of a wide range of velocity update equations using meta optimisation.

2. Establishes which set of parameters leads to the optimum PSO performance.

3. Addresses the questions: Does a constricted PSO perform better than a PSO with Inertia? Does a stable PSO convergence result in better performance? What is the best performing velocity update equation?

4. Investigates if hybridizing various velocity update equations will result in better performance.

5. Provides a novel application of a neural network function approximator to the Watershed Management problem.

The rest of the paper is structured as follows: Section 2 will provide a detailed outline of the Particle Swarm Optimisation algorithm and the velocity update equations that will be examined. Section 3 will explain the concept of Meta Optimisation along with how it will be applied in this research. In Section 4, concepts such as Learning, Artificial Neural Networks and the application of PSO will be described. Section 5 will present the Watershed Management problem. The results of each of the experiments will be presented in Section 6 and finally, the conclusions that can be drawn from these results will be made in Section 7. Here, future work will also be outlined.

### 2.2 Particle Swarm Optimisation

The PSO algorithm consists of a number of particles whose purpose is to evaluate candidate solutions and eventually move towards the best solution [205]. Initially these particles are distributed throughout the problem space with a random position and random velocity. The position and velocity of a particle at a time $t$ are referred to
as $\vec{x}_t$ and $\vec{v}_t$ respectively. At each time step each particle evaluates its position within the problem space defined by an objective function. This objective function measures the fitness of the particles current position which represents a candidate solution. Every particle remembers its previous best position. If a new position has a better fitness than the previous best position for that particle, the particle will remember this new position as its personal best position $\vec{p}_b$. Each particle also has access to the best position within its neighbourhood of particles $\vec{g}_b$. The other particles within a particle’s neighbourhood are dictated by the topology used. Each particle updates its velocity, and as a result its position, using its own best position and that of its neighbours. Balancing this cognitive and social behaviour is critical to the success of the PSO. The motion of the particles throughout the problem space is defined by their equations of motion below:

\begin{align*}
\vec{v}_{t+1} &= \chi (\vec{v}_t + r_1 c_1 (\vec{p}_b - \vec{x}_t) + r_2 c_2 (\vec{g}_b - \vec{x}_t)) \\
\vec{x}_{t+1} &= \vec{x}_t + \vec{v}_t
\end{align*}

Where $r_1$ and $r_2$ are random numbers between 0 and 1. The terms $c_1$ and $c_2 = 2.05$ are acceleration coefficients. The $\chi$ term is the constriction factor and is defined as:

\begin{align*}
\chi &= \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|} \\
\varphi &= c_1 + c_2
\end{align*}

Where $\chi \approx 0.72984$, and $c_1 = c_2 = 2.05$ have been mathematically proven to provide stable convergence [76]. The above definition of the various PSO parameters has been proposed as the standard PSO [54]. For this reason, this variation of PSO will be implemented as the overlaying meta optimiser in this research. The pseudo-code in Algorithm 1 below describes the structure of the PSO algorithm.

### 2.2.1 Velocity Update Equation Variations

The velocity update equation outlined by Equation 2.1 will be implemented in the PSO meta optimiser. Table 2.1 outlines the first set of the velocity update equations
Create \( N \) particles with random position and velocity

\[ \textbf{while} \ \text{Iteration} \ t < \text{Imax} \ \textbf{do} \]

\[ \textbf{for} \ \text{Particle} = 1 \ \text{to} \ N \ \textbf{do} \]

- Update personal best position
- Update neighbourhood best position
- Evaluate particle’s current position
- Update particle’s velocity
- Update particle’s position

\[ \textbf{end} \]

\[ \textbf{end} \]

Return best solution

\textbf{Algorithm 1: PSO Algorithm}

that will be evaluated.

The velocity update equations in Table 2.1 have been divided into three categories to better allow their performance to be analysed. Velocity equations 1, 2 & 3 are all very similar in structure and are considered to be standard. Velocity equations 4, 5, 6 & 7 all incorporate the worst locations of the problem space and are therefore grouped together. The final category of velocity update equations are non-standard velocity update equations, Velocity equations 8 to 13. Additional velocity update equations will be added later to test hybridized velocity update equations. The overall performance of each algorithm will be measured based on convergence speed, average fitness and fitness standard deviation.

The first and second velocity update equations in Table 2.1, are the most commonly used velocity update equations in the PSO literature. This first velocity equation implements a constriction factor \( \chi \) and has been shown to provide stable convergence with certain parameters outlined at the beginning of this section by Equation 2.2a. Although these parameter values provide stable convergence, applying meta optimisation to these values will also determine if these values provide the best performance. This will address the research question: Does a stable PSO convergence result in better performance? Velocity equation 2 utilizes the linearly decreasing inertia term \( \omega \).

The third velocity update equation uses an adaptive inertia weight. This \( \omega \) changes its value based on the success of particles on improving their fitness between iterations. Comparing this group of velocity equations will address research question 3, Does a constricted PSO perform better than a PSO with Inertia?

Velocity equation 4 is the first to incorporate worst locations. It utilizes the both the personal and group worst positions of the particle but not the best locations to dictate the particles movement. Velocity update equations 5 and 6 both utilize the best and worst locations. Velocity equation 6 utilizes the best and worst personal and
### Table 2.1: Velocity Update Equations

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Velocity Equation</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Constriction [76]</td>
<td>( \vec{v}_{i+1} = \chi (\vec{v}_i + r_1 \vec{c}_1 (\vec{p}_b - \vec{x}_i) + r_2 \vec{c}_2 (\vec{g}_b - \vec{x}_i)) )</td>
<td>( c_1, c_2, \chi )</td>
</tr>
<tr>
<td>2) Inertia [334]</td>
<td>( \vec{v}_{i+1} = \omega \vec{v}_i + r_1 \vec{c}_1 (\vec{p}_b - \vec{x}_i) + r_2 \vec{c}_2 (\vec{g}_b - \vec{x}_i) )</td>
<td>( c_1, c_2, \omega_{\text{start}}, \omega_{\text{end}} )</td>
</tr>
<tr>
<td>3) AIW PSO [278]</td>
<td>( \vec{v}_{i+1} = \omega \vec{v}_i + r_1 \vec{c}_1 (\vec{p}_b - \vec{x}_i) + r_2 \vec{c}_2 (\vec{g}_b - \vec{x}_i) )</td>
<td>( c_1, c_2, \omega_{\text{start}}, \omega_{\text{end}} )</td>
</tr>
<tr>
<td>4) PSOA [392]</td>
<td>( \vec{v}_{i+1} = \vec{v}_i + r_1 \vec{c}_1 (\vec{x}_r - \vec{p}_b) + r_2 \vec{c}_2 (\vec{x}_r - \vec{g}_b) )</td>
<td>( c_1, c_2 )</td>
</tr>
<tr>
<td>5) NPSO1 [329]</td>
<td>( \vec{v}_{i+1} = \omega \vec{v}_i + r_1 \vec{c}_1 (\vec{p}_b - \vec{x}_i) + r_2 \vec{c}_2 (\vec{p}_b - \vec{x}_i) + r_3 \vec{c}_3 (\vec{g}_b - \vec{x}_i) )</td>
<td>( c_1, c_2, c_3, \omega_{\text{start}}, \omega_{\text{end}} )</td>
</tr>
<tr>
<td>6) NPSO2 [188]</td>
<td>( \vec{v}_{i+1} = \omega \vec{v}_i + r_1 \vec{c}_1 (\vec{p}_b - \vec{x}_i) + r_2 \vec{c}_2 (\vec{g}_b - \vec{x}_i) + r_3 \vec{c}_3 (\vec{x}_r - \vec{p}_b) + r_4 \vec{c}_4 (\vec{x}_r - \vec{g}_b) )</td>
<td>( c_1, c_2, c_3, c_4, \omega_{\text{start}}, \omega_{\text{end}} )</td>
</tr>
<tr>
<td>7) PSO AWL [251]</td>
<td>( \vec{v}_{i+1} = \chi (\vec{v}_i + t_1 + t_2 + t_3 + t_4) )</td>
<td>( c_1, c_2, c_3, c_4, \chi )</td>
</tr>
<tr>
<td>8) MOL [297]</td>
<td>( \vec{v}_{i+1} = \omega \vec{v}_i + r_1 \vec{c}_1 (\vec{x}_r - \vec{x}_i) )</td>
<td>( c_1, \omega_{\text{start}}, \omega_{\text{end}} )</td>
</tr>
<tr>
<td>9) PSO CV [171]</td>
<td>( \vec{v}_{i+1} = \omega \vec{v}_i + r_1 \vec{c}_1 (\vec{p}_b - \vec{x}_i) + r_2 \vec{c}_2 (\vec{g}_b - \vec{x}_i) + \bar{r} )</td>
<td>( c_1, c_2, \omega_{\text{start}}, \omega_{\text{end}} )</td>
</tr>
<tr>
<td>10) AR PSO [309]</td>
<td>( \vec{v}_{i+1} = \omega \vec{v}_i + \text{dir}(r_1 \vec{c}_1 (\vec{p}_b - \vec{x}_i) + r_2 \vec{c}_2 (\vec{g}_b - \vec{x}_i)) )</td>
<td>( c_1, c_2, \omega_{\text{start}}, \omega_{\text{end}}, \text{dir} )</td>
</tr>
<tr>
<td>11) AR PSO2 [309]</td>
<td>( \vec{v}_{i+1} = \omega \vec{v}_i + \text{dir}(r_1 \vec{c}_1 (\vec{p}_b - \vec{x}_i) + r_2 \vec{c}_2 (\vec{g}_b - \vec{x}_i)) )</td>
<td>( c_1, c_2, \omega_{\text{start}}, \omega_{\text{end}}, \text{dir}, \text{dist} )</td>
</tr>
<tr>
<td>12) DPSO [390]</td>
<td>( \vec{v}_{i+1} = \omega \vec{v}_i + r_1 \vec{c}_1 (\vec{p}_b - \vec{x}_i) + r_2 \vec{c}_2 (\vec{g}_b - \vec{x}_i) )</td>
<td>( c_1, c_2, \omega_{\text{start}}, \omega_{\text{end}}, c_3, c_4 )</td>
</tr>
<tr>
<td>13) AV PSO [13]</td>
<td>( \vec{v}<em>{i+1} = \frac{\vec{p}</em>{\text{best}}}{\text{pDist}_{\text{best}}} )</td>
<td>( n )</td>
</tr>
</tbody>
</table>

Where \( \text{dir} = -1 \) if \( \text{dir} > 0 \text{div} < \text{dir}_{\text{low}} \), \( \text{dir} = 1 \) if \( \text{dir} < 0 \text{div} > \text{dir}_{\text{high}} \), \( \text{div} \) is the swarm diversity,

\[
\text{div} = \frac{1}{\sum_{j=1}^{N} \sum_{i=1}^{N} (\vec{g}_i - \vec{p}_j)^2, |S|} \text{ } |S| \text{ is the swarm size, } L \text{ is the longest diagonal, } N \text{ is the number of dimensions, } p \text{ is a particle’s position, } \vec{p} \text{ is the average position, } \text{dir} = 5 \times 10^{-6} \text{ and } \text{dir}_{\text{high}} = 0.25
\]

The first velocity equation in the final category of non-standard velocity update equations is the many optimising liaisons equation. This velocity equation only utilises the group best position and not a particle’s personal best position. A velocity control parameter is added in velocity equation 9 with the aim of giving particles extra velocity when they have converged. Velocity equations 10 and 11 are very similar, the only difference is that the high & low diversity thresholds will be parameters to be optimised in velocity equation 11. These velocity equations implement a repulsive force once the diversity of particles is low enough. The final two velocity equations are the only two PSO variations that alter the position update equation specified in group positions, while velocity equation 5 does not utilize the group worst position. A constriction factor is implemented in velocity equation 7 rather than an inertia term. The worst locations also have an inverse effect on the particles’ velocity in this equation.
2.3. Meta Optimisation

Meta optimisation refers to the application of one optimisation algorithm to the parameters of another optimisation algorithm that is optimising a problem or set of problems. Since the performance of many optimisation algorithms is subject to their parameters being correctly selected, meta optimisation provides a way of selecting good parameters that let the sub optimisation algorithm operate effectively. The pseudo-code in Algorithm 2 shows how the meta optimisation algorithm works when implemented with PSO. The meta optimisation algorithm generates a number of particles with random positions, i.e. parameter sets for the sub optimisation algorithm. Each meta particle will evaluate and update its position (sub swarm parameter set) for a predefined number of iterations. Each position is evaluated by averaging its cumulative performance over a number of problems for a number of runs. Each problem is optimised by creating a new PSO process and running it for a predetermined number of iterations. Ultimately the meta PSO optimiser will converge on what is hoped to be the optimum parameter set for the sub PSO optimiser.

2.3.1 Application of Meta Optimisation

In this research, the meta optimisation process is represented in Figure 2.1. Each of the velocity update equations in Table 2.1 will be inserted into the sub PSO optimiser. The argument exists that by implementing meta optimisation, one will then need to optimise the parameters of the meta optimisation algorithm using an additional optimisation algorithm which will then need to be optimised, and so on. To circumvent this issue the meta optimisation algorithm implemented in this research is the constricted PSO (Equation 2.1). The parameters for the constricted PSO variant are guaranteed to converge. This does not guarantee the best performance of the meta optimiser but it does mitigate the issue of selecting the meta optimiser’s parameters. Regardless of what meta optimisation algorithm is used, the issue of parameter
Create & randomly initialise $M$ meta particles

\[
\text{while } \text{metaIteration } mI < \text{metaImax} \text{ do}
\]

\[
\text{for } \text{metaParticle} = 1 \text{ to } M \text{ do}
\]

\[
\text{for } \text{run} = 1 \text{ to } R_{\text{max}} \text{ do}
\]

\[
\text{for } \text{problem} = 1 \text{ to } P_{\text{max}} \text{ do}
\]

Create & randomly initialise $N$ sub particles

\[
\text{while } \text{Iteration } t < \text{Imax} \text{ do}
\]

\[
\text{for } \text{Particle} = 1 \text{ to } N \text{ do}
\]

Update personal best position
Update neighbourhood best position
Evaluate particle’s current position
Update particle’s velocity
Update particle’s position

end

end

Update cumulativeFitness

end

Average cumulativeFitness over $R_{\text{max}}$ runs
Update personal best meta position
Update neighbourhood meta best position
Evaluate meta particle’s current position
Update meta particle’s velocity
Update meta particle’s position

end

Return best parameter set

**Algorithm 2:** Meta Optimisation Algorithm

selection will occur. By selecting the constricted PSO, at least it is known that the meta optimiser will converge and provide good performance.

Equation 2.1 is shown to have stable convergence with values $\chi \approx 0.72984$ & $c_1 = c_2 = 2.05$ and will not require parameter tuning. PSO was also selected as a meta optimiser because the problem of parameter selection is not a convex optimisation problem, therefore traditional gradient decent methods are not suited to it. The experimental parameters that will define each simulation are as follows: The meta optimising PSO will consist of 5 particles over 100 iterations. This equates to the evaluation of 500 combinations of parameters which should be sufficient given that the most parameters that need to be optimised by any velocity update equation is 6. The sub optimisation PSO will consist of 50 particles evaluating each problem for 3000 iterations. A larger number of particles and iterations is needed here as the problem spaces are much larger, typically 30 dimensions. The cumulative fitness achieved over all 8 problems is averaged over 10 statistical runs to ensure that any set of PSO parameters didn’t perform well/poorly by chance.
2.3.2 Problem Definition

Table 2.2 outlines each of the functions that will be used to test the performance of each velocity update equation, aside from the watershed problem which will be outlined later on. The performance of each algorithm is judged by a linear combination of its performance on each of the problems. Since the Watershed Management problem is a maximisation problem rather than a minimisation problem, each of the functions in Table 2.2 have been converted to maximisation problems.

2.4 Machine Learning

In Machine Learning research, algorithms can be divided into either online or offline learning [225]. Online learning algorithms refer to algorithms that continuously learn
Chapter 2. A Meta Optimisation Analysis of Particle Swarm Optimisation

Table 2.2: Benchmark Maximisation Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td>( f(x) = -\sum_{i=1}^{d} x_i^2 )</td>
</tr>
<tr>
<td>Rosenbrock</td>
<td>( f(x) = -\sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] )</td>
</tr>
<tr>
<td>Ackley</td>
<td>( f(x) = 20\exp\left(-0.2\sqrt{\sum_{i=1}^{d} x_i^2}\right) + \exp\left(\frac{1}{d}\sum_{i=1}^{d} \cos(cx_i)\right) - 20 - \exp(1) )</td>
</tr>
<tr>
<td>Griewank</td>
<td>( f(x) = -\sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 )</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>( f(x) = -10d - \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi x_i)] )</td>
</tr>
<tr>
<td>Schaffer</td>
<td>( f(x) = -0.5 - \frac{\sin\left(\sqrt{x_1^2 + x_2^2}\right) - 0.5}{\sqrt{x_1^2 + x_2^2} + 0.001} )</td>
</tr>
</tbody>
</table>

in their environment. In offline learning, there is a clear distinction between the training phase and the testing phase. The training phase consists of a fixed set of historic data, known as training data. This data is used for the offline learning algorithm to approximate some function. Once the initial training period is complete, the algorithm is finished learning. The testing period consists of introducing the algorithm to previously unseen data, completely separate from the training data, and evaluating its performance. In online learning, there is no distinction between training and test periods/data. This research will focus on offline learning, namely an Artificial Neural Network (ANN) trained with PSO.

2.4.1 Artificial Neural Networks

Artificial Neural Networks (ANN), is a field of study within Machine Learning that is inspired by the brain [47, 160]. ANNs are have been applied to a range of areas such as classification, regression, control, online and offline learning, robotics and function approximation. ANN consist of an input layer of units (commonly referred to as neurons), one or more hidden layer of neurons and an output layer. The network receives information in the form of a signal into the input layer. This signal is carried through the connected layers of neurons via weighted connections. The network then outputs the signal through the output layer. The most commonly used method used to train the network weights is the backpropagation method [163]. This method requires a set of labelled data and is therefore not suitable for this application as the optimum network outputs for each flow scenario are not known. This provides the motivation for this research and justifies the fifth contribution outlined in the introduction. Unsupervised learning only consists of unlabelled input data where the
targets are not known. This research consists of taking a set of unlabelled input data, in the form of flow scenarios for the Watershed Management problem, and use PSO to train the ANN in an unsupervised offline manner.

Information is fed into the network through the input layer of neurons (or units). This signal is then passed through the neurons in the subsequent hidden layers of the network until it is outputted from the final output layer. This process is commonly referred to as a forward pass. Figure 2.2 shows the structure of a multi-layer ANN. By conducting parameter sweeps, it was found that a network configuration of 1 hidden layer with 4 neurons provided the good performance while still keeping the computation time relatively low. This configuration consists of 28 network weights which must be optimised. It is possible to implement PSO in a manner to also evolve the configuration of the neural network. This is however out of the scope of this research. The primary concern when applying meta optimisation to the PSO in this paper is to find the optimum parameters of each velocity update equation. By also applying the meta optimiser to the neural network configuration, one would run the risk of failing to identify good velocity equation parameters due to poor network configuration. Therefore it is thought to be advantageous in this instance to keep the network configuration constant.

As the signal is propagated through the network, its strength is weighted as it passed between each layer of neurons. A neuron in any given layer will have as input, the sum of the weighted outputs from the previous layer of neurons.
\[ v_j = \sum_{i=1}^{N} w_{i,j} a_i \]  

(2.3)

Where \( v_j \) is the input to a neuron in the \( j^{th} \) layer, layer \( i \) is the preceding layer to \( j \) that contains \( N \) neurons, each neuron in layer \( i \) has output \( a_i \) and each of these output signals are weighted by the value \( w_{i,j} \) as they are passed to each neuron in layer \( j \).

Each neuron \( a_i \) outputs a value between 0 and 1. This output value is determined by the activation function of the neuron. The most commonly used activation function is the sigmoid (or logistic) function:

\[ a_j = \frac{1}{1 + \exp(-v_j)} \]  

(2.4)

2.4.2 Related Work

Training neural networks can also be viewed as optimising a set of weights to minimise a cost function. There are many methods used to do this. As previously mentioned, the gradient descent and back propagation algorithms are the most commonly used methods [163]. Reinforcement Learning (RL) algorithms commonly use neural networks to handle continuous problem spaces [53], e.g. Temporal Difference learning [353]. Evolutionary computing has also been used to train neural networks. This is known as Neuroevolution [122]. Neuroevolution methods incorporate some form of evolutionary algorithm, e.g. the NEAT algorithm which evolves neural networks using genetic algorithms [347]. Recent work has also applied Differential Evolution to the task of neural network weight optimisation [242, 103, 240] Particle Swarm Optimisation has also been applied to train neural networks for classification tasks [297] and control problems [194]. This research builds on these previous methods by applying them to the Watershed Management problem.

2.4.3 Application to Watershed Management

In the case of training a neural network function approximator for the Watershed Management problem, a particle’s position will correspond to the network weights. The 3 network inputs will correspond to the current state of the river \([q_1, q_2, s]\). The
4 outputs of the network correspond to the 4 directly controlled variables of the Watershed problem, i.e. the 4 interested parties seeking to withdraw water. The fitness of the current network weights configuration (or particle position) is calculated as the sum of the individual fitness evaluations for each training state over all training states \([Q_1, Q_2, S]\). The fitness of each state is calculated using Equation 2.11. Each PSO particle will then use the fitness over all states as a means to determine how good the current set of network weights is.

### 2.5 Watershed Problem

As mentioned in the introduction section, the Watershed problem is a resource management problem which consists of multiple interested parties. Each of these parties are withdrawing water from a common and finite supply for their own purposes. This problem has multiple objectives and constraints, and consists of continuous variables. However this problem will be treated as a single objective problem as only the overall system performance is of interest. There are 6 variables which must be optimised, 4 of which are controlled directly, the remaining 2 are reactive variables which are indirectly optimised. The 4 direct variables are the water withdrawn from the river for municipal and industrial use in the city \((x_1)\), the water withdrawn for the irrigation of farms \((x_4 \text{ and } x_6)\) and finally the water released from a dam for hydro power generation \((x_2)\). The 2 reactive variables are the water available for the ecosystems \((x_3 \text{ and } x_5)\). Each of these variables must be selected to maximise a series of objective functions representing the benefits obtained from the water by the various interested parties.

The benefit of the water withdrawn for each interested party is represented by the following function:

\[
f_i(x_i) = a_i x_i^2 + b_i x_i + c_i \quad (2.5)
\]

Where \(a_i, b_i\) and \(c_i\) are dimensionless constants corresponding to each party [394]. Their values are highlighted in the following table, along with the values for \(\alpha_i\) which represents the minimum values for \(x_i\) in \(L^3\), where \(i\) represents each objective.

Since the indirect variables, \(x_3\) and \(x_5\), are not directly controlled, they must be
Table 2.3: Watershed Constants

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value (L^3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1</td>
<td>−0.20</td>
<td>b_1</td>
<td>6</td>
<td>c_1</td>
<td>−5</td>
<td>α_1</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_2</td>
<td>−0.06</td>
<td>b_2</td>
<td>2.5</td>
<td>c_2</td>
<td>0</td>
<td>α_2</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_3</td>
<td>−0.29</td>
<td>b_3</td>
<td>6.28</td>
<td>c_3</td>
<td>−3</td>
<td>α_3</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_4</td>
<td>−0.13</td>
<td>b_4</td>
<td>6</td>
<td>c_4</td>
<td>−6</td>
<td>α_4</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_5</td>
<td>−0.056</td>
<td>b_5</td>
<td>3.74</td>
<td>c_5</td>
<td>−23</td>
<td>α_5</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_6</td>
<td>−0.15</td>
<td>b_6</td>
<td>7.6</td>
<td>c_6</td>
<td>−15</td>
<td>α_6</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

calculated using the following equations:

\[
x_3 = Q_2 - x_4 \quad (2.6a)
\]

\[
x_5 = x_2 + x_3 - x_6 \quad (2.6b)
\]

Where \( Q_2 \) is the monthly tributary inflow in \( L^3 \). The values for \( Q_2 \) will be outlined later. The problem variables \( x_i \) are is subject to the following constraints which restrict their potential values.

\[
\alpha_1 - x_1 \leq 0 \quad (2.7a)
\]

\[
\alpha_2 - Q_1 + x_1 \leq 0 \quad (2.7b)
\]

\[
x_2 - S - Q_1 + x_1 \leq 0 \quad (2.7c)
\]

\[
\alpha_4 - x_3 \leq 0 \quad (2.7d)
\]

\[
\alpha_3 - x_4 \leq 0 \quad (2.7e)
\]

\[
\alpha_4 - Q_2 + x_4 \leq 0 \quad (2.7f)
\]

\[
\alpha_6 - x_5 \leq 0 \quad (2.7g)
\]

\[
\alpha_5 - x_6 \leq 0 \quad (2.7h)
\]

\[
\alpha_6 - x_2 - x_3 + x_6 \leq 0 \quad (2.7i)
\]

Where \( S \) represents the storage capacity of the dam in \( L^3 \) while both \( Q_1 \) and \( Q_2 \) represents the monthly inflows of water into the mainstream and tributary in \( L^3 \).

The Watershed problem evaluated in this paper will consist of a total of 150 flow states for the river. This is divided up into \( \frac{2}{3} \) training data (100 states) and \( \frac{1}{3} \) testing
The Watershed problem can be formulated as optimising the following equation subject to the constraints highlighted in Equation 2.7.

\[ F(x) = \max \sum_{i=1}^{6} f_i(x_i) \]  

(2.8)

The Watershed problem is graphically illustrated in Figure 2.3 from the research of Yang et al. [394].

2.5.1 Boundary Definition

In order to address the Watershed Management problem, the maximum and minimum possible values for each direct variable must first be explicitly defined. The ranges for each of the directly optimised variables are defined as follows based on Equation 2.7:
\begin{align*}
\alpha_1 & \leq x_1 \leq Q_1 - \alpha_2 & (2.9a) \\
\alpha_3 & \leq x_4 \leq Q_2 - \alpha_4 & (2.9b) \\
0 & \leq x_2 \leq S + Q_1 - \alpha_1 & (2.9c) \\
\alpha_5 & \leq x_6 \leq S + Q_1 + Q_2 - \alpha_1 - \alpha_3 - \alpha_6 & (2.9d)
\end{align*}

### 2.5.2 Constraint Handling

Any violations of the constraints highlighted in Equation 2.7 will be handled using the static penalty method [340]. This penalty function will be incorporated into the fitness of any given solution. The penalty function is defined below:

\[
f_p = \sum_{i=1}^{N} C(|h_i + 1|\delta_i) \tag{2.10}
\]

Where \( N = 9 \) is the total number of constraints handled using this method per flow scenario, \( C = 10E2 \) is the violation constant, \( h_i \) is the violation amount of each constraint and \( \delta = 0 \) if there is no violation for a particular constraint and \( \delta = 1 \) if a constraint is violated. The violation constant \( C = 10E2 \) was selected so that any solution which violates a constraint will have a significantly lower fitness than acceptable solutions. Lower \( C \) values were evaluated but caused each algorithm to converge on infeasible solutions.

### 2.5.3 Fitness Function

In order to gauge the suitability of a given solution, a fitness function must first be defined. As previously mentioned, the fitness function will be based on the overall performance of the system based on the objective function in Equation 2.8 and the penalty function from Equation 2.10. These two equations are combined to give the following fitness function which is to be maximised:

\[
F = \sum_{i=1}^{6} f_i(x_i) - \sum_{j=1}^{N} C(|h_j + 1|\delta_j) \tag{2.11}
\]
2.6 Experimental Results

In this section, the results of each of the experiments will be presented and discussed. The performance of each velocity update equation will be gauged based on its convergence speed, best fitness and consistency. These will be judged for both the cumulative fitness over all 8 problems and for each problem individually. Any performance comparisons between two algorithms will be done using the two tailed t-test with a significance level of 5%. This will ensure that any differences in performance are statistically significant.

2.6.1 Constriction vs Inertia

Table 2.5 and Figure 2.4 shows the cumulative fitness convergence for the constricted, linear decreasing inertia and adaptive inertia velocity update equations. These results clearly show that the linear decreasing inertia provides better performance than both constriction and adaptive inertia weight, constriction being the worst of the three. This gives a clear answer to the third research question: stable convergence does not guarantee optimum performance and a linearly decreasing inertia outperforms a constricted PSO.

![Velocity Update Equation: Constriction vs Inertia](image)

Figure 2.4: Convergence of Velocity Update Equations using Constriction & Inertia
This graph illustrates the average cumulative fitness convergence of the standard constricted PSO, PSO with linear inertia and adaptive inertia PSO over all 8 problems.
2.6.2 Avoidance Strategies

Since the results from the previous section show that a linearly decreasing inertia offers superior performance, two additional modified versions of the PSOA and PSOAWL were also tested to include a linearly decreasing inertia. These velocity equations can be seen in Table 2.4, numbered velocity equations 14 & 15. The parameter sets to be optimised for each of the PSOA2 and PSOAWL2 are \([c_1, c_2, \omega_{\text{start}}, \omega_{\text{end}}]\) and \([c_1, c_2, c_3, \omega_{\text{start}}, \omega_{\text{end}}]\) respectively. These two velocity update equations were included to ensure that a fair comparison was made amongst each of the velocity update equations.

The performance results of each of the avoidance velocity update equations can be seen in Table 2.5 and Figure 2.5. The two best performing velocity equations here are the PSOAWL2 and the NPSO2. Each of these variants converge to almost the same solution fitness. The PSOAWL2 performs statistically better than the NPSO2. The third best performing algorithm is the PSOAWL with constriction followed by the NPSO1. Both of these algorithms converge much faster than the PSOAWL2 and NPSO1, however they converge on a significantly worst solution. The PSOAWL2 implemented with a linear decreasing inertia performs much better than the PSOAWL implemented with a constriction. This reaffirms the results from the previous section where the linear decreasing inertia outperformed the constriction value. Both variants of the PSOA perform worst. This was expected as each of these variants only avoid the worst locations and do not converge on the best locations. Each of these velocity update equations produce solutions far worse than those previously mentioned and therefore do not appear in Figure 2.5. These velocity update equations were included for comparative purposes.

2.6.3 Non-Standard Velocity Update Equations

As illustrated in Figure 2.6 and highlighted in Table 2.5, ARPSO2 and ARPSO1 are the 2 best performing velocity update equations. These are followed closely by the DPSO. The difference between these two variants of the ARPSO is the parameters being optimised. The ARPSO1 uses the default values for the critical upper and lower diversity levels, \(d_{\text{low}} = 5 \times 10^{-6}\) and \(d_{\text{high}} = 0.25\) [309]. The meta optimiser for the ARPSO2 found the optimum values to be \(d_{\text{low}} = 0.0570\) and \(d_{\text{high}} = 0.0123\) (rounded to 4 decimal places). Interestingly the meta optimiser actually found the optimum \(d_{\text{high}}\) value to be lower than the optimum \(d_{\text{low}}\) value. This would lead to the
2.6. Experimental Results

Figure 2.5: Convergence of Avoidance Velocity Update Equations This graph illustrates the average cumulative fitness convergence of the 6 PSO variants that utilize avoidance strategies over all 8 problems.

situation where the parameter \( \text{dir} \) is alternating between \(-1\) and \(1\) at each iteration.

This will be discussed later in more detail in Section 2.6.6. The ARPSO1 offers faster convergence than the ARPSO2, however the ARPSO2 converges to a better solution and has a smaller standard deviation meaning that it is more consistent. MOL is the next best performing algorithm which converges quickly but converges to a suboptimal solution.

The second worst performing velocity update equation is the AVPSO. This is an interesting velocity update equation however as this variant has only one parameter that needs to be optimised. The AVPSO is also very inconsistent with a high standard deviation. The worst performing of the non-standard velocity update equations is the PSOCV. This variant does not provide fast convergence as seen in Figure 2.6. The velocity control in this equation is similar in philosophy to the ARPSO but performs statistically worse. The advantage of implementing the PSOCV over the ARPSO however is that the PSOCV does not require calculating the swarm diversity at each iteration. This variant converges very slowly to a poor solution.

2.6.4 Hybrid Velocity Update Equations

In this section, a number of hybrid velocity update equations will be proposed based on the performance of the algorithms in the previous sections. Since essentially all
The results of these hybrid velocity equations are displayed in Table 2.5 and in Figure 2.7. The AR PSOAWL is the best performing hybrid velocity update equation, followed by the D PSOAWL. The AR PSOAWL performs statistically better than the D PSOAWL and also has a very small standard deviation meaning that its performance is consistent.

As a result of this, hybrid velocity update equations were created based on the best performing of the standard velocity update equations. Table 2.4 displays the 5 hybrid velocity update equations which were evaluated, numbered 16 - 20.

Table 2.4: Modified Velocity Update Equations

![Graph of Fitness vs Iteration](image)

Figure 2.6: Convergence of Non-Standard Velocity Update Equations. This graph illustrates the average cumulative fitness convergence of the 6 non-standard PSO velocity equations over all 8 problems.
was consistent across the 10 statistical runs. This is a highly desirable feature of an optimisation algorithm. This velocity update also converges very quickly, converging to a nearly optimum solution after 1000 iterations. The reason for this fast convergence to a near optimum solution is thought to be due to the enhanced exploration provided by the attractive/repulsive part of the velocity update equation and the enhanced convergence provided by avoiding the worst locations. The best 2 hybrid velocity update equations were variants of the PSO AWL while the worst and 3rd worst were variants of the NPSO 2.

![Figure 2.7: Convergence of Hybrid Velocity Update Equations](image)

This graph illustrates the average cumulative fitness convergence of the 5 hybrid PSO velocity update equations over all 8 problems.

### 2.6.5 Overall Comparison

The convergence graphs in Figure 2.8 displays the convergence of the best 8 performing velocity update equations. The best two performing algorithms are linearly changing inertia and the AR PSOAWL. When compared the two tailed t-test these two algorithms produce solutions that are statistically equal.

There are advantages and disadvantages with each velocity update equation. The linearly changing inertia PSO is computationally cheaper and simpler to implement. This would make the linearly decreasing inertia PSO ideal for some problems where limited computational resources are available. Conversely the AR PSOAWL converges much faster. This makes the algorithm much more suitable for problems where a solution is needed rapidly.
It is also worth noting the performance of the Linear Decreasing Inertia PSO with MOL. MOL was originally proposed as a simplified version of the standard PSO and it was found to perform better than the standard PSO with a static inertia and therefore was claimed to be better than the standard PSO [297]. The research conducted in this paper demonstrates that this is not necessarily the case. MOL is the 12th best performing algorithm and is statistically outperformed by more complex velocity update equations. However it is among the least computationally expensive velocity update equations of those evaluated.

Each of the velocity update equations evaluated here were reported to be superior to the standard linearly decreasing inertia. This research demonstrates that this is not strictly true. This research has confirmed that negative inertia values can provide increased performance as previously established [297]. This research has also established that a linear increasing inertia can provide superior performance. However when each of these velocity update equations were compared to the standard linear changing inertia, negative or increasing inertia values were never considered. This is because without the use of meta optimisation, a practitioner implementing PSO would never consider to implement such a velocity update equation. It was found that the PSOAWL2 and AR PSOAWL perform statistically equal to the linear changing inertia. Since the linear changing inertia is much simpler to implement and requires less computational time, this basic velocity update equation would still be the preferred choice. However the AR PSOAWL does converge faster, which would make this algorithm the obvious choice if a solution is needed in fewer iterations.

There is also the possibility that the meta optimiser has not fully converged on the optimum parameters for the more complex velocity update equations. The linear changing inertia has 4 parameters that must be optimised while the AR NPSO has 8 parameters. The meta optimiser was implemented with same number of iterations (50) and particles (10) for each velocity update equation regardless of the number of parameters. It was not possible to run the meta optimiser for more iterations due to the large computational cost required to do so. In the future, it will be possible to conduct larger meta optimisation analysis as computers become faster. Running larger experiments now would require more computational resources than is available. The results presented in this paper required 12 days to produce when implemented in parallel using a separate thread for each velocity update equation. 4 threads were running at any one time. This was simply due to the scale of the
2.6. Experimental Results

Experiments conducted in the research rather than the efficiency of the PSO algorithm. To evaluate 20 variants of any optimisation algorithm over 8 problem spaces using meta optimisation will require a considerable amount of time.

Figure 2.8: Convergence of Best 8 Velocity Update Equations This graph illustrates the average cumulative fitness convergence of the 8 best performing PSO velocity update equations over all 8 problems.

2.6.6 Optimum PSO Parameters

The parameter boundaries for the meta optimiser are as follows:

\[ \chi \in [-2, 2], \ \omega_{start,end} \in [-2, 2], \ c_{1,2,3,4} \in [-4, 4], \ d_{high,low} \in [0, 1], \ c_{v,l} \in [0, 1], \ n \in [-10, 10], \]

These reflect the maximum and minimum values that the parameters outlined in Tables 2.1 and 2.4 can obtain. It is possible to apply meta optimisation to other PSO parameters outside of the velocity update equation, such as the swarm size and number of iterations. This was not implemented in this research however for the same reasons previously stated for not applying meta optimisation to the neural network configuration. The goal is to keep everything else consistent other than the velocity update equation. The optimum parameters found by the meta optimiser for each velocity update equation can be seen in Table 2.5. All of the values in this table are rounded to four decimal places. It should be noted that velocity clamping was implemented for each simulation. This was to ensure that the velocity of the particle did not exceed the length of the problem in each dimension.
Table 2.5: Optimum Parameter Settings & Fitness Results

<table>
<thead>
<tr>
<th>Velocity Equation</th>
<th>Average</th>
<th>StdDev</th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_3$</th>
<th>$c_4$</th>
<th>$\chi$</th>
<th>$w_{\text{start}}$</th>
<th>$w_{\text{end}}$</th>
<th>$d_{\text{low}}$</th>
<th>$d_{\text{high}}$</th>
<th>$\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constriction</td>
<td>23808.5182</td>
<td>6.736</td>
<td>0.4790</td>
<td>3.9911</td>
<td>-</td>
<td>-</td>
<td>0.6952</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Linear Inertia</td>
<td>23828.2486</td>
<td>5.6245</td>
<td>1.3518</td>
<td>2.3349</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-1.6980</td>
<td>1.0446</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AIWPSO</td>
<td>23812.5791</td>
<td>8.486</td>
<td>0.1419</td>
<td>3.5428</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-0.2897</td>
<td>0.5107</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PSOA</td>
<td>19239.9814</td>
<td>1938.4106</td>
<td>-0.6689</td>
<td>-3.1511</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.1945</td>
<td>-0.8163</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NPSO1</td>
<td>23789.2629</td>
<td>9.7344</td>
<td>2.6440</td>
<td>0.0016</td>
<td>0.7050</td>
<td>-</td>
<td>-</td>
<td>0.5862</td>
<td>-1.0542</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NPSO2</td>
<td>23820.1917</td>
<td>4.6547</td>
<td>1.4219</td>
<td>0.9686</td>
<td>-0.0001</td>
<td>0.7510</td>
<td>-</td>
<td>-1.2431</td>
<td>1.2678</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PSOAWL</td>
<td>23831.4837</td>
<td>9.1318</td>
<td>1.4900</td>
<td>2.7556</td>
<td>-0.3557</td>
<td>2.8016</td>
<td>0.4629</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PSOA2</td>
<td>19239.9814</td>
<td>1938.4106</td>
<td>-0.6689</td>
<td>-3.1511</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.1945</td>
<td>-0.8163</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AIWPSO2</td>
<td>23825.3919</td>
<td>3.6048</td>
<td>1.6220</td>
<td>2.4593</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-1.4442</td>
<td>0.8631</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MOL</td>
<td>23749.1889</td>
<td>47.6048</td>
<td>3.1569</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.0260</td>
<td>-1.2086</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PSOCV</td>
<td>23401.1286</td>
<td>40.9073</td>
<td>1.0876</td>
<td>3.1321</td>
<td>-0.0017</td>
<td>-</td>
<td>-</td>
<td>0.1860</td>
<td>-1.5133</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ARPSO</td>
<td>23816.0695</td>
<td>10.3410</td>
<td>1.6283</td>
<td>1.8452</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.4836</td>
<td>-0.1581</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ARPSO2</td>
<td>23822.6522</td>
<td>5.4883</td>
<td>0.8250</td>
<td>1.6549</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-1.2192</td>
<td>0.0210</td>
<td>0.0570</td>
<td>0.0123</td>
<td>-</td>
</tr>
<tr>
<td>DFPSO</td>
<td>23884.4577</td>
<td>6.8192</td>
<td>0.9830</td>
<td>2.1932</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-1.2585</td>
<td>1.5159</td>
<td>-</td>
<td>-</td>
<td>0.0013</td>
</tr>
<tr>
<td>AVPSO</td>
<td>23550.5377</td>
<td>177.4378</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AR PSOAWL</td>
<td>23826.8770</td>
<td>4.922</td>
<td>0.6635</td>
<td>0.9234</td>
<td>-0.2334</td>
<td>-</td>
<td>-</td>
<td>-1.1235</td>
<td>0.5260</td>
<td>0.1610</td>
<td>0.0087</td>
<td>-</td>
</tr>
<tr>
<td>AR PSOAWL</td>
<td>23748.5109</td>
<td>13.8534</td>
<td>1.4513</td>
<td>1.6637</td>
<td>-1.4938</td>
<td>-</td>
<td>-</td>
<td>1.0247</td>
<td>-0.0040</td>
<td>-</td>
<td>0.0092</td>
<td>0.0124</td>
</tr>
<tr>
<td>AR NPSO</td>
<td>23657.5277</td>
<td>21.1172</td>
<td>0.6697</td>
<td>3.3012</td>
<td>-0.0376</td>
<td>-1.4002</td>
<td>-</td>
<td>0.2529</td>
<td>-1.2638</td>
<td>0.2390</td>
<td>0.8096</td>
<td>-</td>
</tr>
<tr>
<td>D NPSO</td>
<td>23705.5609</td>
<td>29.7061</td>
<td>1.6866</td>
<td>0.7249</td>
<td>0.0061</td>
<td>0.2196</td>
<td>-</td>
<td>-1.5168</td>
<td>1.6145</td>
<td>-</td>
<td>0.0033</td>
<td>0.0031</td>
</tr>
<tr>
<td>AR DPSO</td>
<td>23567.1871</td>
<td>20.0029</td>
<td>1.0394</td>
<td>0.6239</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.1151</td>
<td>0.8586</td>
<td>0.7689</td>
<td>0.4601</td>
<td>0.0056</td>
</tr>
</tbody>
</table>

As seen in the results of other studies [297], the optimum parameter settings found by the meta optimiser are drastically different from those commonly used in the PSO literature. The optimum parameters found for the constricted PSO are very far from those proven to provide stable convergence. The meta optimiser found $\chi = 0.6952$ with $c_1 = 0.4790$, $c_2 = 3.9911$ gave the best performance. It was mathematically proven that $\chi = 0.72984$ with $c_1 = c_2 = 2.05$ provide stable convergence [76]. The meta optimiser did not find these values to give the best performance.

There are two interesting observations with regards to the optimum inertia values found. The first observation is that in many cases, the meta optimiser found that a negative inertia value leads to better performance for many of the velocity update equations evaluated. It is very interesting that the meta optimiser found a negative $\omega$ to be optimal. The vast majority of research into PSO does not consider implementing a negative $\omega$. This would have the effect of alternating the direction of the particles’ velocity between iteration, possibly leading to greater exploration. For every velocity update equation implemented with an inertia term, the meta optimiser found that an $\omega < 0$ at some stage of the optimisation process gives the best performance. This would be a potential avenue of future research to further investigate why this is the case.

The second observation is that in many cases, a linearly increasing inertia value was found to be optimal. It would therefore be more accurate to refer to the inertia term as a linearly changing inertia rather than a linearly decreasing inertia. In the velocity update equations where an increasing inertia was found to be optimal, typically the
optimum found initial inertia value is $\omega < -1$. This would cause the particles to initially move with a lot of velocity changing direction at each iteration, causing the particles to explore the problem space. These inertia terms then always increase to final values where $\omega > 0$, in some cases where $\omega > 1$. In situations where $1 > \omega_{\text{end}} > 0$, the particles will converge on the best known location as is normal with PSO. This is the case with the AIWPSO, PSOAWL2, ARPSO2 and AR PSOAWL. For other velocity update equations where $\omega_{\text{end}} > 1$ this is not the case. It is likely that the particles will have converged to some solution while $\omega$ changes between 0 and 1. However when $\omega$ increases above 1 again, the particles would receive more velocity and begin exploring again. It is thought that this would aid the performance of the PSO algorithm by enabling the particles to perform a local search after they have converged. This is the case for the standard linear changing inertia PSO, NPSO2, DPSO and DNPSO.

In many cases it is still found that a linearly decreasing inertia is still optimal. This is the case for the NPSO1, PSOA2, MOL, PSOCV, ARPSO, D PSOAWL and AR NPSO. In many of these velocity update equations $\omega_{\text{end}} < -1$. This would serve the purpose of performing a local search after the particles have converged.

It is also worth noting that each of these parameter sets highlighted in Table 2.5 have small standard deviations relative to their average fitness. This indicates that they give consistent performance. This is not the case for the PSOA or PSOA2 as this velocity update equation was included for comparative purposes and does not perform well.

The meta optimiser found interesting upper and lower diversity thresholds for the ARPSO. In its first proposal, the ARPSO found that $d_{\text{low}} = 5 \times 10^{-6}$ and $d_{\text{high}} = 0.25$ gave the best performance [309]. Table 2.5 however shows that the optimal diversity thresholds are very different from these values. Most notably that the $d_{\text{low}} > d_{\text{high}}$. The optimal values found for the ARPSO2 are $d_{\text{low}} = 0.0570$ and $d_{\text{high}} = 0.0123$. As the particles converge, when the diversity gets to a point between these two values, the value for $\text{dir}$ would alternate between -1 and 1 until the current diversity moves out of this range. This could happen if the inertia was a small enough value. These values for the diversity thresholds are not what the authors had in mind in the original proposal of the ARPSO. The meta optimiser however found that these values produce a better solution and are more consistent. However these two variants perform statistically the same due to the relatively large standard deviation of the ARPSO.
The optimum $c_v$ and $c_l$ terms for the DPSO were found to be very small. This is unsurprising as giving the particles a random velocity and position is intended to increase their exploration. This is only supposed to happen a small portion of the time because the algorithm still needs to converge. Table 2.5 shows that for each variant of the DPSO, the assignment of a random velocity or position occurs less than 1% of the time.

The optimum values for each of the acceleration coefficients ($c$) vary for each velocity update equation. For each of the standard velocity update equations, a stronger influence from the global best position provides the best performance. Typical $c \approx 2$ is used in the literature. In many cases these $c$ values were far larger or smaller than 2. In all cases $c > 0$ for the global and personal best position portions of the velocity update equations. In some velocity update equations where worst locations influence the motion of the particles, the $c$ values relating to the velocity received from the worst locations are negative. It is thought that the reason for this is to aid exploration. With regards to the PSO CV, the $c_3$ term which relates to the additional velocity control term is almost 0. This indicates that the additional control velocity term should not have a strong influence on the particles’ motion. The results presented in Table 2.5 clearly show that this small influence from the velocity control term has a very adverse effect on the convergence of the particles when compared to the standard linearly changing inertia PSO. The velocity control term $(\frac{c_3 r v^2}{v^2})$ within the PSO CV velocity update equation could be an avenue for future research. When $v^2$ is large and the particles are exploring, the algorithm would operate as expected as the velocity control term would be very small and insignificant. However as $v^2$ approaches 0, the $c_3 r v^2$ term becomes much more significant. The problem with this is that the value of $c_3$ then needs to be tailored for each problem because the velocity control term does not possess any terms that can scale the additional velocity to the current problem being optimised. This lack of generalisation is a fundamental problem with this velocity update equation in its current form and would be worth investigating.

In summary, it is well known that the performance of the PSO algorithm is heavily reliant on the parameters selected. This is particularly true for the more complex hybrid velocity update equations. This results section has presented some very important results: 1) Negative inertia terms should be considered when implementing a PSO algorithm. 2) An increasing inertia should be considered as well as a decreasing inertia term. 3) The diversity thresholds of the ARPSO should be the subject of fur-
2.6. Experimental Results

This research demonstrates that very unorthodox diversity thresholds can perform well. 4) With regards to the constricted PSO, stable convergence does not equate to optimum performance.

2.6.7 Individual Problem Performance

The performance results of each velocity update equation on each individual problem are presented in Table 2.6. The most interesting observation from Table 2.6 is the performance of the AV PSO. When evaluating the performance of the AV PSO in Table 2.5, it performs 4th worst of all velocity update equations. The results presented in Table 2.5 are a linear combination of the fitness of each individual problem performances. This is also the fitness that the meta optimiser uses to select the optimum parameter set for each velocity update equation. The results presented in Table 2.6 however reveal that the AV PSO performs best on 6 out of the 8 problems evaluated. The AV PSO performs best on all of the benchmark problems except for the Rosenbrock function. The AR PSO performs best on the Rosenbrock function while the AR PSOAWL performs best for the Watershed ANN training. The linear changing inertia does not perform best on any one of the functions evaluated. The size of the Watershed training problem (28 weights) is very similar to many of the benchmark functions, most of which have 30 dimensions except for the Schaffer 2D and Griewank 10D functions.

The fact that the AV PSO performs best on 6 benchmark functions but performs 3rd worst on the Watershed training problem is indicative that the Watershed training problem requires an optimisation algorithm with different characteristics than the benchmark problems do. It is thought that the Watershed training problem requires much more exploration than the benchmark problems. The AV PSO performs best on the Sphere, Griewank 10D and Schaffer 2D. All of these problems are much simpler than the Watershed training problem. It would therefore be advantageous of an optimisation algorithm to converge quickly for these problems. The AV PSO is also unique when compared to the other velocity update equations tested as it only possess one parameter that must be optimised in its velocity update equation. This is clearly of benefit for the AV PSO when optimising simpler problems. This design simplicity is also an issue for more complex problems such as the Watershed training problem and the Rosenbrock function because the simple velocity update equation lacks the capability for its parameters to be selected for better exploration.
The impressive performance of the AV PSO on the benchmark functions demonstrates that a linear combination of each problem fitness should not be the only measure of performance used to compare algorithms. It should also be highlighted with regards to the Watershed neural network training, that a good fitness score for the training data does not equate to good performance on test data. This will be covered in more detail in the next section.

In terms of the computational cost of each velocity update equation, there was little variation in their computational requirements. In general the simpler velocity update equations were computationally cheaper, e.g. MOL, constricted PSO and PSOA. These variations in computational costs were minimal however when compared to the significant differences in the average fitness results.

2.6.8 Watershed Test Data

This final results section will look at the performance of each neural network function approximator on independent test data. This will be done for each of the 20 velocity update equations. As previously stated, these neural networks were trained using the set of training data consisting of 100 training cases outlined in the appendix. After the initial training period, the neural networks were then tested on a completely separate set of test data consisting of 50 test cases, also outlined in the appendix. Table 2.7 displays the test data results of each neural network trained with a different velocity update equation. The fitness result is a cumulative fitness over all 50 test cases, referring to the cumulative utility of the water for all interested parties for 50 different river flow scenarios. The fitness values have been rounded to two decimal places.

The results in Table 2.7 illustrate a key issue in Machine Learning research, the problem of over training. When evaluated using the test data, the best 5 performing neural networks are those trained with:

1. PSOAWL2
2. Linear Changing Inertia PSO
3. AR PSO1
4. DPSO
Table 2.6: Individual Problem Average Fitness & Standard Deviation. This table presents the average fitness and fitness standard deviation of each velocity update equation for all 8 problem spaces.

<table>
<thead>
<tr>
<th>Velocity Equation</th>
<th>Watershed</th>
<th>Sphere</th>
<th>Rosenbrock</th>
<th>Ackley</th>
<th>Griewank</th>
<th>Rastrigin</th>
<th>Schaffer2D</th>
<th>Griewank10D</th>
</tr>
</thead>
<tbody>
<tr>
<td>constriction</td>
<td>2.384E+04</td>
<td>4.508E+00</td>
<td>3.194E-09</td>
<td>2.404E+01</td>
<td>3.890E+00</td>
<td>3.854E+04</td>
<td>1.798E-04</td>
<td>1.426E+02</td>
</tr>
<tr>
<td>linear inertia</td>
<td>2.385E+04</td>
<td>5.435E+00</td>
<td>1.361E-16</td>
<td>1.692E-16</td>
<td>2.363E+01</td>
<td>2.487E-01</td>
<td>7.523E-08</td>
<td>9.828E-08</td>
</tr>
<tr>
<td>AW PSO</td>
<td>2.384E+04</td>
<td>7.922E+00</td>
<td>2.613E-06</td>
<td>2.282E-06</td>
<td>6.431E+00</td>
<td>6.732E+00</td>
<td>3.280E-03</td>
<td>3.058E-02</td>
</tr>
<tr>
<td>PSOA</td>
<td>1.947E+04</td>
<td>1.481E+03</td>
<td>6.572E+01</td>
<td>6.811E+00</td>
<td>3.220E+02</td>
<td>1.847E+01</td>
<td>3.659E-01</td>
<td>2.437E-01</td>
</tr>
<tr>
<td>NPSO1</td>
<td>2.584E+04</td>
<td>8.425E+00</td>
<td>4.790E-04</td>
<td>5.486E-05</td>
<td>2.493E-01</td>
<td>1.914E+00</td>
<td>2.317E-02</td>
<td>2.138E+01</td>
</tr>
<tr>
<td>NPSO2</td>
<td>2.355E+04</td>
<td>4.385E+00</td>
<td>1.438E+05</td>
<td>2.324E-06</td>
<td>2.678E+01</td>
<td>1.973E-01</td>
<td>5.278E-01</td>
<td>1.889E+02</td>
</tr>
<tr>
<td>PSOAWL</td>
<td>2.384E+04</td>
<td>7.648E+00</td>
<td>3.657E+11</td>
<td>2.809E-11</td>
<td>2.464E+01</td>
<td>1.631E+00</td>
<td>3.420E+05</td>
<td>2.732E+05</td>
</tr>
<tr>
<td>PSOA2</td>
<td>1.974E+04</td>
<td>1.927E+00</td>
<td>7.968E+00</td>
<td>1.526E+00</td>
<td>2.676E+02</td>
<td>4.631E+00</td>
<td>1.041E+01</td>
<td>2.974E+00</td>
</tr>
<tr>
<td>PSOA2W</td>
<td>2.305E+04</td>
<td>4.430E+00</td>
<td>4.828E-15</td>
<td>1.372E+14</td>
<td>2.257E+01</td>
<td>4.992E-01</td>
<td>7.791E+00</td>
<td>5.243E+00</td>
</tr>
<tr>
<td>MOL</td>
<td>2.384E+04</td>
<td>6.048E+00</td>
<td>2.343E+03</td>
<td>7.083E-03</td>
<td>3.833E+01</td>
<td>2.577E+01</td>
<td>9.638E+00</td>
<td>7.498E+00</td>
</tr>
<tr>
<td>PSO CY</td>
<td>2.381E+04</td>
<td>5.480E+00</td>
<td>3.098E+01</td>
<td>1.774E+01</td>
<td>3.833E+01</td>
<td>1.610E+00</td>
<td>4.960E+00</td>
<td>1.885E+00</td>
</tr>
<tr>
<td>AR PSO</td>
<td>2.385E+04</td>
<td>5.246E-01</td>
<td>2.365E+11</td>
<td>1.838E+11</td>
<td>2.122E+04</td>
<td>1.613E-00</td>
<td>2.288E+06</td>
<td>1.056E-06</td>
</tr>
<tr>
<td>AR PSO2</td>
<td>2.385E+04</td>
<td>5.654E+00</td>
<td>1.156E+05</td>
<td>6.416E+00</td>
<td>2.663E-02</td>
<td>2.493E-01</td>
<td>2.478E+01</td>
<td>2.141E+01</td>
</tr>
<tr>
<td>DPSO</td>
<td>2.385E+04</td>
<td>5.440E+00</td>
<td>5.598E+01</td>
<td>2.987E+01</td>
<td>2.700E+02</td>
<td>2.987E-01</td>
<td>1.434E+01</td>
<td>6.798E-02</td>
</tr>
<tr>
<td>AV PSO</td>
<td>2.385E+04</td>
<td>1.774E+00</td>
<td>0.000E+00</td>
<td>0.000E+00</td>
<td>2.878E-02</td>
<td>7.473E-02</td>
<td>4.441E-16</td>
<td>0.000E+00</td>
</tr>
<tr>
<td>AR PSOA2W</td>
<td>2.385E+04</td>
<td>4.384E+00</td>
<td>2.370E-03</td>
<td>3.508E-03</td>
<td>2.722E-01</td>
<td>1.865E-01</td>
<td>1.700E-02</td>
<td>9.298E-03</td>
</tr>
<tr>
<td>D PSOA2W</td>
<td>2.384E+04</td>
<td>5.563E+00</td>
<td>6.002E+00</td>
<td>1.854E+02</td>
<td>2.773E+00</td>
<td>1.903E+00</td>
<td>2.793E+02</td>
<td>3.415E+03</td>
</tr>
<tr>
<td>AR NPSO</td>
<td>2.376E+04</td>
<td>1.272E+01</td>
<td>1.294E-03</td>
<td>6.609E+01</td>
<td>5.088E+00</td>
<td>6.289E+00</td>
<td>5.485E+02</td>
<td>3.458E+02</td>
</tr>
<tr>
<td>D NPSO</td>
<td>2.384E+04</td>
<td>4.510E+00</td>
<td>3.639E-02</td>
<td>6.389E-03</td>
<td>2.800E-01</td>
<td>6.074E-01</td>
<td>2.060E+01</td>
<td>1.308E+01</td>
</tr>
<tr>
<td>AR D NPSO</td>
<td>2.376E+04</td>
<td>5.996E+00</td>
<td>1.231E-02</td>
<td>2.524E-01</td>
<td>1.010E+02</td>
<td>8.868E+00</td>
<td>1.246E+01</td>
<td>2.395E+00</td>
</tr>
</tbody>
</table>

2.6. Experimental Results
<table>
<thead>
<tr>
<th>Velocity Equation</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constriction</td>
<td>11766.53</td>
</tr>
<tr>
<td>linear inertia</td>
<td>11771.45</td>
</tr>
<tr>
<td>AIW PSO</td>
<td>11758.68</td>
</tr>
<tr>
<td>PSOA</td>
<td>10651.81</td>
</tr>
<tr>
<td>NPSO1</td>
<td>11763.56</td>
</tr>
<tr>
<td>NPSO2</td>
<td>11767.38</td>
</tr>
<tr>
<td>PSOAWL</td>
<td>11766.57</td>
</tr>
<tr>
<td>PSOA2</td>
<td>10790.92</td>
</tr>
<tr>
<td>PSOAWL2</td>
<td>11771.45</td>
</tr>
<tr>
<td>MOL</td>
<td>11758.68</td>
</tr>
<tr>
<td>PSO CV</td>
<td>11672.73</td>
</tr>
<tr>
<td>ARPSO</td>
<td>11771.43</td>
</tr>
<tr>
<td>ARPSO2</td>
<td>11766.63</td>
</tr>
<tr>
<td>DPSO</td>
<td>11771.26</td>
</tr>
<tr>
<td>AVPSO</td>
<td>11694.33</td>
</tr>
<tr>
<td>AR PSOAWL</td>
<td>11771.16</td>
</tr>
<tr>
<td>D PSOAWL</td>
<td>11762.08</td>
</tr>
<tr>
<td>AR NPSO</td>
<td>11722.51</td>
</tr>
<tr>
<td>D NPSO</td>
<td>11763.66</td>
</tr>
<tr>
<td>AR DPSO</td>
<td>11716.07</td>
</tr>
</tbody>
</table>

5. AR PSOAWL

However Table 2.6 in Section 2.6.7 shows that the best 5 performing neural networks on the training data are:

1. AR PSOAWL
2. Linear Changing Inertia PSO
3. AR PSO2
4. PSOAWL2
5. NPSO2

The linearly changing inertia consistently performs 2nd best on both the training and test data. From an optimisation perspective the AR PSOAWL is the best performing algorithm for the Watershed problem as it performs best on the training data. From a Machine Learning perspective, the PSOAWL2 is the best performing algorithm as it performs best on the test data.
2.7 Conclusion

The aim of this paper was to compare a wide range of PSO velocity update equations and to apply them to the Watershed Management problem in an offline learning manner. Meta optimisation was implemented to ensure that the optimum parameters for each velocity update equation were selected. Overall the linear changing inertia PSO performs extremely well. It is thought that the reason for this is due to the unusual parameters found by the meta optimiser which leads to its increased performance. The proposed hybrid velocity update equation AR PSOAWL performs statistically the same as the linear changing PSO. The proposed AR PSOAWL provides faster convergence however it requires more computational time to execute. When comparing each velocity update equation on each of the 8 problems individually, it was found that the AV PSO performs best on the 6 out of 8 problems. The AV PSO performs best on simpler problems but lacks the ability to perform well on more complex problems that require more exploration. A neural network trained with the proposed PSOAWL2 performs best on unseen test data for the Watershed Management problem. However this variant only performs 4th best on the training data. This highlights the problem of over training for Watershed Management control.

The main contributions of this paper are:

1. Overall the best velocity update equations are the linearly changing inertia PSO, AR PSOAWL and AV PSO. The linearly changing inertia PSO performs statistically equal to the AR PSOAWL. The AV PSO performs best on the highest number of problems but performs badly on complicated problems.

2. A linear changing inertia provides superior performance than the constriction value. This demonstrates that stable convergence does not equate to superior performance.

3. Better performance can be obtained by utilizing both a negative inertia and an increasing inertia value.

4. For the AR PSO, meta optimisation reveals that a maximum diversity threshold that is lower than the minimum diversity threshold can provide better exploration.

5. Commonly used parameters such as those recommended as standard [54] do not provide the best performance. Meta optimisation reveals that very strange
parameters that are not commonly implemented actually provide better performance.

6. The proposed hybrid velocity update equations, AR PSOAWL, performs extremely well providing faster convergence than other velocity update equations.

7. This research also provided a novel application of a neural network controller trained using PSO for the Watershed Management problem.

2.7.1 Future Work

There are many avenues of future work that have arisen as a result of this research. The inertia values and AR PSO diversity thresholds that were found by the meta optimiser lead to superior performance. A more in depth analysis into these parameters is needed to determine why this is. The AV PSO performs well on simple optimisation problems but not on more complex problems. Modifying this algorithm to also perform well on more complex problems would therefore be a worthy route for future research.
Chapter 3

A Combined Genetic Algorithm - Differential Evolution Strategy

The work outlined in this chapter was published in:


3.1 Introduction

This research focuses on a particular type of NE, Topology and Weight Evolving Artificial Neural Network (TWEANN), which dates back to 1992 [89]. TWEANN algorithms serve a dual purpose of optimizing the structure and the weights of the neural network. One of the most well-known TWEANN algorithms is NeuroEvolution of Augmenting Topologies (NEAT) [347]. NEAT was shown to provide increased performance compared to other neuroevolution approaches. The NEAT algorithm begins with a minimal network size. As the algorithm executes, more neurons and connections are added via mutation. Networks are divided into species to protect
innovation. One of the main advantages of this algorithm is that by starting with a small network, the algorithm is biased to produce smaller networks. The NEAT algorithm provides the inspiration for the research presented in this paper.

In recent years, many algorithms have been developed for continuous optimization problems, e.g. Differential Evolution (DE) [348], Particle Swarm Optimization (PSO) [205], etc. These methods have shown to be very effective at the task of optimizing the weights of neural networks [178, 339, 285]. It is therefore hypothesized that it would be beneficial to incorporate these continuous optimization algorithms for the task of neural network topology and weight optimization. This paper proposes a novel TWEANN algorithm, Neuro Differential Evolution (NDE) that utilizes the strengths of GAs to optimize the topology of the neural network while leveraging the advantages of DE as a continuous function optimizer, to optimize the weights of the network. NDE implements many of the desirable features of NEAT, such as a bias towards minimal network design and speciation to protect innovation. Previous examples of a combined evolutionary strategy include the use of Grammatical Evolution to evolve the network topology with a Genetic Algorithm to optimize the network weights [6] and the use of evolutionary programming to evolve the network structure combined with simulated annealing and backpropagation to evolve the network weights[396].

Neuroevolution methods have been applied to many diverse real world problems. There are many advantages to using these methods such as: there is no need for target outputs, they are suitable for large and complex problems and they are resistant to noise. This makes them particularly well suited to engineering control problems. This research will apply a range of neuroevolution algorithms to watershed management [394].

The watershed management problem consists of managing the distribution of a finite amount of water to a number of interested individuals that seek to maximise their consumption. The various uses of the water include municipal use, irrigation, hydroelectricity and for the surrounding ecosystems. There have been a number of approaches to this problem including: Multi Agent Systems [394, 139], Multi Agent Reinforcement Learning [252], Multi Agent Systems combined with Genetic Algorithm [24], Robust Decision Making [201] and Multi Population Evolutionary Algorithm [120]. Each of these approaches address the watershed management as a distributed control problem where multiple controllers are implemented to distribute the water to each party. This research will involve evolving a single centralized neural network
controller to allocate the water to the various parties seeking water.

The primary contribution of this research is presenting a meaningful combined approach using differential evolution and genetic algorithms for neural network topology and weight optimization. Many hybrid differential evolution algorithms have already been proposed in the literature [164, 88]. The proposed NDE algorithm in this paper is applicable to problems that have both discrete and continuous elements. As such it is capable of evolving neural network topologies and weights while previous methods are not.

The contributions of this paper are as follows:

1. The application of differential evolution to topology and weight evolving artificial neural networks.

2. Combining genetic algorithms with differential evolution in a meaningful way that exploits their respective strengths, to evolve neural networks.

3. Applying neuroevolution to watershed management.

3.2 Background

This section will start by giving a broad overview of the relevant literature on neuroevolution and its applications. The NEAT and ESP algorithms will then be outlined. The section will finish with a description of the differential evolution algorithm and some of its applications.

3.2.1 Neuroevolution

A simple definition of neuroevolution would be the use evolutionary algorithms to search for the optimal neural network configuration to approximate some function [395]. Neuroevolution algorithms have a wide range of applications including Atari games [159], pattern recognition [180], classification [368, 63], air pollution time series [280], decision making [216] and robotics [214]. As far as the authors can determine, there exists no research in the literature relating to the application of neuroevolution algorithms to watershed management. This is one of the primary contributions of this research.
Neuroevolution methods consist of a population of solutions. The population of networks iteratively improves over time as a result of the application of evolutionary methods to the population, such as genetic algorithms. In neuroevolution, aspects of the neural network design are encoded into genotype. These typically contain network information such as synaptic weight values, number of neurons, connectivity, etc. These network traits form the genotype. These genotypes are evolved over a series of generations. The phenotype is the expression of the genotype. In neuroevolution, the phenotype is the actual neural network. At each iteration, genotypes are selected via one of many methods (linear ranking, roulette wheel, etc.). These genotypes are then mated through crossover and then possibly mutated to form the next generation of genotypes.

Neurevolution methods can be sub divided into direct encoding schemes and indirect encoding schemes. In direct encoding schemes, the bits in the genotype are mapped directly onto the phenotype [272, 143]. Indirect encoding schemes are more abstract. Instead the genotype provides more general rules as to how the phenotype should be formed [235, 212, 372]. These indirect schemes are advantageous in forming large networks as they do not require large genotypes.

3.2.2 NEAT Algorithm

NEAT is one of the most popular neuroevolution algorithm and was first proposed by Stanley and Miikkulainen in 2002 [347, 345]. It is a state of the art method for solving the pole balancing problem and operates using a similar philosophy to the proposed NDE algorithm. The NEAT algorithm is also one of the key algorithms that have inspired the NDE algorithm presented in this research. For this reason, NEAT is implemented as a benchmark to compare the proposed NDE algorithm to. This method uses a GA to evolve both the network topology and weights. It employs a direct encoding scheme and begins with a minimal network. As the algorithm runs, more complexity is added to the network by adding neurons and connections via mutation. Genes are divided into species based on their innovation number. NEAT remembers the innovation number of each gene so that crossover can occur between networks with different topologies.

The distance between genes can be easily measured by comparing their chromosomes. The terms genotype and chromosome are used interchangeably for methods with
3.2. Background

direct encoding schemes. When comparing two chromosomes, the genes that do not match are considered to be either disjoint or excess. The distance between genes is therefore just the sum of the number of disjoint and excess genes. Genes are categorized into species based on this distance. This speciation allows the algorithm to protect topological innovation.

3.2.3 ESP Algorithm

The second algorithm that NDE will be compared to is Enforced SubPopulations (ESP) [142]. This algorithm was proposed by Gomez and Miikkulainen in 1999 and is an extension of the Symbiotic Adaptive Neuro-Evolution (SANE) algorithm by Moriarty and Miikkulainen, originally proposed in 1996 [271]. Unlike NEAT and the proposed NDE, ESP uses a fixed neural network size. In ESP, the population is divided into distinct sub populations. Networks are formed by selecting a neuron from each sub population and combining them to form a network. Crossover only occurs between chromosomes within the same subpopulation. No inter sub population breeding is allowed. Offspring also remain within the parents’ sub population. ESP is a state of the art neuroevolution algorithm and will therefore be the second benchmark algorithm that the proposed NDE will be compared to.

3.2.4 Differential Evolution Algorithm

Differential Evolution is an optimization algorithm first proposed by Storn and Price in 1997 [348]. DE is a state of the art algorithm for optimizing large and complex continuous search spaces. The algorithm itself needs no prior information about the problem space, does not rely on any gradients and is highly resilient to noise. For these reasons, DE is the obvious choice for optimizing network weights. DE has been applied to many real world problems such as robotics [65] and energy systems [161]. DE has also previously been applied to neural network weight optimization [178, 339] but never to topology and weight optimization or the pole balancing problem. This is one of the key contributions of this research. The DE algorithm uses evolutionary operators to find the optimum solution. The algorithm locates the optimum solution by iteratively combining each candidate position (solution) with three other distinct positions, as shown in Algorithm 3. \( CR \) is the crossover probability and \( F \) is the differential weight.
Chapter 3. A Combined Genetic Algorithm - Differential Evolution Strategy

Initialize X agents with random positions

\[ \text{while } \text{Iteration } t < T_{\text{max}} \text{ do} \]

\[ \text{for } \text{Agent} = 1 \text{ to } N \text{ do} \]

\[ \text{Select 3 other agents A, B and C} \]

\[ \text{Select random dimension index } R \]

\[ \text{for dimension} = 1 \text{ to } D \text{ do} \]

\[ \text{generate random number } r \in [0,1] \]

\[ \text{if } r < CR \text{ Or } i = R \text{ then} \]

\[ \text{new position } y_i = a_i + F \times (b_i - c_i) \]

\[ \text{else} \]

\[ y_i = x_i \]

\[ \text{end if} \]

\[ \text{end if} \]

\[ \text{end if} \]

\[ \text{end } \text{for} \]

\[ \text{end } \text{for} \]

Return best solution

**Algorithm 3:** Differential Evolution (DE) Algorithm

The primary contribution of this research is presenting a meaningful combined approach using differential evolution and genetic algorithms for neural network topology and weight optimization. Many hybrid differential evolution algorithms have already been proposed in the literature \[164, 88\]. The proposed NDE algorithm in this paper is applicable to problems that have both discrete and continuous elements. As such it is capable of evolving neural network topologies and weights while previous methods are not.

### 3.3 Neuro Differential Evolution (NDE)

In this section, the proposed Neuro Differential Evolution (NDE) algorithm will be outlined. The key insight of NDE is to address the tasks of topology and weight optimization separately. Selecting the correct neural network topology is a discrete optimization problem while choosing the correct weights for the network is a continuous optimization problem. NDE uses a genetic algorithm to optimize the topology of the network and differential evolution to optimize the weights of any given network topology. The second advantage of NDE is that a network size does not need to be selected. NDE effectively grows a neural network by beginning with a network of just one neuron. Populations of networks of the same size of neurons are considered to be of the same species. New species with extra neurons are added in two ways: 1) They are added at each iteration with a certain probability. 2) If the progress of the algorithm stagnates. Adding neurons to the network in this way helps to ensure
that the smallest suitable network is found. Only networks of the same species are al-
lowed to be mated with one another. Similar to the NEAT algorithm, having multiple
species of networks increases diversity. The final advantage of NDE is that it allows
for the evolution of recurrent neural networks. This is a desirable quality of NDE as
recurrent connections can provide increased performance on tasks that require net-
work memory such as the double pole balancing problem that will be described later.
The pseudocode in Algorithm 4 in Section 3.3.4 describes how the algorithm func-
tions. Before describing the functioning of the NDE algorithm in detail, the network
encoding, speciation and growth must be outlined.

3.3.1 Encoding

Each chromosome represents a configuration of a network. NDE utilizes a direct
encoding scheme whereby each bit in a chromosome corresponds with a network con-
nection. A chromosome of length 6, as illustrated in Figure 3.1, corresponds to a
network that has 6 weights if fully connected. A 1 in the chromosome means that a
connection between two neurons is enabled. A 0 implies that the connection is dis-
abled. If the network is fully connected, the chromosome will consist of all 1s. Each
bit in the chromosome has an associated weight value. These weights have a value of
0 where the connection is disabled. NDE’s encoding scheme is illustrated graphically
in Figure 3.1.

3.3.2 Speciation

In order to add diversity to NDE and enable crossover, chromosomes are divided into
species based on the size of the network they represent, i.e. the number of hidden
neurons. Mating only occurs between chromosomes within the same species. This
ensures that all bits in each genotype match up with the correct network connection.
Parents from the same species are selected for mating via roulette wheel selection.
When two parents are selected, their chromosomes are split midway and combined
with their partners as illustrated in Figure 3.2.

Note that in Figure 3.2, all parents and all offspring have the same chromosome
length and network size. This is because they are from the same species. Figure
3.3 illustrates the difference between species in NDE. If the algorithm stagnates, i.e.
has not improved in fitness for a predetermined number of iterations, or if a random
Chapter 3. A Combined Genetic Algorithm - Differential Evolution Strategy

Figure 3.1: Neuro Differential Evolution Encoding. A direct encoding scheme is implemented to map the genotype to the network phenotype. A 0 implies no connection while a 1 implies that a connection exists.

number (r) is less than a threshold value (sm) a new species is created. This consists of creating a new population of chromosomes that correspond to a network with one extra neuron than the previous species. The chromosomes within this new species are now incapable of mating with those from another species.

3.3.3 Network Growth

Many TWEANN algorithms are initialized with random network topologies [142]. This is done to ensure that a sufficient level of diversity exists within the population. However by initializing the algorithm in this way, the population of network topologies are potentially unnecessarily large. An alternate approach is to initialize the algorithm with a minimal network size and to grow the network as needed. This biases the algorithm to produce smaller networks and therefore keeps the search space smaller. This approach was first implemented with the NEAT algorithm and produced promising results when compared to previous methods. This is also the approach that was adopted for NDE.

The proposed NDE algorithm begins with a single hidden neuron as shown in Figure 3.3. As the algorithm executes, larger networks are evaluated. These new species of chromosomes that correspond to larger networks are then forgotten if they do not
3.3. Neuro Differential Evolution (NDE)

Figure 3.2: Neuro Differential Evolution Crossover. This image illustrates how crossover occurs between genes within the same species and how this relates to the network topologies.

provide a gain in performance. The proposed NDE algorithm has a maximum number of species that the algorithm allows at any one time. The algorithm begins with just a single species and adds new species as previously outlined. When the algorithm reaches a point where a new species is created and the total number of species exceeds a predetermined maximum, the species with the worst fitness is removed (or goes extinct to use terminology from life sciences).

It is true to say that by removing the worst performing species, the NDE algorithm is forgetting potentially useful information. This is necessary however because the only occasion that a species will be removed is if a species comes into existence that improves upon the best fitness found by any of the existing species. If no species are ever removed, a situation could arise where there is an overwhelmingly large number of species, many of which are incapable of producing solutions that are close to optimal. This exceedingly large group of species would result in wasted computations that
Chapter 3. A Combined Genetic Algorithm - Differential Evolution Strategy

Figure 3.3: Neuro Differential Evolution Species Mutation. When \( \text{rand()} < sm \), a new species is created. This new species consists of a network with an additional neuron. The weights and connectivity of this new neuron are initialized at random as illustrated above.

would be better spent on more promising species of networks. The loss of potentially useful information can also be guarded against by selecting a large maximum number of species. Removing species is necessary however for the algorithm to operate efficiently.

3.3.4 NDE Algorithm

The NDE algorithm is initialized with a single species of \( c \) chromosomes. Each of these initial chromosomes represent a network with just a single hidden neuron as illustrated in Figure 3.1. It is possible to initialize the algorithm with 0 neurons but this would be insufficient for any problems with non-trivial complexity. These problems are not
of any interest and for this reason the initial chromosomes correspond to a network
with one hidden neuron rather than none.

At each iteration of the algorithm, differential evolution is applied to the network
weights that correspond to each chromosome in each species. The aim here is to
optimize the weights of a given network topology to determine the fitness of that
network topology and therefore that chromosome. After the weights corresponding
to each chromosome in a species are optimized using differential evolution, the best
performing chromosome and corresponding set of weights are automatically passed on
unchanged to the next generation of that species. This ensures that the best network
configurations are not forgotten. A roulette wheel selection method is then used to
select parents for mating in order to produce child chromosomes for the next gener-
ation. As previously outlined, the mating of two chromosomes can be seen in Figure
3.2. Each bit in the child chromosome has a probability \( mp \) of undergoing muta-
tion. Producing offspring in this way ensures that the algorithm maintains diversity
in subsequent generations.

New species are added to the population in two ways. At each iteration of the al-
gorithm there is a \( s \times sm \) probability of a new species being added, where \( s \) is the
current number of species and \( sm \) is the species mutation probability. The second
method that a new species is added is if the algorithm’s performance has stagnated.
The process for species mutation has been outlined in Figure 3.3. At each iteration
of NDE, if the maximum number of allowed species is exceeded, the worst performing
species is removed. This is done for the reasons outlined in Section 3.3.3. NDE runs
until a predetermined maximum number of evaluations of the problem is reached or
until some fitness threshold is reached. In this research, the maximum fitness thresh-
old is only implemented for the cart pole experiments as the algorithm must pass a
generalization test. For the subsequent watershed experiment, the algorithm is left
to run for the maximum number of evaluations. Algorithm 4 describes in detail how
NDE operates.

3.4 Pole Balancing Experiment

This section will outline the pole balancing experiment used to evaluate the propsed
NDE. The performance of NDE will be compared to NEAT and ESP on four variants
of the non markovian double pole balancing problem.
Initialize:
Create c chromosomes for species of 1 neuron

while \( E \text{val} < \text{maxEvals} \land \text{fitness} < \text{threshold} \) do
  while \( \text{numberOfSpecies} > \text{maxSpecies} \) do
    Remove worst species
  end
  if lastImprovement > stagnationTime then
    Create new species with extra neuron
    lastImprovement = 0
  end if
  lastImprovement++
  for Species = 1 to S do
    for Chromosome c = 1 to C do
      Apply DE to weights of c (Alg. 3)
      if Fitness > bestOverallFit then
        bestOverallChromosome = c
        lastImprovement = 0
      end if
      if Fitness > bestSpeciesFit then
        bestSpeciesChromosome = c
      end if
    end
    ADD bestSpeciesChromosome to next gen
  end
  while nextGenSize < currentGenSize do
    SELECT parents for mating
    Perform crossover
    MUTATE child chromosome with prob, mp
    ADD child chromosome to next gen
  end
  if rand() < sm then
    Create new species with extra neuron
    for Chromosome c = 1 to C do
      Apply DE to weights of c
      if Fitness > bestOverallFit then
        bestOverallChromosome = c
        lastImprovement = 0
      end if
      if Fitness > bestSpeciesFit then
        bestSpeciesChromosome = c
      end if
    end
  end
end
Return best Network

Algorithm 4: Neuro Differential Evolution (NDE)
3.4. Pole Balancing Experiment

3.4.1 Pole Balancing Problem

In order to evaluate the performance of the proposed NDE algorithm, NDE will be applied to a variant of the pole balancing problem called the non markovian double pole balancing problem.

Figure 3.4: Snapshot of Double Pole Balancing Problem 3D Simulation [142]. The evolved neural network controller must continuously balance both poles by applying a force to the cart. Balancing two poles simultaneously without velocity information is a difficult task.

The standard single pole balancing problem consists of a cart placed on a movable track. One end of the pole is placed on the cart and the system is initialized where the pole is slightly off perpendicular. The problem consists of moving the cart so that the pole remains balanced.

3.4.2 Double Pole Problem

The double pole balancing problem is identical to the single pole balancing problem with the addition of an extra pole, typically of a different length from the first pole [152]. This is a much more difficult problem than the single pole balancing problem. For this reason, it is now a much more commonly used benchmark problem used to gauge the performance of control systems. The criteria for success in the double pole balancing problem is for the system to remain balanced for 100,000 time steps and to pass a generalization test that will be outlined later. If the angle of either poles was greater than 36 degrees from vertical, the system is no longer balanced. The system is initialized with the long pole at an angle of 1 degree from vertical.
3.4.3 Non Markovian Variant

This research will consider a more difficult variant of the double pole balancing problem in which the control algorithm does not have velocity information. As a result of this, the NDE algorithm must internally compute these velocities and use them to balance the poles. Since the controller only has access to the position information of the cart and the pole, this problem is classified as a non markovian problem. Modern algorithms can solve the markovian variant of the double pole balancing problem in a few, thousand iterations. Removing velocity information is an effective way to substantially increase the difficulty of the problem. The markovian variant of the double pole balancing problem, in which the controller has access to velocity information, can be solved easily by modern neuroevolution approaches such as NEAT and ESP. The markovian variant has 6 state variables, the cart’s position $x$, the cart’s velocity $\dot{x}$, the long pole’s position $\theta_1$, the long pole’s velocity $\dot{\theta}_1$, the short pole’s position $\theta_2$ and the short pole’s velocity $\dot{\theta}_2$. This variant will not be considered in this research as it is too easy for modern neuroevolution algorithms. Instead, NDE, NEAT and ESP will be applied to four variants of the non markovian double pole balancing problem. At each variation of the problem, the short pole length will be incrementally increased. This will increase the difficulty of the problem. The purpose of doing this is to test at which point these algorithms fail. The state variables in the non markovian case are the positions of the cart and two poles.

3.4.4 Fitness Function and Generalization

Since the non markovian variant of the double pole balancing problem is much more difficult, a specialized fitness function (Equation 3.1) has been proposed by Gruau et al. to prevent the cart from oscillating back and forth to balance the poles [152].

$$F = 0.1f_1 + 0.9f_2$$  \hspace{1cm} (3.1)

Where $f_1$ relates to the number of time steps $t$ that the system is balanced for, as shown in Equation 3.2.

$$f_1 = t/1000$$  \hspace{1cm} (3.2)
If the number of time steps $t \leq 100$, $f_2 = 0$. If $t > 100$, $f_2$ is described by Equation 3.3.

$$f_2 = \frac{0.75}{\sum_{t=t-100}^{t}(|x^i| + |\dot{x}^i| + |\theta_1^i| + |\dot{\theta}_1^i|)}$$

(3.3)

If the system is balanced for 100,000 time steps (30 minutes of simulated time), the controller must then pass a generalization test. The purpose of this test is to determine if the system can recover from a variety of different starting positions. The generalization test will examine 625 distinct initial states. The evolved controller must be able to balance the system for 1,000 time steps in at least 200 of the 625 initial states. The initial states will be each state variable ($x$, $\dot{x}$, $\theta_1$ and $\dot{\theta}_1$) with values of 0.05, 0.25, 0.5, 0.75 and 0.95 times the range of each variable. In each of these initial states, $\theta_2$ and $\dot{\theta}_2$ will be set to 0.

The pole balancing experiment implemented in this research is the same as that used in previous experiments [347, 142]. The physics of the pole balancing system was implemented using the fourth order Runge-Kutta method, using a time step of 0.01 seconds. Each state variable is scaled between -1 and 1 before being fed into the network. The network outputs a force between -10 and 10 N every 0.2 seconds. The track is 4.8 meters long. The length of the long pole is unchanging in all experiments and set to 1 m. The length of the short pole is increased 4 times for each of the experiments, making the problem progressively more difficult for each short pole increase. The lengths of the short pole for each experiment are 0.1 m, 0.2 m, 0.3 m and 0.4 m. Increasing the short pole length substantially increases the difficulty of the problem.

### 3.5 Watershed Management Experiment

The Watershed Management problem is a resource allocation problem which consists of multiple parties interested in the same finite resource [394].

Each of these parties are withdrawing water from a common and finite source, i.e. a river, for their own individual needs. This particular problem has multiple constraints and consists of continuous variables. The problem also involves a dynamic environment as the amount of water available in the river is not fixed. The problem ultimately consists of maximising the overall utility of this water for all interested
parties. There are 6 variables which must be optimized, 4 of which are controlled directly, the remaining 2 are reactive variables which are indirectly optimized. The 4 direct variables are the water withdrawn from the river for municipal and industrial use in the city ($x_1$), the water withdrawn for the irrigation of farms ($x_4$ and $x_6$) and finally the water released from a dam for hydro power generation ($x_2$). These will be the outputs of the evolved neural networks. The 2 reactive variables are the water available for the ecosystems ($x_3$ and $x_5$). Each of these variables must be selected to maximise a series of objective functions representing the benefits obtained from the water by the various interested parties.

The benefit of the water withdrawn for each interested party is represented by the following function:

$$f_i(x_i) = a_ix_i^2 + b_ix_i + c_i$$

(3.4)

Where $a_i$, $b_i$, and $c_i$ are dimensionless constants corresponding to each party [394]. Their values are highlighted in the following table, along with the values for $\alpha_i$ which represents the minimum values for $x_i$ in $L^3$, where $i$ represents each objective.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value ($L^3$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$-0.20$</td>
<td>$b_1$</td>
<td>6</td>
<td>$c_1$</td>
<td>$-5$</td>
<td>$\alpha_1$</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_2$</td>
<td>$-0.06$</td>
<td>$b_2$</td>
<td>2.5</td>
<td>$c_2$</td>
<td>0</td>
<td>$\alpha_2$</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_3$</td>
<td>$-0.29$</td>
<td>$b_3$</td>
<td>6.28</td>
<td>$c_3$</td>
<td>$-3$</td>
<td>$\alpha_3$</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_4$</td>
<td>$-0.13$</td>
<td>$b_4$</td>
<td>6</td>
<td>$c_4$</td>
<td>$-6$</td>
<td>$\alpha_4$</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_5$</td>
<td>$-0.056$</td>
<td>$b_5$</td>
<td>3.74</td>
<td>$c_5$</td>
<td>$-23$</td>
<td>$\alpha_5$</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_6$</td>
<td>$-0.15$</td>
<td>$b_6$</td>
<td>7.6</td>
<td>$c_6$</td>
<td>$-15$</td>
<td>$\alpha_6$</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The indirect variables relating to the water available to the surrounding ecosystem, $x_3$ and $x_5$, are not directly controlled. They must be calculated using the following equations:

$$x_3 = Q_2 - x_4$$

(3.5a)

$$x_5 = x_2 + x_3 - x_6$$

(3.5b)

Where $Q_2$ is the monthly tributary inflow in $L^3$. The values for $Q_2$ will be outlined later. The problem variables $x_i$ are is subject to the following constraints which restrict their potential values.
3.5. Watershed Management Experiment

\[
\begin{align*}
\alpha_1 - x_1 & \leq 0 \\
\alpha_2 - Q_1 + x_1 & \leq 0 \\
x_2 - S - Q_1 + x_1 & \leq 0 \\
\alpha_4 - x_3 & \leq 0 \\
\alpha_3 - x_4 & \leq 0 \\
\alpha_4 - Q_2 + x_4 & \leq 0 \\
\alpha_6 - x_5 & \leq 0 \\
\alpha_5 - x_6 & \leq 0 \\
\alpha_6 - x_2 - x_3 + x_6 & \leq 0
\end{align*}
\] 

(3.6a)

(3.6b)

(3.6c)

(3.6d)

(3.6e)

(3.6f)

(3.6g)

(3.6h)

(3.6i)

The parameter $S$ represents the storage capacity of the dam in $L^3$ while both $Q_1$ and $Q_2$ represents the monthly inflows of water into the mainstream and tributary respectively in $L^3$. The Watershed problem evaluated in this paper will consist of a total of 100 flow states for the river. The values for $Q_1$, $Q_2$ and $S$ can be found in the Appendix A.

The overall system utility of the water allocated to all interested parties is calculated as the sum of the utility that the water provides to each party. This is represented by Equation 3.7.

\[
F(x) = \max \sum_{i=1}^{6} f_i(x_i)
\]

(3.7)

The watershed problem is graphically illustrated in Figure 3.5 from the research of Yang et al. [394].

### 3.5.1 Boundary Definition

In order to address the Watershed Management problem, the maximum and minimum possible values for each direct variable must first be explicitly defined. The ranges for each of the directly optimized variables are defined as follows based on Equation 3.6:
Chapter 3. A Combined Genetic Algorithm - Differential Evolution Strategy

Figure 3.5: Watershed Illustration [394]

\[ \alpha_1 \leq x_1 \leq Q_1 - \alpha_2 \quad (3.8a) \]
\[ \alpha_3 \leq x_4 \leq Q_2 - \alpha_4 \quad (3.8b) \]
\[ 0 \leq x_2 \leq S + Q_1 - \alpha_1 \quad (3.8c) \]
\[ \alpha_5 \leq x_6 \leq S + Q_1 + Q_2 - \alpha_1 - \alpha_3 - \alpha_6 \quad (3.8d) \]

### 3.5.2 Constraint Handling

A static penalty method [340] will be implemented to handle any constraint violations highlighted in Equation 3.6. This penalty function will be incorporated into the fitness of any given solution. The penalty function is defined in Equation 3.9.

\[ f_p = \sum_{i=1}^{N} C(|h_i + 1|\delta_i) \quad (3.9) \]

Where \( N = 9 \) is the total number of constraints handled using this method per flow.
scenario, $C = 10E2$ is the violation constant, $h_i$ is the violation amount of each constraint and $\delta = 0$ if there is no violation for a particular constraint and $\delta = 1$ if a constraint is violated. The violation constant $C = 10E2$ was selected so that any solution which violates a constraint will have a significantly lower fitness than acceptable solutions. Lower $C$ values were evaluated but caused each algorithm to converge on infeasible solutions.

### 3.5.3 Fitness Function

In order to establish the fitness of a given solution, a fitness function must first be defined. This fitness function will be a combination of the overall performance of the system (Equation 3.7) and the penalty function for any constraint violations (Equation 3.9). These two equations are combined to form the fitness function in Equation 3.10 which is to be maximised.

$$F = \sum_{i=1}^{6} f_i(x_i) - \sum_{j=1}^{N} C(|h_j + 1|\delta_j) \quad (3.10)$$

The network input for the watershed management problem consist of the 3 flow rates previously outlined, normalised between 0 and 1. The network outputs 4 values corresponding to the water allocation for directly controlled interested parties.

### 3.6 Results

This section will outline the specific details of how each experiment was conducted. The results of the experiments will then be presented and discussed in order to gain some insight.

In total 3 algorithms were evaluated and compared over the 5 experiments. The 3 algorithms evaluated were NEAT, ESP and the proposed NDE algorithm. The 5 experiments conducted were 4 variants of the non markovian double pole balancing problem and the watershed problem. All problems were evaluated over 10 runs, each run consisting of $10^6$ evaluations.

The NDE algorithm was implemented with the following parameters: Species Mutation probability $sm = 0.1$, Chromosome Mutation probability $mp = 0.4$, chromosomes
per species $C = 3$, maximum number of species $max\text{Species} = 5$, stagnation time $\tau = 10$, differential evolution maximum number of iterations $I_{max} = 3 \times D$, (where $D$ is the number of weights in the current network configuration), differential evolution crossover probability $CR = 0.9$, differential weight $F = 0.5$ and finally the number of differential evolution agents $N = D$, ($N = 4$ if $D < 4$). Parameter tuning revealed that these parameters gave the best performance.

NEAT and ESP were implemented using the same parameters as specified in their original implementations for the non markovian double pole balancing problem [347, 142]. Parameter tuning revealed that these parameters gave the best performance. A recurrent neural network was evolved by each algorithm in all experiments except the watershed management problem. This is because the internal memory that is provided by a recurrent memory is of benefit when solving the cartpole problems as the outputs of the network at time $t$ affects the environment at time $t + 1$. This is not true for the watershed problem as previously outlined. As a result, a feed forward neural network will be evolved for the watershed problem.

The two tailed t test with a significance level of $\alpha = 0.05$ was conducted to determine significant performance differences when comparing algorithms. All values were rounded to one decimal place in the pole balancing experiments and 4 decimal places in the watershed experiment. In cases where an algorithm fails to find an acceptable solution in the maximum allowed $10^6$ evaluations, $10^6$ is used when calculating the average number of evaluations.

### 3.6.1 Pole Balancing Experiments

As is evident from Table 3.2 and Figure 3.6, all algorithms perform very similarly on the easier variant of the cart pole problem. In experiment 1, the short pole is 0.1m long. NDE performs statistically the same as NEAT and ESP. In experiments 3 and 4, where the short pole is 0.3m and 0.4m long respectively, NDE performs statistically better than both ESP and NEAT.

Another observation that can be made from Table 3.2, is the high standard deviation of each of the algorithms tested. In many of the results reported in Table 3.2, the standard deviation is as large as or larger than the average number evaluations taken. This highlights how difficult the non markovian double pole balancing problem is. Each of the algorithms tested have a large amount of variance in the number of
Table 3.2: Average Evaluations Taken for 4 Variants of Non Markovian Double Pole Balancing Problem. This table displays the average evaluations taken and standard deviation over 10 runs for the proposed NDE, NEAT [347] and ESP [142] algorithms.

<table>
<thead>
<tr>
<th>Pole Increase</th>
<th>NDE Avg (StDev)</th>
<th>NEAT Avg (StDev)</th>
<th>ESP Avg (StDev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>46313.8 (32991.3)</td>
<td>30945.3 (22574.8)</td>
<td>64905.5 (38867.8)</td>
</tr>
<tr>
<td>1</td>
<td>99628.5 (109294.0)</td>
<td>239714.6 (401160.3)</td>
<td>150851.1 (57237.8)</td>
</tr>
<tr>
<td>2</td>
<td>167100.3 (111380.5)</td>
<td>459959.4 (447950.8)</td>
<td>327049.6 (260866.7)</td>
</tr>
<tr>
<td>3</td>
<td>279708.6 (232386.9)</td>
<td>918183.3 (258727.1)</td>
<td>453825.4 (169078.3)</td>
</tr>
</tbody>
</table>

evaluations they require for each run. This is as a result of the stochastic nature of each of these algorithms. A possible route of future research would be to investigate how to increase the consistency of these algorithms.

Figure 3.6: Short Pole Length Increases vs Average Number of Evaluations Taken. This figure graphs the increase in the number of evaluations taken as the length of the short pole is iteratively increased. This illustrates the scalability of the proposed NDE, NEAT [347] and ESP [142] algorithms.

One of the objectives of this paper was to investigate at what point each of these neuroevolution algorithms fail to solve the non markovian double pole balancing problem. In this regard, NDE and NEAT performed best with no failures. This is illustrated in Figure 3.7. As previously mentioned, Table 3.2 demonstrates that the performance of these algorithms lacks consistency with regards to the number of evaluations required. The results in Figure 3.7 however do reveal that the NDE and ESP are capable of consistently finding an acceptable solution that passes the generalization test in ever
3.6.2 Watershed Management Experiment

When evaluated on the watershed problem, all algorithms were able to find acceptable solutions without any constraint violations. As is illustrated in Figure 3.8, the proposed NDE algorithm converges the fastest to the best solution with an average fitness of $24137.0401 \pm 0.5242$. The next best performing algorithm is ESP with an average fitness of $24087.3883 \pm 58.8458$ followed by NEAT which had an average fitness of $23712.3900 \pm 188.8516$. Figure 3.8 shows that the NEAT algorithm has a faster initial convergence than ESP but then slows as the algorithm progresses in its operation. It also appears that the NEAT algorithm has not finished converging after $10^6$ evaluations. In terms of reliability and robustness, NDE has a much lower standard deviation than ESP and NEAT, 2 and 3 orders of magnitude lower respectively. This indicates that NDE is consistent in its performance. NDE is statistically better when evaluated using the two tailed t test. When compared to the PSO results from the previous chapter, it is also found that NDE performs statistically better than the
3.6. Results

The best performing PSO variant (AR PSOAWL) which converged to a solution with an average fitness of $23854.4598 \pm 4.3446$. This is statistically worse than NDE and ESP but statistically better than NEAT.

![Fitness Convergence for Watershed Management Problem](image)

The results presented here demonstrate that neuroevolution is a suitable approach for the watershed management problem. None of the algorithms failed to find an acceptable solution on any run. It is evident however that the proposed NDE algorithm performs much better when addressing this problem. NDE consistently converges faster to a better solution.

3.6.3 Summary

The results of the 5 experiments conducted are summarized in Table 3.3. This table outlines how many times each algorithm failed to find an acceptable solution on each problem. It is evident here that the proposed NDE and ESP algorithms were the most robust on the problems evaluated. They did not fail on a single problem. In each experiment the ESP algorithm converged slower than NDE. The NEAT algorithm converged faster than NDE and ESP on the first variant of the non markovian double pole balancing problem and initially converged faster than the ESP algorithm on the watershed problem. This indicates that if limited computational resources are available, the NEAT algorithm would be a suitable choice due to its faster convergence.
However if the problem complexity is known to be high and more computational resources are available for many problem evaluations, the NDE would be a more sensible and safer choice of algorithm.

Table 3.3: Number of Failed Runs. This table presents the number of failed runs (out of 10) for each algorithm on each experiment. A failed run on the cart pole problem refers to failing to pass the generalization test.

<table>
<thead>
<tr>
<th>Problem</th>
<th>NDE</th>
<th>NEAT</th>
<th>ESP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cart Pole 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cart Pole 2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Cart Pole 3</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Cart Pole 4</td>
<td>0</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>Watershed</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

This paper compared the performance of the proposed NDE to two state of the art neuroevolution algorithms, NEAT and ESP. However it is not entirely fair to compare the performance of NDE and NEAT to ESP. ESP optimizes the topology and weights of the network, however the size of the network must be selected by the user beforehand. This is not the case for NDE and NEAT. Each of these algorithms optimize the size, topology and weights of the network. Each of these algorithms are predisposed to finding the smallest network possible to solve a particular task. This means that extra computational resources are required to grow the network. ESP on the other hand does not have this additional computational burden, instead the size of the network is determined by parameter tuning before conducting the experiments. The computational resources spent on finding the optimum network size before conducting the experiments is not portrayed in the results.

There are numerous advantages for evolving neural networks using NDE rather than simply applying DE to the weights of a network. Firstly, as previously mentioned, the network size does not need to be determined by the user when implementing NDE. NDE is biased towards producing smaller networks. When applying DE to a network, the user must determine the size of the network, usually by conducting parameter sweeps. The second advantage of using NDE is it’s ability to optimize the topology of the network rather than just the weights, as is the case for DE. This is beneficial as it allows the development of networks with fewer connections, which are easier to optimize and have fewer components for physical hardware implementations. The ability to optimize the topology of the network also allows for the development of more modular and regular networks, which have been shown to be better at learning multiple tasks [200, 77]. This would be advantageous for the engineering control problems evaluated here in scenarios where the environment dramatically changes.
None of this is possible when simply applying DE to the weights of the network. The results presented here demonstrate the effectiveness of NDE at solving the non markovian double pole balancing problem, which many state of the art reinforcement learning algorithms have failed to solve [141]. The results here also demonstrate the applicability of NDE to real world engineering control problems. This further highlights the usefulness of the proposed NDE.

3.7 Conclusion

The central aims of this research were to firstly investigate if differential evolution could be incorporated into the neuroevolution paradigm to evolve both network topology and weights and secondly to establish how suitable neuroevolution algorithms are for engineering control problems. The results presented in the previous section indicate that NDE is an extremely robust algorithm and performs well compared to the current state of the art. The proposed NDE performs statistically equal to NEAT and ESP on the easier variants of the non markovian double pole balancing problem. As the difficulty of the problem increases, NDE significantly outperforms other neuroevolution algorithms due to its robustness and scalability. When evaluated on the watershed management problem, all algorithms find acceptable solutions, however NDE performs statistically better.

In summary, the contributions of this paper are:

1. Differential evolution can be applied to topology and weight evolving artificial neural networks. The proposed NDE performs better than state of the art methods on more complex reinforcement learning tasks.

2. Separating the optimization of network topologies and weights into two distinct but connected optimization problems can yield superior results. This is beneficial for more complex problems. This paper presents a novel hybrid algorithm for neural network optimization using genetic algorithms and differential evolution.

3. The NEAT algorithm has very fast initial convergence. This is beneficial when a limited amount of computational resources is available.

4. The ESP algorithm is a good all round neuroevolution algorithm for both the cart pole problem and watershed problem.
5. This research demonstrates the suitability of neuroevolution methods to the watershed management problem, in particular the proposed NDE algorithm.

3.7.1 Future Work

The performance of each of the neuroevolution approaches evaluated in this research are heavily reliant on correct parameter selection. It would therefore be beneficial to either reduce the number of parameters that must be selected or to find parameters that perform well regardless of the problem at hand. As previously mentioned, it would also be beneficial to explore increasing the consistency of these neuroevolution algorithms.

Another logical extension of the research presented in this research would be to apply NDE to problems with a large number of inputs and outputs, such as Atari games [36] and image classification [368]. The NEAT algorithm was extended to the hyperNEAT algorithm [346] which has given promising results in learning Atari games with raw pixel input [159].
Chapter 4

A Multi-Objective Neural Network Trained with Differential Evolution

The work outlined in this chapter was published in:


4.1 Introduction

The Dynamic Economic Emission Dispatch (DEED) problem [30] is a dynamic multi-objective optimisation problem. The aim of this problem is to optimise a set of power generators over a period of time in a manner that both minimizes: 1) The power generation operating cost and 2) The emission of harmful atmospheric pollutants. The task of power generation is critical for modern society to function. It is crucial that electricity is generated in a cost-effective and environmentally responsible fashion. Power generator scheduling is a highly complex task due to the many factors that influence the power generation process. There are a number of constraints including: 1) The generator operation limits. 2) The generator ramp limits. 3) Balancing the power demand and network losses. Generators also have varying levels of efficiency
in terms of cost and emissions produced, thus making the problem multi-objective. Variation in the power demand over time makes the problem dynamic, i.e. the optimal configuration for the power generators at time $t$ is no longer optimal at time $t + 1$. For utility companies to operate effectively, it is imperative that these power generators are scheduled efficiently. Large increases in running costs would be incurred due to suboptimal power generator scheduling. In recent years, many countries have pledged to reduce their carbon footprint [370]. As a result, utility companies must consider the environmental cost of generating electricity in addition to the financial cost. The emission of harmful atmospheric pollutants such as sulphur dioxide ($SO_2$) and nitrogen oxide ($NO$) must be kept to a minimum when scheduling power generators.

When searching for an optimum solution that optimises multiple objectives, it is soon apparent that there is no single optimum solution that optimises all of the objectives. The field of multi-objective optimisation is instead concerned with finding a range of solutions that optimises each objective to a different degree. This set of solutions is known as the Pareto optimal set, where each solution is considered to be equally optimal. The DEED problem is also dynamic in nature due to the changing power demand from hour to hour. The field of dynamic optimisation is concerned with the optimisation of a dynamic objective function, i.e. one that changes with time. There are three main factors to consider when addressing dynamic optimisation problems: 1) Discrete vs continuous time. 2) Deterministic vs stochastic change. 3) Finite vs infinite time horizon. Neural networks have proven to be an effective control method for these dynamic optimisation problems. Neural networks are function approximators that are inspired by the biological brain and are commonly used in machine learning research [47]. They operate by reading in a signal through an input layer of neurons, this signal is then propagated through the network by weighted connections to subsequent hidden layers of neurons. Neural networks can be either fully connected or partially connected. Fully connected networks have the advantage of being easy to implement as there is no need to select a network topology. The disadvantage is that fully connected networks have more weights that need to be optimised than a partially connected network with the same number of neurons. Partially connected networks have the advantage of having fewer weights to train (and therefore less complexity), have improved generalization and have reduced hardware requirements in physical implementations [117]. This research will explore the performance of partially connected networks for the DEED problem.
4.1. Introduction

This research proposes a novel Multi-Objective Neural Networks trained using Differential Evolution (MONNDE). Differential Evolution (DE) is a state of the art global optimisation algorithm [348]. The effectiveness and robustness of DE makes it a suitable choice for training the weights of the neural network. DE is however a single objective optimisation algorithm, although there have been multi-objective variants of DE proposed e.g. Pareto-frontier Differential Evolution (PDE) [3]. Although the aim in multi-objective optimisation is to find multiple solutions that are in the Pareto front, the aim here is to find the single set of network weights that can output a Pareto front depending on the current state of the environment and current objective weight.

The proposed MONNDE algorithm will be applied to the Dynamic Economic Emission Dispatch (DEED) problem [30]. Many algorithms have been applied to the DEED problem in the literature, however the vast majority of these methods are purely optimisation algorithms and therefore do not produce any approximate functions that are capable of producing solutions the optimisation problem. The focus of optimisation algorithms is to find the optimum configuration of the problem variables to maximize/minimize an objective function. The distinction between the proposed MONNDE and previous approaches in the literature is that MONNDE learns to produce solutions on demand whereas previous approaches view the DEED problem purely as an optimisation problem. Once the power demand changes, a multi-objective optimisation algorithm would have to be reapplied to optimise the power generators for the new power demand. This is not the case for the proposed MONNDE algorithm. After the initial training period, no further optimisation is needed for any changes to the power demand. MONNDE builds a function approximator that incorporates the problem characteristics which optimisation algorithms do not do. In short, the MONNDE algorithm applies an optimisation algorithm to the neural network which is used to adjust the problem variables for DEED. Multi-objective optimisation algorithms optimise the DEED problem variables directly. Of course there are many examples in the literature of neural networks being used for economic dispatch [96]. The difference between MONNDE and previous studies is that MONNDE evolves networks capable of producing Pareto fronts for multi-objective problems such as DEED. There are no such examples of this in the literature. The research presented in this paper demonstrates that this is in fact a valid approach. The results presented later show that MONNDE performs on a par with state of the art multi-objective algorithms and can produces Pareto fronts for new power demands with no further optimisation
Chapter 4. A Multi-Objective Neural Network Trained with Differential Evolution

after the initial training period.

This research will also investigate how the proposed multi-objective neural network controllers perform in an online learning environment, i.e. when the environment is susceptible to drastic changes. The research presented in this paper is at the intersection of a number of research areas: multi-objective optimisation, dynamic optimisation, evolutionary computing, neural networks and energy generation. The contributions of this paper are as follows:

1. The design of a novel Multi-Objective Neural Network trained with Differential Evolution (MONNDE) algorithm that can produce a Pareto front for dynamic multi-objective problems.

2. A novel fitness function is proposed that incorporates a Pareto penalty function to help the network to successfully produce a Pareto front at each time step.

3. To compare the performance of both fully and partially connected neural networks for producing the Pareto front

4. To investigate dynamically selecting the network topology as the network is being optimised.

5. To apply the proposed MONNDE to the Dynamic Economic Emission Dispatch problem for both offline and online learning, i.e. when the fitness function dramatically changes in the form of a power generator failure.

6. To investigate the scalability of MONNDE to different size problems.

7. To test MONNDE with new power demands after the initial training period.

The rest of this paper is structured as followed: Section 2 provides a more detailed background into the literature on Multi-Objective Optimisation, Neural Networks, Partially Connected Neural Networks and Differential Evolution. Section 3 outlines the Dynamic Economic Emission Dispatch (DEED) problem that is used to evaluate the proposed MONNDE algorithm. Section 4 describes how the MONNDE algorithm is implemented. The experimental methodology is outlined in Section 5. The results of the experiments are presented in Section 6. Finally, Section 7 draws conclusions based on these results and outline potential future research.
4.2 Background

This section will start by giving a brief overview of the relevant literature on Multi-Objective Optimisation and its applications. This will be then followed by an overview of Neural Networks followed by Partially Connected Neural Networks. The section will finish with a description of both Differential Evolution and neural network topology and weight optimisation.

4.2.1 Multi-Objective Optimisation

Multi-objective optimisation is a sub discipline within optimisation research that explores problems with two or more objectives. These problems have an additional element of complexity due to the conflict that arises when optimising multiple objectives. As the objectives are optimised, there comes a point where by improving upon one object will result in the deterioration of another objective. In multi-objective optimisation problems, the goal is to find a range of solutions, where each solution optimises the different objective with a varying level of significance. They are all considered equally optimal as long as they optimise at least one of the objectives better than any other solution. These optimal solutions are referred to as Pareto optimal solutions [209]. A more strict definition of Pareto optimality states that a solution \( \vec{x} \) is Pareto optimal if there exists no other acceptable solution \( \vec{y} \) which would improve upon the fitness of one objective and not result in the detriment of the fitness of another objective. Mathematically this can be described as solution \( \vec{u} = (u_1, ..., u_n) \) is said to dominate solution \( \vec{v} = (v_1, ..., v_n) \) if \( \forall i \in \{1, ..., n\}, u_i \leq v_i \land \exists i \in \{1, ..., n\} : u_i < v_i \).

The Pareto optimal set can be defined as \( P^* := \{x \in \Omega | \neg \exists x' \in \Omega \vec{f}(x') \preceq \vec{f}(x) \} \) where \( \Omega \) represents the feasible set of solutions and \( \vec{f} \) is the vector of objective functions.

The Pareto Front can be defined as \( PF^* := \{\vec{u} = \vec{f} = (f_1(x), ..., f_k(x)) | x \in P^* \} \).

A comprehensive overview of multi-objective optimisation can be found the work of Coello et al. [78].

The multi-objective framework has proven to be very popular in recent years due as it acknowledges that many real world problems have multiple objectives. A subset of these real world problems include: stock portfolio management [4], software construction management [105], supply chain simulation [104] and design [239]. Some of the most prominent multi-objective optimisation algorithms include: Non-dominated Sorting Genetic Algorithm (NSGA-II) [92], Pareto-frontier Differential Evolution (PDE)
Chapter 4. A Multi-Objective Neural Network Trained with Differential Evolution

algorithm [3], Multi Objective Particle Swarm Optimisation (MOPSO) variant [79], Pareto Archive Evolutionary Strategy (PAES) [213] and Strength Pareto Evolutionary Algorithm (SPEA) [407].

4.2.2 Neural Networks

The field of Neural Networks (NN) is a key area of Machine Learning. Neural networks are function approximators inspired by the biological brain [47, 160] and have been successfully applied to a diverse range of problems such as classification, regression, control, online and offline learning and robotics. The standard feed forward network consists of an input layer of neurons, one or more hidden layer of neurons and finally an output layer of neurons. A neuron is a signal processing unit that takes a number of signals as input and then outputs a signal using the sigmoid function. The network receives environmental information in the form of a normalized signal into the input layer. For the power generation problem that will be explored in this problem, the environmental input signal consists of the hourly power demand. This signal is propagated forward through the connected layers of neurons via weighted synapses (or connections). The network then outputs a signal through the output layer of neurons. In terms of the power generation problem, this output signal corresponds to the power output of each generator. This research will implement a particular type of neural network known as a Recurrent Neural Network (RNN), illustrated in Figure 4.1. Recurrent networks differ from the standard feed forward networks in that they have recurrent connections between hidden neurons. These recurrent connections differ from the other network connections as they connect neurons within the same hidden layer. This gives the system memory which makes them particularly well suited to the power generation problem.

A signal is passed into the network through the input layer of neurons. This signal is then passed through the layers of hidden neurons and is then outputted from the final output layer. This process is referred to as a forward pass. Recurrent connections retain memory as a result of the hidden neurons retaining information from the previous forward pass.

As the signal is propagated through the network, its strength is adjusted by the weights between neurons. Aside from the input layer, a neuron in any other layer will have as input, the sum of the weighted signals that are outputted from other
4.2. Background

Figure 4.1: Recurrent Neural Network. This figure depicts the structure of a fully connected recurrent neural network.

connected neurons. A neuron’s input is described in Equation 4.1.

\[ v_j = \sum_{i=1}^{N} w_{i,j} a_i \]  

(4.1)

Where \( v_j \) is the input to a neuron in the \( j^{th} \) layer, layer \( i \) is the preceding layer to \( j \) that contains \( N \) neurons, each neuron in layer \( i \) has output \( a_i \) and each of these output signals are weighted by the value \( w_{i,j} \) as they are passed to each neuron in layer \( j \).

Each neuron \( a_i \) outputs a value between 0 and 1. This output value is determined by the activation function of the neuron. The most commonly used activation function is the sigmoid function outlined in Equation 4.2.

\[ a_j = \frac{1}{1 + \exp(-v_j)} \]  

(4.2)

4.2.3 Partially Connected Neural Networks

One disadvantage of using Fully Connected Neural Networks (FCNNs) potentially unnecessary connections in the network. Each connection that must be optimised adds an extra dimension to the optimisation problem. Removing these unnecessary connections and reducing the complexity of the optimisation problem would therefore
be incredibly advantageous. This forms the main motivation for Partially Connected Neural Networks (PCNN). Partially connected networks are networks in which some of the weighted connections are removed. This provides a number of advantageous [117]: 1) Increased generalization; 2) Reduced hardware requirements in physical implementations; 3) Step closer to biological reality; 4) Reduction in number of weights to be optimised and therefore reduced complexity. It is this reduction in the number of weights to train that is of main interest in this research. Partially connected neural networks have been applied to a range of problems including: stock price forecasting [67], breast cancer detection [35] and flight control [387]. There has been limited research however evaluating both PCNNs and FCNNs in a multi-objective environment.

4.2.4 Differential Evolution

Differential Evolution (DE) is a state of the art global optimisation algorithm that operates using the principles of evolutionary computing. The DE algorithm was first proposed by Storn and Price in 1997 [348]. An advantage of DE over more traditional gradient based optimisation methods is that DE does not rely on any gradient information and is applicable to noisy problems. This makes DE less susceptible to converging onto a local minimum than gradient based methods. The resilience and effectiveness of DE makes it a suitable choice for optimising network weights.

At each iteration, the current agent’s position (solution) is combined with three other distinct agents’ positions to produce a new position $y_i$. If the new position has a better fitness than the previous position, the previous position is replaced. This process is repeated for all agents for a predetermined number of problem evaluations. Algorithm 5 outlines the algorithms operation where $CR$ is the crossover probability and $F$ is the differential weight.

One of the primary contribution of this research is the presentation of a novel Multi-Objective Neural Network trained with Differential Evolution (MONNDE). There are numerous examples in the literature of successful applications of DE to neural network weight training for applications such as tire contact patch prediction [103] and signal prediction [288]. The purpose of this research is to build on these successful applications and train a neural network using DE for multi-objective problems. There are also examples in the literature of evolving neural networks in multi-objective...
Initialize N agents with random positions $\vec{x}$

\begin{algorithm}
\textbf{while} evaluation $e < E_{\text{max}}$ \textbf{do}
\begin{algorithmic}
\State Select 3 other agents A, B and C at random
\State Select random dimension index $R$
\For{\text{dimension } i = 1 \text{ to } D}
\State generate random number $r \in [0,1]$
\If{$r < CR \text{ Or } i = R$}
\State new position $y_i = a_i + F \times (b_i - c_i)$
\Else
\State $y_i = x_i$
\EndIf
\EndFor
\State getFitness($\vec{y}$)
\If{fitness($\vec{y}$) $< \text{fitness}(\vec{x})$}
\State replace $\vec{x}$ with $\vec{y}$
\EndIf
\EndFor
\EndWhile
\Return best solution
\caption{Differential Evolution (DE) Algorithm}
\end{algorithm}

Algorithm 5: Differential Evolution (DE) Algorithm

environments for games [324] and hybrid fuel cell turbine control [80]. This research however makes the distinct contribution of evolving the neural network to produce the Pareto front, which these previous studies do not.

### 4.2.5 Neural Network Topology and Weight Optimisation

A combined evolutionary strategy is utilized to optimise the neural network’s topology and weight. A variant of the NDE (Neuro Differential Evolution) algorithm is implemented here [242]. This algorithm uses a genetic algorithm to optimise the topology of the network (thus producing PCNNs with fewer connections) and differential evolution to optimise the weights of the network. The use of differential evolution in the NDE algorithm makes it the obvious choice for network topology and weight optimisation in this research as it allows for the direct comparison of topology and weight optimisation versus weight optimisation.

The network topology is encoded into a chromosome that the genetic algorithm can optimise. A chromosome of length 6 (illustrated in Figure 4.2) corresponds to a network that has 6 weights if fully connected. A 1 in the chromosome means that a connection between two neurons is enabled. A 0 implies that the connection is disabled. If the network is fully connected, the chromosome will consist of all 1s. Each bit in the chromosome has an associated weight value. These weights have a value of 0 where the connection is disabled. NDE’s encoding scheme is illustrated
graphically in Figure 4.2. This figure illustrates a PCNN with 3 connections enabled and 3 disabled. When applied to the DEED problem, which will be discussed in Section 4.3, the network will have N-1 outputs (where N is the number of generators) and 3 inputs for the objective weight, current and previous power demands. This will be further explained in Section 4.4.

In order for new network typologies to be created, NDE uses the standard evolutionary algorithms operators: 1) Selection. 2) Crossover. 3) Mutation. Two of the fittest network topologies are selected using a roulette wheel selection method. Crossover is then performed on these selected parent networks as illustrated in Figure 4.3. The bits in these chromosomes are then mutated with probability $m_p$. These new chromosomes are then added to the next generation and then evaluated. Each chromosome is evaluated by optimising its weights using differential evolution in order to maximize/minimize some objective. This is necessary because optimising the topology and weights of a network is a two part optimisation problem. Optimising the topology of the network is a discrete optimisation problem which is done using a GA. However the fitness of any given topology depends on how its set of weights are configured. This is a real valued optimisation problem and is done using DE. With regards to the DEED problem, the objective corresponds to minimizing cost and emissions. This process is repeated in order to find the best network topology and set of weights for that topology.

The variant of the NDE algorithm implemented in this research will not optimise the size of the network, i.e. the number of hidden neurons. The reason for this is to ensure a fair comparison between weight optimisation (DE) and topology and weight optimisation (NDE). Therefore a constant number of hidden neurons was used in all experiments. NDE is implemented here to simply optimise the connectivity of a fixed size network.

The pseudocode in Algorithm 6 outlines how this variant of the NDE algorithm optimises the topology and weight of the neural network. The number of chromosomes $c = 3$, the mutation probability of each bit in the chromosome $m_p = 0.1$, the number of DE agents was equal to the number of connections in the current network (as in all other simulations) and finally the number of iterations the DE algorithm ran for was equal to $10 \times$ the number of network connections. This is less than the $10^6$ used in all other experiments because NDE combines DE with a GA. Therefore the NDE algorithm as a whole will run for the same $10^6$ evaluations as DE.
To summarize, the NDE algorithm is utilized here to evolve PCNNs in a dynamic manner during run time (rather than simply selecting a topology before the weights are optimised). This relates to the DEED problem as the aim is to evolve a network capable of scheduling power generators to minimize cost and emissions. A network produced by NDE is evaluated by outputting a number of values between 0 and 1 for a given input (power demand). Each of the outputs correspond to a generator’s power output. The generator’s output is selected using the output of the network. For example, a network output of $o_i = 0.5$ will correspond to a generator power output of 60 MW, if that generator has a minimum power output of 20 MW and maximum of 100 MW. So for a $N = 10$ generator system, the network will have N-1 outputs (-1 for the slack generator, discussed in Section 4.3). Each of these network outputs $o_i$ will be used to adjust the power generators outputs for the current power demand that the network receives. This is repeated for each power demand over 24 hours. The fitness of the network is determined based on cumulative sum of the fitness evaluations of the power generators at each hour. This fitness evaluation also takes into account the various constraints associated with the DEED problem (discussed in Section 4.3).
4.3 Dynamic Economic Emission Dispatch

The trade-off between cost and emissions is an important factor that must be accounted for in power generation problems. Objectives are conflicting if improving upon one objective will result in the detriment of the other objective. The problem that will be evaluated in this research is the Dynamic Economic Emission Dispatch (DEED) problem. The DEED problem is both dynamic and has multi-objectives. It is therefore suitable to evaluate the performance of the proposed MONNDE algorithm as MONNDE is specifically designed for dynamic multi-objective problems. The problem consists of multiple constraints which increases the problem complexity. These include both hard and soft constraints. The DEED problems constraints include: the power demand, the generator operation limits and ramp limits. The problem involves optimising the scheduling of a number of power generators over a period of time in a manner that minimizes both cost and emissions [30].
4.3. Dynamic Economic Emission Dispatch

Initialize:
Create c chromosomes

\textbf{while} Eval < maxEvals \& fitness < threshold \textbf{do}

\textbf{for} Chromosome c = 1 \textbf{to} C \textbf{do}

Apply DE to weights of c (Alg. 5)

if Fitness < bestFitness then

\quad bestChromosome = c

end if

end

ADD bestChromosome to next gen

\textbf{while} nextGenSize < currentGenSize \textbf{do}

SELECT parents for mating

Perform crossover

MUTATE child chromosome with prob, mp

ADD child chromosome to next gen

end

end

Return best Network

\textbf{Algorithm 6:} Neuro Differential Evolution (NDE)

Previous multi-objective approaches to power generation include evolutionary based approaches such as Non-dominated Sorting Genetic Algorithm-II [30], Modified Adaptive Multi-Objective Differential Evolution [190], Evolutionary Programming / Fuzzy Logic [32] and Genetic Algorithms [95]. There are many instances of swarm based approaches applied to DEED including Bee Colonies [138], Ant Colonies [59], Modified Artificial Bee Colony Algorithm [327], Fuzzy based Bacterial Foraging [172], Multi-Objective Bacterial Colony [230], Parallelized Particle Swarm Optimization [154] and Group Search Optimiser [153]. Previous studies have found adversarial approaches to perform well on the DEED problem. Examples of these include: Teaching Learning Based Optimization [279], Quasi-Oppositional Teaching Learning Based Optimization [313], Opposition-Based Harmony Search [69] and Game Theory [283]. There are also many hybrid approaches in the literature combining methods such as Particle Swarm Optimization and Differential Evolution [144] and Hybrid Particle Swarm Optimization [189].

Recently machine learning has become a popular area of research in energy systems. In 2017 Goudarzi et al. proposed a hybrid mathematical approach for predicting the behavior of the power grid using a least square support vector machine [146] and cultural algorithm [145]. In 2013 Dieu et al. applied Quadratic Programming and Hopfield Networks to the single objective Economic Dispatch problem [96]. These successful applications of machine learning to energy systems further validate the use of neural networks in this paper.

Another popular area of research within the power generation literature is the use of
Chapter 4. A Multi-Objective Neural Network Trained with Differential Evolution

renewable energy [220]. Many studies have developed models that incorporate wind energy [219], solar energy [17], tidal energy [275] and battery storage [389] into their simulations. These will not be considered in this research however. This paper will only consider thermal power generators as there is a wealth of literature using this simulator design that will enable the results presented in this paper to be compared with many state of the art algorithms.

4.3.1 Problem Definition

The cost function $f_1$ in Equation 4.3 represents the hourly running cost of all power generators.

$$f_1 = \sum_{i=1}^{N} [a_i + b_iP_{im} + c_iP_{im}^2 + d_i\sin\{e_i(P_{min}^i - P_{im})\}]$$ (4.3)

Where $m$ is the current hour (maximum of $m = 24$), $N$ refers to the number of power generators, $a_i, b_i, c_i, d_i$ and $e_i$ are all cost coefficients associated with each generator $i$, the power output from generator $i$ at time $m$ is defined as $P_{im}$ and the minimum possible power of generator $i$ is defined as $P_{i min}$. Equation 4.4 represents the emissions function $f_2$ that determines the amount of harmful pollutants produced by all power generators per hour.

$$f_2 = \sum_{i=1}^{N} [\alpha_i + \beta_iP_{im} + \gamma_iP_{im}^2 + \eta\exp\delta P_{im}]$$ (4.4)

Here $\alpha_i, \beta_i, \gamma_i, \eta$ and $\delta_i$ are the emission coefficients associated with each generator $i$. All solutions are subject to the equality constraint in Equation 4.5 that the total power output must be equal to the sum of the power demand and transmission loss.

$$\sum_{i=1}^{N} P_{im} = P_{Dm} + P_{Lm}$$ (4.5)

Where $P_{Dm}$ represents the total power demand at time $m$ and $P_{Lm}$ represents the power loss within the transmission lines at time $m$.

There are two inequality constraints which all power generator configurations are subject to, the operating limits for each power generator and the ramp limits for each power generator. The operating limits specify the maximum and minimum possible
power output of each generator. The ramp limit of a generator specify the maximum allowed increase or decrease in the power output of that generator between two time steps. The operating limits for each generator are defined in Equation 4.6.

\[ P_{\text{min}}^i \leq P_{im} \leq P_{\text{max}}^i \]  \hspace{1cm} (4.6)

Here \( P_{\text{max}}^i \) and \( P_{\text{min}}^i \) refer to the maximum and minimum power output of each generator, \( i \in N \) and \( m \in M \). The ramp limits of each generator are outlined in Equation 4.7.

\[ P_{im} - P_{i(m-1)} \leq U R_i \]  \hspace{1cm} (4.7a)
\[ P_{i(m-1)} - P_{im} \leq D R_i \]  \hspace{1cm} (4.7b)

Here \( UR_i \) and \( DR_i \) are the ramp up and ramp down limits for each generator respectively, \( i \in N \) and \( m \in M \).

As a result of the equality constraint described by Equation 4.5, the first power generator, \( i = 1 \), will be the slack generator. The slack generator will be used to react to the fluctuations in power demand between each hour. The slack generator will therefore be a dependant variable during the optimisation process while the other \( N - 1 \) generators will be independent variables that the neural network can directly control. The power level of the first generator, \( P_{1m} \), is given by Equation 4.8.

\[ P_{1m} = P_{Dm} + P_{Lm} - \sum_{i=2}^{N} P_{im}. \]  \hspace{1cm} (4.8)

The power loss in the transmission lines between generators, \( P_{Lm} \), is calculated as outlined in Equation 4.9.

\[ P_{Lm} = \sum_{i=2}^{N} \sum_{i=2}^{N} P_{im} B_{ij} P_{jm} + 2P_{1m}(\sum_{i=2}^{N} B_{i1}P_{im}) + B_{11}P_{1m}^2. \]  \hspace{1cm} (4.9)

Where \( B \) is the transmission line loss coefficient matrix [30]. By combining Equation 4.8 with Equation 4.9, the quadratic equation (Equation 4.10) is obtained.
\begin{align*}
0 &= B_{11}P_{1m}^2 + (2 \sum_{i=2}^{N} B_{i1}P_{im} - 1)P_{1m} + \\
&\quad (P_{Dm} + \sum_{i=2}^{N} \sum_{j=2}^{N} P_{im} B_{ij} P_{jm} - \sum_{i=2}^{N} P_{im}) \quad (4.10)
\end{align*}

Solving this quadratic equation using basic algebra will give the power output of the slack generator, \( P_{1m} \), at each hour. All values for the cost coefficients, ramp limits, generator capacity limits, power demands and transmission line loss B matrix can be found in the Appendix. Each neural network output represents a power generator output at a given time. The slack power generator is not a network output as it is a reactive variable calculated using Equation 4.10. The inputs of the network are the current power demand and time step \( t \) and the previous power demand at time step \( t - 1 \) normalised between -1 and 1. The third input will be the objective weight which will be outlined later.

### 4.3.2 Constraint Handling

Balancing the power generated with the power demand and power loss is self-constrained due to the slack power generator which will bridge the gap between the power generated and the power demand and transmission losses (Equation 4.8).

The power generator operating limits for the 9 non-slab generators (Equation 4.6) will be handled by normalizing the network outputs between the maximum and minimum possible generator outputs. This will ensure that the output of any generator will be within its acceptable operating region.

The slack generator’s operating limits constraint and ramp limit of each power generator constraint will be enforced using the static penalty method [340] outlined in Equation 4.11. This penalty function will be incorporated into the objective function in order to train the network to avoid invalid solutions.

\begin{align*}
f_v &= \sum_{i=1}^{D} C(|h_i + 1|\delta_i) \quad (4.11)
\end{align*}

Were \( D = 11 \) is the number of constraints per hour handled using this method (1 slack generator operation limits and 10 generator ramp limits), \( C = 10E6 \) is the violation
constant, \( h_i \) is the violation of each constraint and \( \delta = 0 \) if there is no constraint violation in a given dimension and \( \delta = 1 \) if a constraint is violated. The violation constant \( C = 10\times10^6 \) was selected so that violations would have a significant impact on the fitness of the solution produced by the network.

### 4.4 MONNDE Algorithm

As stated in the introduction, the main motivation of the proposed MONNDE algorithm is to provide a model based approach to address dynamic multi-objective optimisation problems. With regards to the DEED problem, when the power demand changes, a multi-objective optimisation algorithm would have to be once again applied to optimise the power generators for the new power demand. This is not the case for the proposed MONNDE algorithm. After the initial optimisation of the weights of the network, no further optimisation is needed and the network is capable of addressing any new power demands. This section outlines how the neural networks are evolved to produce the Pareto front. This is followed by the proposal of a novel Pareto penalty function to help enforce Pareto optimality of the set of solutions produced.

The implementation of the proposed MONNDE algorithm is outlined in the pseudocode in Algorithm 7 and the flowchart in Figure 4.4. The first portion of the algorithm is the same as the standard DE algorithm (Algorithm 5). The distinction between MONNDE and standard DE is in the getFitness() function. As previously explained, an agent's position in MONNDE corresponds to a set of network weights. This position is then passed into the get fitness function which then assigns the current agent's position (i.e. set of network weights) to the network. This function then loops through the objective weight \( w \) from 0 to 1, so that a range of solutions can be obtained, i.e the Pareto front. Within this loop, the function loops through all \( M \) training states. In the case of the DEED problem, this corresponds to a 24 hour time period. At each iteration, the current power demand, previous power demand and \( w \) are inputted into the network to get \( N-1 \) outputs for \( N-1 \) generators. These network outputs, in the range \([0,1]\), are used to select the generators' power outputs between their maximum and minimum possible values. These generator outputs are then evaluated using the fitness function. The cumulative fitness over all \( w \) values over 24 hours is then returned as the fitness of the network and therefore the fitness of
the DE current agent. The agent then updates its position if the new position has a better fitness. A new position is then generated for the next agent and the process is repeated until a predetermined number of problem evaluations have been completed and the best network is found for scheduling the power generators. The fitness of the network is evaluated as follows for the DEED problem:

**Step 1)** The network receives the current objective weight $w$ and the current training state $i$ and the previous state $i-1$. In the case of the DEED problem the state $i$ is the power demand at time $i$. Note here that if the training state $i = 1$ and there are no previous power demands, the current power demand ($i$) is passed in as input twice.

**Step 2)** The network performs a forward pass using its 3 inputs. This means that the input signal is propagated through the network via weighted connections and hidden neurons. The network will then output a series of values between 0 and 1. For the DEED problem, there will be $N-1$ outputs for the $N-1$ non slack power generators. The slack generator’s power output will be calculated as previously discussed in the DEED section of the paper.

**Step 3)** The network’s outputs are then applied to the DEED problem variables, i.e. the generators’ power outputs. Since the power generators have a maximum and minimum power output $P_i^{max}$ and $P_i^{min}$, the range of each generator is therefore $r_i = P_i^{max} - P_i^{min}$. The generator outputs can then be calculated using $P_i = (r_i \times o_i) + P_i^{min}$ where $o_i$ is the $i^{th}$ output of the network in the range of $[0, 1]$. Note that if the power output violates a constraint such as the ramp limit, the network weights will be penalized using the penalty function and less likely to make it through to subsequent generations.

**Step 4)** Involves evaluating the current solution, i.e. configuration of power generators $\vec{P}_i$ (using the objective function in Equation 4.12 or the proposed objective function in Equation 4.13).

**Step 5)** Assigns the objective function evaluation as the current fitness (curFit).

**Step 6)** CurFit is added to networkFit which sums up all of the fitnesses for each training state for each objective weighting. This fitness is then passed back to the DE algorithm which continues to operate in its usual manner.

It should also be noted here that this algorithm can also easily be combined with the
NDE algorithm outlined in Algorithm 6. Multiple networks are created in NDE which have different topologies determined by the current chromosome. The step “Apply DE to weights of c” in Algorithm 6 can therefore utilize the “getFitness()” function from Algorithm 7 to train a multi-objective neural network.

Initialize $N$ agents with random positions $\vec{x}$

while evaluation $e < E_{\text{max}}$ do
  for Agent = 1 to $N$ do
    Select 3 other agents A, B and C at random
    Select random dimension index $R$
    for dimension = 1 to $D$ do
      generate random number $r \in [0,1]$
      if $r < CR$ Or $i = R$ then
        new position $y_i = a_i + F \times (b_i - c_i)$
      else
        $y_i = x_i$
      end if
    end
    getFitness($y$)
    if fitness($y$) < fitness($x$) then
      replace $x$ with $y$
    end if
  end
end
Return best solution

getFitness(agentPosition $s$)
network weights = $s$
networkFit = 0
for objective weight $w = 0$ to 1 do
  for training state $i = 1$ to $M$ do
    1) Pass $w$ and state $i$ into network
    2) Get network output for $w$ and $i$
    3) Adjust power generator outputs using network outputs
    4) Evaluate current generator configuration using objective function
    5) currentFit = fitness of current generator configuration
    6) networkFit += currentFit
  end
end
Return networkFit

Algorithm 7: MONNDE Algorithm
Figure 4.4: MONNDE Algorithm Flowchart
4.4. MONNDE Algorithm

4.4.1 Standard Fitness Function

The primary contribution of this research is to evolve neural networks capable of producing the Pareto optimal front. The network will receive 3 inputs and have N-1 outputs. The 3 inputs correspond to: 1) The current power demand. 2) The power demand at the previous time step. 3) The current objective weight value \( w \). The weight \( w \) will change from 0 to 1 in increments of 0.1. This will train the network to produce the Pareto front of solutions that vary the importance of cost and emissions. The N-1 outputs correspond to the non slack power generators. The values of N-1 = 4, 9 and 14 for the 5, 10 and 15 unit systems.

In order for the DEED problem to be optimised, the cost, emissions and constraint violations penalty functions will be combined using a linear combination to form a single objective function to be minimized [307]. By varying the value of the objective weight \( w \), the neural network will be trained to produce the Pareto front. The objective function in Equation 4.12 is the first objective function that will be evaluated.

\[
F = w f_1 + (1 - w) \lambda f_2 + f_v
\]

The fitness function \( F \) is the hourly fitness function, where \( f_1 \) is the cost function, \( f_2 \) is the emissions function, \( f_v \) is the constrain violation penalty function (Equation 4.11), \( w \) is the weight and \( \lambda \) is the scaling factor [30]. The purpose of \( \lambda \) is to ensure each objective has equal influence. By conducting parameter sweeps it was found that the best values for \( \lambda = 3, 10 \) and 2 for the 5, 10 and 15 unit systems. It should be noted here that the constraint violation function is not part of the Pareto front and therefore is not assigned any weighting value. The purpose of this function is to get the constraint violations to 0.

4.4.2 Proposed Fitness Function

This section will outline the second key contribution of this research, the proposal of a novel penalty function to further encourage the evolution of a network that produces the Pareto front. The fitness function outlined in Equation 4.12 puts evolutionary pressure on the optimisation algorithm to produce a network that does not have any constraint violations. This is achieved adding a constraint violation penalty to any solution that violates these constraints. It is true to say that this fitness function
also encourages the optimisation algorithm to evolve networks capable of producing
the Pareto front by varying $w$. The proposed fitness function, outlined in Equation
4.13, introduces a novel Pareto penalty function that increases the emphasis put on
the optimisation algorithm to produce networks that generate the Pareto front.

$$F = w f_1 + (1 - w) \lambda f_2 + f_v + f_p$$  \hspace{1cm} (4.13)

Where the Pareto penalty function $f_p$ is outlined in Equation 4.14.

$$f_p = \sum_{j=1}^{P} D(\lvert g_1 \rvert + 1) \delta_1 + (\lvert g_2 \rvert + 1) \delta_2$$  \hspace{1cm} (4.14)

The values of $g_i$ and $\delta$ are outlined in Equation 4.15.

$$g_i = f_{i,j} - f_{i,j+1}$$  \hspace{1cm} (4.15a)

$$\delta_1 = \begin{cases} 
0, & \text{if } f_{1,j} \geq f_{1,j+1} \\
1, & \text{otherwise}
\end{cases}$$  \hspace{1cm} (4.15b)

$$\delta_2 = \begin{cases} 
0, & \text{if } f_{2,j} \leq f_{2,j+1} \\
1, & \text{otherwise}
\end{cases}$$  \hspace{1cm} (4.15c)

In Equations 4.14 and 4.15, $D$ is the Pareto violation constant, $g_i$ is the difference in
objective $f_i$ between two subsequent objective weight $w$ values (e.g. $w = 0.1$ at $j = 1$
and $w = 0.2$ at $j = 2$), $\delta = 0$ if there is no Pareto violation and $\delta = 1$ if there is.

The weightings on each objective $w$ changes from 0 to 1 in increments of 0.1 to produce
$P$ Pareto points. As $w$ increases, the cost objective $f_1$ receives more of a weighting
and should decrease. The opposite is true for the emission objective $f_2$. The purpose
of the Pareto penalty function in Equation 4.14 is to enforce this more strongly by
applying a harsh penalty when the cost doesn’t decrease (or when the emissions don’t
increase) as $w$ is increased. For this objective function, it was found that $D = 10E6$
and $C = 10E8$ (from Equation 4.11) gave the best performance. This was determined
by conducting parameter sweeps.

To summarize the purpose of each objective functions in Equation 4.13: $f_1$ is the
4.5. Experimental Procedure

cost objective function that represents the running cost of each generator, \( f_2 \) is the emission objective function that represents the emissions produced by each generator, \( f_v \) is the constraint violation penalty function which has the purpose of penalizing any solution that has any of the aforementioned constraint violations and finally \( f_p \) is the proposed Pareto penalty function which has the purpose of enforcing Pareto optimality in the set of solutions produced by the network.

4.4.3 Parameter Selection

Each iteration will consist of evaluating 24 distinct states (power demands) as the objective weight \( w \) is varied from 0 to 1 for each state. The fitness of a network corresponds to the cumulative fitness \( F \) (Equation 4.12 or 4.13) over the range of \( w \) for 24 hours. The DE crossover probability \( CR = 0.9 \) and differential weight \( F = 0.5 \), as determined by parameter sweeps.

The MONNDE variant for optimizing the network topology and weights was implemented with the following parameters: Chromosome Mutation probability \( mp = 0.4 \), chromosomes \( C = 3 \), differential evolution maximum number of iterations \( Imax = 5 \times con \), (where \( con \) is the number of connection weights in the current network configuration), differential evolution crossover probability \( CR = 0.9 \), differential weight \( F = 0.5 \) and finally the number of differential evolution agents \( N = con, (N = 4 \text{ if } con < 4) \). Parameter tuning revealed that these parameters gave the best performance. In all 5 and 10 generator unit simulations, a network with 4 hidden neurons was implemented. For the 15 unit simulations, a network with 3 hidden neurons was implemented. Parameter tuning revealed these gave the best performance.

4.5 Experimental Procedure

This section details the experiments conducted in this research. The MONNDE algorithm was evaluated over the 6 experiments. The test system is the DEED problem as stated in Section 4.3. All coefficients relating to its implementation can be found in the Appendix. Experiment 1 involves implementing the MONNDE algorithm with a standard fitness function. Experiment 2 investigates the performance of the proposed fitness function with the Pareto penalty function. In experiment 3 the performance of fully and partially connected networks are evaluated. Experiment 4 involves testing
the neural network training algorithms in an online learning environment to evaluate how they perform when there is a drastic change to the environment. The distinction between offline and online learning environment is that during the neural network training process, the problem environment is not subject to change in an offline learning environment. In online learning, the problem environment is subject to drastic change during network training, e.g. a power generator failure. A 10 unit system [30] is considered for experiments 1 - 4. Experiment 5 looks at the scalability of MONNDE by evaluating a 5 [29] and 15 [298] unit system. The final experiment investigates the performance of the evolved networks over a new set of power demands for all test systems. All simulations were averaged over 10 runs, each run consisting of 10⁶ evaluations. A large number of evaluations was necessary due to the high complexity of the problem spaces in each experiment. Parameter sweeps determined that a recurrent neural network with 4 hidden neurons performed best for the 5 and 10 unit systems, while 3 hidden neurons performed best for the 15 unit system (due to the larger number of outputs). The simulator was developed using the Java programming language.

4.5.1 Experiment 1 - Bench Marking MONNDE

The MONNDE algorithm with standard fitness function is implemented on a 10 unit generator system. The performance of MONNDE will be judged based on cost, emissions, constraint violations and Pareto set coverage with respect to other approaches in the literature. The aim of comparing MONNDE with other approaches is ensure it can provide reasonable performance when compared to the state of the art.

4.5.2 Experiment 2 - Proposed Fitness Function

Next the proposed fitness function with the Pareto penalty function will be evaluated. This experiment will analyze the hourly Pareto fronts and compare the new fitness function with the standard fitness function. The purpose of this is to ensure that the evolved network can produce acceptable Pareto fronts.
4.5.3 Experiment 3 - Partially Connected Neural Networks

The next experiment involves investigating the performance of evolving partially connected networks for producing the Pareto front. There are many benefits associated with partially connected networks most notably their reduced complexity compared to a fully connected network of equal size. This experiment will firstly establish if some of these connections are redundant and if so, how many can be removed before the networks ability to produce the Pareto front deteriorates.

In each of the previous experiments, a fixed network topology was implemented for all evaluations of the problem. This required the user to select the network topology in advance before optimising the network. This raises the question of whether or not it is possible to select the topology of the network dynamically during the optimisation process to avoid the additional computations required to determine the best network topology before optimising the weights. This experiment explores this question and aims to investigate the performance of neural network topology and weight optimisation versus neural network weight optimisation for producing the Pareto front. The NDE (Neuro Differential Evolution) algorithm will be implemented for Neural Network Topology and Weight Optimisation (NN-TWO) during the network training process.

4.5.4 Experiment 4 - Generator Failure

This experiment aims to evaluate the performance of each algorithm in an online learning environment. This experiment will involve simulating a power generator failure halfway through optimisation process. After $5 \times 10^5$ evaluations (using the 10 unit system), generator 7 can no longer output any power. This experiment will evaluate how well each algorithm can react to this failure by retraining the network to output more power from the other generators and produce a new Pareto front. This will determine how well each algorithm operates in an online learning environment and evaluate how robust it is.

4.5.5 Experiment 5 - Scalability

In all previous experiments, MONNDE was evaluated on the 10 unit system. This experiment will evaluate its performance a 5 unit and 15 unit system. The purpose
of this experiment is to demonstrate the suitability of MONNDE for different size problems.

4.5.6 Experiment 6 - New Power Demands

The final experiment in this paper will test out the performance of the MONNDE algorithm when it is passed new previously unseen power demand data. The main advantage of evolving a neural network to schedule power generators, as opposed to directly optimising the generators using an optimisation algorithm, is that a model of the problem has been developed after the network is trained. 24 hours of new power demand data will be passed into the network to establish if the network can successfully produce a Pareto front with new unseen data and no further optimisation. This would demonstrate the unique advantage that MONNDE has over all purely optimisation based approaches. In order for these methods to produce a Pareto front for new data, the entire optimisation process needs to be re done. This is not true for MONNDE.

4.6 Results

This section presents the results of each of the experiments outlined above followed by a discussion in order to gain insight and highlight their significance. The two tailed t test with a significance level of $\alpha = 0.05$ was conducted to determine significant performance differences when comparing algorithms. All values were rounded to 4 decimal places. In order to evaluate the Pareto front produced by the MONNDE algorithm, it will be compared to the Pareto fronts produced by each other algorithm using the Two Set Coverage approach [406]. This method involves comparing each point ($a_1$) in the Pareto front of one set ($S_1$) with the corresponding point ($a_2$) in the second set ($S_2$) by calculating the CS score. Equation 4.16 outlines how the CS score is calculated.

$$CS(S_1, S_2) = \frac{|\{a_2 \in S_2, \exists a_1 \in S_1 : a_1 \prec a_2\}|}{|S_2|} \tag{4.16}$$

The CS metric is used to compare Pareto fronts in this paper because it does not require any knowledge of the true Pareto front. As this is the case for the DEED
problem, it is a suitable metric to evaluate the Pareto front produced by MONNDE. A $CS(S_1, S_2) = 1$ score implies that the set $S_1$ completely dominates all of the solutions in $S_2$. Conversely a $CS(S_1, S_2) = 0$ implies that all solutions are non-dominant. It is necessary to calculate both $CS(S_1, S_2)$ and $CS(S_2, S_1)$ as the CS metric is asymmetric.

4.6.1 Standard Fitness Function

The Pareto front produced by the MONNDE algorithm with the standard fitness function is illustrated in Figure 4.5. This clearly demonstrates that an evolutionary neural network based approach is applicable to dynamic multi-objective optimisation problems.

![Average 24 Hour Pareto Front](image)

Figure 4.5: Average Pareto Front Over 24 Hours.

Table 4.1 outlines the locations of the hourly Pareto fronts produced by the MONNDE algorithm. This table gives the $w=0.5$ cost and emissions for the full 24 hours. This highlights how the location of the Pareto front changes depending on the current power demand. The large hourly variation in cost and emission is a direct result of the variation in the power demand. The advantage of MONNDE over traditional multi-objective optimisation algorithms is its ability to produce hourly Pareto fronts without any further optimisation after the initial training period. The proposed MONNDE algorithm develops a model that approximates the problem rather than directly optimising the set of problem variables. This means that after the initial training period, the neural network is capable of producing Pareto optimal fronts without any further training.

Table 4.2 presents the average performance of the MONNDE algorithm compared to
Table 4.1: Locations of Hourly Pareto Fronts For 24 Hours \((w = 0.5)\) - 10 Generator System

<table>
<thead>
<tr>
<th>Hour</th>
<th>Cost ($)</th>
<th>Emissions (lbs)</th>
<th>Hour</th>
<th>Cost ($)</th>
<th>Emissions (lbs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64580</td>
<td>3925</td>
<td>13</td>
<td>144392</td>
<td>21730</td>
</tr>
<tr>
<td>2</td>
<td>67685</td>
<td>4290</td>
<td>14</td>
<td>125683</td>
<td>16602</td>
</tr>
<tr>
<td>3</td>
<td>75012</td>
<td>5372</td>
<td>15</td>
<td>112732</td>
<td>12997</td>
</tr>
<tr>
<td>4</td>
<td>84496</td>
<td>6869</td>
<td>16</td>
<td>94772</td>
<td>8911</td>
</tr>
<tr>
<td>5</td>
<td>89200</td>
<td>7794</td>
<td>17</td>
<td>88879</td>
<td>7962</td>
</tr>
<tr>
<td>6</td>
<td>98954</td>
<td>10165</td>
<td>18</td>
<td>98871</td>
<td>10205</td>
</tr>
<tr>
<td>7</td>
<td>106883</td>
<td>11371</td>
<td>19</td>
<td>114154</td>
<td>12825</td>
</tr>
<tr>
<td>8</td>
<td>111519</td>
<td>13124</td>
<td>20</td>
<td>132446</td>
<td>18024</td>
</tr>
<tr>
<td>9</td>
<td>125601</td>
<td>16871</td>
<td>21</td>
<td>127127</td>
<td>16568</td>
</tr>
<tr>
<td>10</td>
<td>138789</td>
<td>19673</td>
<td>22</td>
<td>100880</td>
<td>10149</td>
</tr>
<tr>
<td>11</td>
<td>149113</td>
<td>26507</td>
<td>23</td>
<td>79843</td>
<td>6193</td>
</tr>
<tr>
<td>12</td>
<td>156440</td>
<td>26392</td>
<td>24</td>
<td>71354</td>
<td>4779</td>
</tr>
</tbody>
</table>

Table 4.2: Average 24 Hour Cost and Emissions \((w = 0.5)\) - 10 Unit System

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cost ((\times 10^6)$)</th>
<th>Emissions ((\times 10^5)) lbs</th>
<th>CS(MONNDE, ALG)</th>
<th>CS (ALG, MONNDE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONNDE</td>
<td>2.5600</td>
<td>2.9782</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SPSO [241]</td>
<td>2.6044</td>
<td>3.1075</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>PSOAWL [241]</td>
<td>2.5463</td>
<td>2.9455</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NSGA-II [30]</td>
<td>2.5226</td>
<td>3.0994</td>
<td>0.33</td>
<td>0</td>
</tr>
<tr>
<td>IBFA [292]</td>
<td>2.5171</td>
<td>2.9904</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RCGA [30]</td>
<td>2.5251</td>
<td>3.1246</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MAMODE [190]</td>
<td>2.5141</td>
<td>3.0274</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CRO [314]</td>
<td>2.5178</td>
<td>3.0194</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HCRO [314]</td>
<td>2.5171</td>
<td>2.9907</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MARL [238]</td>
<td>2.6641</td>
<td>3.3255</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MODE [403]</td>
<td>2.5276</td>
<td>2.9805</td>
<td>0</td>
<td>0.33</td>
</tr>
<tr>
<td>MOHD-SAT [403]</td>
<td>2.5280</td>
<td>2.9776</td>
<td>0</td>
<td>0.66</td>
</tr>
</tbody>
</table>

When compared to other approaches, MONNDE performs better than Standard Particle Swarm Optimisation (SPSO) [205] and Multi Agent Reinforcement Learning (MARL) [237], worse than Particle Swarm Optimisation with Avoidance of Worst Locations (PSOAWL) [249, 251] and Multi-Objective Hybrid Differential Evolution with Simulated Annealing Technique (MOHD-SAT) [403] and equal the other 7 algorithms. These are the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [92], Improved Bacterial Foraging Algorithm (IBFA) [292], Real Coded Genetic Algorithm (RCGA) [30], Multi-Objective Differential Evolution (MODE) [403], Modified Adaptive Multi-Objective Differential Evolution (MAMODE) [190], Chemical Reaction Optimisation (CRO) [314] and Hybrid Chemical Reaction Optimisation (HCRO) [314]. The CS scores on the right of Table 4.2 show how the Pareto front produced by
MONNDE compares to those produced by other methods. The Pareto front produced by MONNDE totally dominates that of SPSO and MARL. Conversely the MONNDE Pareto front is totally dominated by that of PSOAWL and mostly dominated (66.6%) by MOHD-SAT. The MONNDE Pareto front is partially covered by that of MODE (33.3%). Conversely the NSGA-II Pareto front is partially covered by that of MONNDE (also 33.3%). MONNDE is non dominant, i.e., equally optimal, when compared to all other approaches outlined in Table 4.2. To summarize Table 4.2, in terms of $w=0.5$ solutions, MONNDE is better than 2 algorithms, worse than 2 algorithms and equal to the remaining 7 algorithms. In terms of Pareto front coverage, MONNDE completely covers 2 algorithms and partially covers 1 algorithm. Conversely it is completely covered by 1 algorithm and partially covered by 2. In all other cases the Pareto fronts do not overlap. This demonstrates that the evolving multi-objective neural networks for DEED is highly competitive when compared to state of the art optimisation algorithms and worthy of further research.

The proposed algorithm is similar to Multi Agent Reinforcement Learning (MARL) [237] in the sense that each approach involves building a model that is used to control the power generators. It is therefore worth highlighting the performance differences between these two approaches in particular. Unlike the purely optimisation approaches, these methods aim to develop controllers for the power generators that can produce a Pareto front when given new power demands without requiring any further optimisation. The proposed MONNDE performs significantly better than MARL in terms of both cost, emissions and Pareto front coverage. This further validates the choice of evolutionary neural networks to develop multi-objective controllers for the DEED problem. An interesting topic for future research would explore combining the evolutionary neural network approach used in this paper with reinforcement learning.

Table 4.3 displays the best 24 hour power generator configuration produced by MONNDE for an equal cost/emissions weighting, i.e. $w = 0.5$. The value $w = 0.5$ was selected as it is standard in the DEED literature to display solutions with equal cost and emissions weighting. In Table 4.3, the unit of power P is the megawatt (MW), cost is $\times 10^6$ and emissions are $lb \times 10^5$. This table clearly demonstrates how MONNDE discovered that it is favorable to run certain power generators at maximum capacity constantly. It was found it to be optimal for power generators 7, 8, 9 & 10 to be maximising their outputs at every hour. This is consistent with previously observed results [30, 241]. This indicates that these generators are very efficient in
120Chapter 4. A Multi-Objective Neural Network Trained with Differential Evolution
Table 4.3: Best Generator Configuration for w = 0.5 - 10 Unit System
Hour

P1

P2

P3

P4

P5

P6

P7

P8

P9

P10

Cost (×106 $)

Emissions (×105 lbs)

Violations

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24

150.2762
150.6279
163.8728
197.6648
219.3916
208.6499
251.0270
231.7690
299.7127
355.4711
390.3014
403.4567
363.6337
306.2713
272.2708
223.3948
206.1325
206.9657
252.1108
327.4518
316.2439
236.4204
157.9773
154.0609

135.0031
136.8644
153.8094
173.5217
188.3629
223.4399
246.9358
279.3663
323.4421
356.4639
386.5048
412.4308
368.2273
324.3332
282.9989
225.5800
198.4161
222.4484
269.7886
335.0268
328.6778
249.8149
191.9544
143.6907

80.7232
77.4870
100.7837
132.6547
151.9216
219.4039
233.4224
274.4045
306.1999
318.8008
334.7264
339.7676
338.3652
302.5844
254.7106
184.9084
159.5296
220.4481
266.9176
314.1410
296.2510
216.3113
136.7495
113.8324

64.2789
105.0596
150.8308
181.1736
194.3452
239.9075
240.0617
262.4476
278.0750
283.5437
294.6690
299.6242
298.4758
274.4749
242.4942
202.8712
194.0168
241.2619
259.4475
282.9539
267.1620
217.5391
168.0403
121.7714

124.0503
133.9260
181.7517
215.4537
223.6141
240.7662
240.1517
242.3962
242.8725
242.9230
242.9126
242.3097
242.8167
242.7609
238.8945
220.8792
220.3583
240.9967
242.2622
242.9412
242.3509
217.9751
181.1928
158.6211

116.5711
143.5705
150.7945
156.8708
157.9165
159.8926
159.8161
159.9768
159.9969
159.9981
159.9971
159.9140
159.9921
159.9921
159.5957
156.3593
157.0017
159.9092
159.9707
159.9989
159.9646
154.6742
143.5306
132.5438

130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000
130.0000

120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000
120.0000

80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000
80.0000

55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000
55.0000

0.0621
0.0660
0.0749
0.0847
0.0898
0.0983
0.1065
0.1111
0.1264
0.1398
0.1498
0.1561
0.1442
0.1270
0.1139
0.0948
0.0896
0.0982
0.1122
0.1328
0.1282
0.1016
0.0797
0.0701

0.0386
0.0425
0.0539
0.0691
0.0781
0.1021
0.1136
0.1313
0.1658
0.1971
0.2350
0.2629
0.2164
0.1654
0.1281
0.0882
0.0780
0.1023
0.1298
0.1796
0.1650
0.1007
0.0613
0.0475

0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0

24 Hour

Cost

2.5579 ×106

Emission

2.9522 ×105

terms of the cost and emissions.

4.6.2

Fitness Function with Pareto Penalty

This section presents the results of the proposed fitness function with Pareto penalty
and compare it to that of the standard fitness function from the previous section.
The drawback to using the standard fitness function presented in the previous section
is that it does not enforce the hourly solutions in the Pareto front to be Pareto
optimal. This is illustrated in Figure 4.6. This graph demonstrates how simply using
the standard fitness function is not sufficient to train a neural network capable of
producing Pareto optimal sets. It can be clearly seen that many of the solutions
produced by the network when trained using the standard fitness function are not
part of the Pareto optimal set as they are dominated by other solutions in the set.
In contrast, the proposed fitness function with Pareto penalty always produced a
Pareto front where the solutions are non-dominated with respect to one another. On
average the standard fitness function resulted in the MONNDE algorithm converging
with 64.7 Pareto violations while the proposed fitness function with Pareto penalty
resulted in 0 Pareto violations. The definition of a Pareto violation here is when one
of the solutions in the hourly Pareto front is dominated by another solution. These
results motivate the need for the proposed fitness function as the standard fitness
function is not sufficient to produce hourly Pareto fronts at every hour as Figure 4.6
illustrates.
Figure 4.7 does show however that by using the standard fitness function, the evolved


4.6. Results

Figure 4.6: Hour 12 Pareto Front - 10 Unit System

neural network is still capable of producing an acceptable hourly Pareto front on some time steps. The proposed fitness function is much more consistent however and produces acceptable Pareto fronts at every time steps. This is due to the evolutionary pressure that the proposed fitness function applies on the network to avoid producing solutions that are dominated with respect to other solutions in the set. From Figure 4.6 it is clear that the old fitness function produces many solutions that are dominated and therefore the set produced is not a Pareto front. This observation validates the new fitness function. Another observation that can be made from Figure 4.7 is that the Pareto fronts from the old and new fitness function are at different locations. Neither fronts intersect or dominate one another. This means that the are equally optimal and approximate the Pareto front at different locations in the solution space.

Figure 4.7: Hour 14 Pareto Front - 10 Unit System

The proposed fitness function with novel Pareto penalty does significantly increase the ability of MONNDE to produce the Pareto front. Simply implementing the standard fitness function used in previous studies will produce solutions with no DEED constraint violations, but does not provide enough of an incentive for DE to evolve
a network which produces the Pareto front. The proposed Pareto penalty places evolutionary pressure to evolve a network that produces set of solutions that are non-dominated with respect to one another.

### 4.6.3 Partially Connected Neural Networks

A Fully Connected Neural Network (FCNN) was evolved in each of the previous experiments. This section will present the performance results of both a Partially Connected Neural Network (PCNN) with a static network topology and also a PCNN in which the network topology is dynamically changed during the optimisation process.

As illustrated in Figure 4.8, there are both benefits and drawbacks associated with FCNNs and PCNNs. The PCNN provides much faster convergence than a FCNN and can converge with a network that performs statistically equal to the FCNN. The PCNN here corresponds to a network with 85% connectivity, i.e. 15% of the connections were randomly removed. This was the maximum amount of connections that could be removed before the network began to produce solutions that had Pareto and constraint violations. It should also be highlighted that every algorithm evaluated here produced no Pareto or constraint violations. It can also be seen in Figure 4.8 that the when the topology and weights were optimised (NN-TWO), much slower convergence was observed. This is due to the additional computational burden required to optimise the topology dynamically. The benefits of a PCNN include, faster convergence, less complexity and good overall performance (statistically equal to the FCNN). The negatives associated with the PCNN is the need to select the network topology. Optimising the topology and weight of the network dynamically is much slower to converge to an optimal configuration.
When comparing the performance between a fully connected and partially connected network, there are advantages and disadvantages of both methods. The fully connected network is straightforward to implement and performs very well in terms of its final fitness score. Conversely the partially connected network has fewer connections (and therefore less complexity), converges faster and to a fitness score statistically equal to the fully connected network. Figure 4.9 (a) and (b) shows the best and average 24 hour Pareto front produced by a FCNN (blue) and a PCNN (red). Figure 4.9 (a) shows that the best Pareto front produced by a PCNN completely covers that of a FCNN. Figure 4.9 (b) however shows that the FCNN covers the PCNN on average. This indicates that the PCNN is not consistent in its performance. The suspected reason for this is that when producing the PCNN, 15% of the network connections are randomly removed. In some instances (such as in Figure 4.9 (a)) this is clearly advantageous. In others however this is disadvantageous. This implies that the choice of connections removed is important. Selecting which connections to remove and which to include is not an easy task however. As Figure 4.8 shows, the NN-TWO approach which utilizes the NDE algorithm to select the network connectivity during run time is far slower to converge. This figure shows that the NN-TWO approach is still converging. With more evaluations, it is possible that this approach can produce more competitive solutions with more time. If producing a smaller network is very important for a particular problem and there is a lot of computational time available, optimising the topology and weight of the network during run time would be a desirable option. If not, selecting the topology before optimising the weights of the network would be a better approach.
4.6.4 Generator Failure

This results section will explore the performance of each of the MONNDE variants in an online learning environment. In each of the previous experiments a network was trained to produce the Pareto front in an offline manner, i.e. there was no change to the environment. This section will explore the performance of each MONNDE variant in an online learning environment where changes can occur. This experiment will test the adaptability of each algorithm by simulating a power generator failure halfway through the training process. Figure 4.10 illustrates the fitness convergence of each MONNDE variant when a power generator fails midway through the training process. As this graph demonstrates, dynamically optimising the topology and weights of the neural network during run time in an online learning environment leads to significantly better performance. This is in stark contrast to the results observed in the previous experiment. The additional computational burden of optimising the network topology is a hindrance in offline learning as it leads to slower convergence. However in an online learning environment where the environment is subject to change, the additional computational burden of optimising the network topology leads to continuous exploration and enables the network to better adapt to a changing environment. The reason for this is that the NN-TWO approach continuously generates new networks and optimises their weights. This enables the NN-TWO approach to quickly abandon the previous network that was optimal before the generator failure and find new networks that work better in the system with a failed generator. This graph indicates that when applying DE to a static network topology, the algorithm converges to an optimum solution early on and does not have capability to find effective network weights after a generator fails. It is worth noting here that none of the MONNDE variants were able to converge to a network configuration without any Pareto or constraint violations. The MONNDE variant in which the topology was dynamically optimised (NN-TWO) had far fewer violations however.

Table 4.4 displays the best 24 hour power generator configuration produced by MONNDE for an equal cost/emissions weighting, i.e. $w = 0.5$ when power generator 7 fails. The units in Table 4.4 are the same as those in Table 4.3. This is the optimum solution that the MONNDE algorithm with dynamic network topology optimisation could produce after generator 7 fails.

The results of the experiments relating to offline training clearly indicate that optimising a neural network with a static network topology using differential evolution is
4.6. Results

Figure 4.10: Fitness Convergence with Power Generator Failure - 10 Unit System

Table 4.4: Best Generator Configuration for $w = 0.5$ - 10 Unit System With Generator Failure

<table>
<thead>
<tr>
<th>Hour</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>P9</th>
<th>P10</th>
<th>Cost ($ \times 10^6$)</th>
<th>Emissions ($ \times 10^5$ lbs)</th>
<th>Violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>157</td>
<td>156</td>
<td>148</td>
<td>148</td>
<td>150</td>
<td>135</td>
<td>128</td>
<td>141</td>
<td>141</td>
<td>141</td>
<td>157.5942</td>
<td>86.4523</td>
<td>0.0689</td>
</tr>
<tr>
<td>2</td>
<td>196</td>
<td>163</td>
<td>161</td>
<td>161</td>
<td>160</td>
<td>145</td>
<td>134</td>
<td>145</td>
<td>145</td>
<td>145</td>
<td>237.2716</td>
<td>53.8189</td>
<td>0.0749</td>
</tr>
<tr>
<td>3</td>
<td>219</td>
<td>202</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>187</td>
<td>178</td>
<td>187</td>
<td>187</td>
<td>187</td>
<td>270.7752</td>
<td>58.0865</td>
<td>0.0846</td>
</tr>
<tr>
<td>4</td>
<td>192</td>
<td>206</td>
<td>202</td>
<td>202</td>
<td>202</td>
<td>192</td>
<td>187</td>
<td>187</td>
<td>187</td>
<td>187</td>
<td>267.5193</td>
<td>56.0140</td>
<td>0.0934</td>
</tr>
<tr>
<td>5</td>
<td>187</td>
<td>206</td>
<td>202</td>
<td>202</td>
<td>202</td>
<td>192</td>
<td>187</td>
<td>187</td>
<td>187</td>
<td>187</td>
<td>207.7753</td>
<td>59.2844</td>
<td>0.0987</td>
</tr>
<tr>
<td>6</td>
<td>221</td>
<td>227</td>
<td>220</td>
<td>220</td>
<td>220</td>
<td>210</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>230.8411</td>
<td>61.3800</td>
<td>0.1106</td>
</tr>
<tr>
<td>7</td>
<td>245</td>
<td>250</td>
<td>244</td>
<td>244</td>
<td>244</td>
<td>235</td>
<td>216</td>
<td>216</td>
<td>216</td>
<td>216</td>
<td>238.8411</td>
<td>63.7075</td>
<td>0.1164</td>
</tr>
<tr>
<td>8</td>
<td>267</td>
<td>273</td>
<td>266</td>
<td>266</td>
<td>266</td>
<td>257</td>
<td>238</td>
<td>238</td>
<td>238</td>
<td>238</td>
<td>242.6388</td>
<td>66.1722</td>
<td>0.1231</td>
</tr>
<tr>
<td>9</td>
<td>311</td>
<td>297</td>
<td>290</td>
<td>290</td>
<td>290</td>
<td>272</td>
<td>253</td>
<td>253</td>
<td>253</td>
<td>253</td>
<td>256.6388</td>
<td>67.6122</td>
<td>0.1377</td>
</tr>
<tr>
<td>10</td>
<td>351</td>
<td>356</td>
<td>351</td>
<td>351</td>
<td>351</td>
<td>342</td>
<td>324</td>
<td>324</td>
<td>324</td>
<td>324</td>
<td>359.0509</td>
<td>69.5492</td>
<td>0.1502</td>
</tr>
<tr>
<td>11</td>
<td>436</td>
<td>452</td>
<td>446</td>
<td>446</td>
<td>446</td>
<td>430</td>
<td>411</td>
<td>411</td>
<td>411</td>
<td>411</td>
<td>469.5924</td>
<td>71.5492</td>
<td>0.1640</td>
</tr>
<tr>
<td>12</td>
<td>479</td>
<td>496</td>
<td>489</td>
<td>489</td>
<td>489</td>
<td>475</td>
<td>457</td>
<td>457</td>
<td>457</td>
<td>457</td>
<td>499.8962</td>
<td>73.5962</td>
<td>0.1746</td>
</tr>
<tr>
<td>13</td>
<td>399</td>
<td>403</td>
<td>398</td>
<td>398</td>
<td>398</td>
<td>386</td>
<td>368</td>
<td>368</td>
<td>368</td>
<td>368</td>
<td>422.9727</td>
<td>75.6145</td>
<td>0.1847</td>
</tr>
<tr>
<td>14</td>
<td>355</td>
<td>349</td>
<td>346</td>
<td>346</td>
<td>346</td>
<td>337</td>
<td>320</td>
<td>320</td>
<td>320</td>
<td>320</td>
<td>371.6526</td>
<td>77.6526</td>
<td>0.1942</td>
</tr>
<tr>
<td>15</td>
<td>297</td>
<td>291</td>
<td>287</td>
<td>287</td>
<td>287</td>
<td>274</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>318.3852</td>
<td>79.6526</td>
<td>0.2042</td>
</tr>
<tr>
<td>16</td>
<td>222</td>
<td>215</td>
<td>207</td>
<td>207</td>
<td>207</td>
<td>200</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>193</td>
<td>237.6150</td>
<td>81.6526</td>
<td>0.2143</td>
</tr>
<tr>
<td>17</td>
<td>200</td>
<td>194</td>
<td>186</td>
<td>186</td>
<td>186</td>
<td>180</td>
<td>173</td>
<td>173</td>
<td>173</td>
<td>173</td>
<td>217.6150</td>
<td>83.6526</td>
<td>0.2244</td>
</tr>
<tr>
<td>18</td>
<td>225</td>
<td>218</td>
<td>211</td>
<td>211</td>
<td>211</td>
<td>204</td>
<td>197</td>
<td>197</td>
<td>197</td>
<td>197</td>
<td>233.7176</td>
<td>85.6526</td>
<td>0.2345</td>
</tr>
<tr>
<td>19</td>
<td>237</td>
<td>230</td>
<td>223</td>
<td>223</td>
<td>223</td>
<td>216</td>
<td>209</td>
<td>209</td>
<td>209</td>
<td>209</td>
<td>246.6276</td>
<td>87.6526</td>
<td>0.2445</td>
</tr>
<tr>
<td>20</td>
<td>308</td>
<td>301</td>
<td>294</td>
<td>294</td>
<td>294</td>
<td>287</td>
<td>280</td>
<td>280</td>
<td>280</td>
<td>280</td>
<td>289.6127</td>
<td>89.6526</td>
<td>0.2545</td>
</tr>
<tr>
<td>21</td>
<td>303</td>
<td>296</td>
<td>289</td>
<td>289</td>
<td>289</td>
<td>282</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>292.1990</td>
<td>91.6526</td>
<td>0.2645</td>
</tr>
<tr>
<td>22</td>
<td>226</td>
<td>219</td>
<td>212</td>
<td>212</td>
<td>212</td>
<td>205</td>
<td>198</td>
<td>198</td>
<td>198</td>
<td>198</td>
<td>234.5222</td>
<td>93.6526</td>
<td>0.2745</td>
</tr>
<tr>
<td>23</td>
<td>192</td>
<td>186</td>
<td>179</td>
<td>179</td>
<td>179</td>
<td>172</td>
<td>165</td>
<td>165</td>
<td>165</td>
<td>165</td>
<td>214.6667</td>
<td>95.6526</td>
<td>0.2845</td>
</tr>
<tr>
<td>24</td>
<td>172</td>
<td>166</td>
<td>159</td>
<td>159</td>
<td>159</td>
<td>152</td>
<td>145</td>
<td>145</td>
<td>145</td>
<td>145</td>
<td>207.5193</td>
<td>96.6526</td>
<td>0.2945</td>
</tr>
</tbody>
</table>

24 Hour Cost: $2.8111 \times 10^6$ Emission: $4.1726 \times 10^5$

...a much better approach than optimising both the topology and weights of the network in terms of the fitness of the solutions produced. This is because by just optimising the weights of the network, there is no additional computations wasted optimising the topology of the network. The algorithm just needs to converge on the optimum set of weights and does not need to do any continued exploring. The results of the experiments in this section highlight one of the main advantages of optimising the topology and weights of the network. By optimising the topology and weights of a network, new networks with different topologies are constantly being created and then optimised. This makes the MONNDE variant with topology and weight optimisation particularly well suited to online learning problems. In this experiment, a power generator was set to fail halfway through the neural network training process. This experiment required the MONNDE algorithms to adapt to the new environment where fewer generators are available to meet the power demand. This experiment...
revealed that optimising the topology and weights of the network performs significantly better than just optimising the weights. When a power generator fails, the algorithm is better able to adjust than simply optimising the weights of the network. The MONNDE algorithm with topology and weight optimisation is therefore better able to approximate the Pareto optimal front in live environments with a dynamic environment.

4.6.5 5 and 15 Generator Systems

In each of the experiments conducted so far, the MONNDE algorithm has been evaluated on a 10 unit system. The results presented in this section will evaluate the performance of MONNDE on a 5 and 15 unit system. The purpose of this is to ensure that it maintains a competitive level of performance for problems of different scales. Table 4.5 displays the cost and emissions of each algorithm for the 5 unit system. Similar to the results from the 10 unit system, this table illustrates that MONNDE performs competitively when compared to state of the art optimisation algorithms. MONNDE performs statistically better than Particle Swarm Optimisation (PSO) [29]. MONNDE performs statistically equal to 5 algorithms: Differential Evolution - Sequential Quadratic Programming (DE-SQP) [116], Particle Swarm Optimisation - Sequential Quadratic Programming (PSO-SQP) [116], Evolutionary Programming (EP) [32], Simulated Annealing (SA) [10] and Parallel Search (PS)[10]. Finally MONNDE performs statistically worse than 3 algorithms: Multi-Objective Differential Evolution (MODE) [403], Multi-Objective Differential Evolution with Simulated Annealing Technique (MOHDE-SAT) [403] and New Pitch Adjustment Harmony Search (NPAHS) [281]. This highlights the competitiveness of the MONNDE approach. When comparing the Pareto fronts using the Two Set Coverage approach [406], it is evident that in many cases there is very little coverage between the Pareto front of MONNDE and other algorithms with the exception of MODE, MOHDE-SAT and NPAHS. The Pareto front of MONNDE partially covers that of PSO (33.3%), is partially covered by that of MOHDE-SAT (66.6%) and is totally covered by those of MODE and NPAHS. As observed with the results for the 10 unit system, MONNDE does not outperform every other approach. However this table does highlight that MONNDE performs statistically better than or equal to 6 out of 9 algorithms to which it is compared to, demonstrating that it is a competitive algorithm for the 5 unit system. The best generator configuration produced is presented...
in Table 4.6.

Table 4.5: Average 24 Hour Cost and Emissions ($w = 0.5$) for 5 Generator System

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cost ($)</th>
<th>Emissions (lbs)</th>
<th>CS(MONNDE, ALG)</th>
<th>CS (ALG, MONNDE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONNDE</td>
<td>49884</td>
<td>18647</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DE-SQP [116]</td>
<td>4450</td>
<td>19616</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PSO-SQP [116]</td>
<td>4452</td>
<td>19772</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PSO [20]</td>
<td>50983</td>
<td>20163</td>
<td>0.33</td>
<td>0</td>
</tr>
<tr>
<td>NPAHS [281]</td>
<td>45196</td>
<td>18630</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>EP [32]</td>
<td>48628</td>
<td>21154</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SA [10]</td>
<td>48621</td>
<td>21188</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PS [10]</td>
<td>47911</td>
<td>18927</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MODE [403]</td>
<td>47330</td>
<td>18116</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MOHDE-SAT [403]</td>
<td>48214</td>
<td>18011</td>
<td>0</td>
<td>0.66</td>
</tr>
</tbody>
</table>

Table 4.6: Best Generator Configuration for $w = 0.5$ - 5 Unit System

<table>
<thead>
<tr>
<th>Hour</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>Cost ($)</th>
<th>Emission (lbs)</th>
<th>Violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.805</td>
<td>63.2142</td>
<td>112.3904</td>
<td>106.6944</td>
<td>120.3741</td>
<td>1588.8909</td>
<td>417.7025</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>27.4131</td>
<td>70.5219</td>
<td>109.9417</td>
<td>108.9385</td>
<td>122.0805</td>
<td>1690.6314</td>
<td>431.6243</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>52.8084</td>
<td>74.9252</td>
<td>113.6402</td>
<td>113.9896</td>
<td>124.2075</td>
<td>1757.6421</td>
<td>472.9567</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>74.6243</td>
<td>82.9464</td>
<td>125.0687</td>
<td>125.6892</td>
<td>127.4547</td>
<td>1795.6446</td>
<td>563.3496</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>73.1960</td>
<td>90.7645</td>
<td>136.2656</td>
<td>135.8068</td>
<td>128.3849</td>
<td>1939.4866</td>
<td>614.2476</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>73.5787</td>
<td>93.5005</td>
<td>159.4395</td>
<td>156.0524</td>
<td>133.0499</td>
<td>2138.8723</td>
<td>720.7692</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>73.5608</td>
<td>96.7456</td>
<td>165.2586</td>
<td>163.9215</td>
<td>134.6040</td>
<td>2175.7867</td>
<td>764.7287</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>71.6638</td>
<td>98.0709</td>
<td>172.7141</td>
<td>181.2548</td>
<td>139.1462</td>
<td>2185.4923</td>
<td>840.0473</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>74.3772</td>
<td>98.1411</td>
<td>174.8687</td>
<td>203.9203</td>
<td>148.5771</td>
<td>2211.9898</td>
<td>905.2921</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>73.4534</td>
<td>98.1667</td>
<td>174.9790</td>
<td>213.7032</td>
<td>154.0669</td>
<td>2234.7957</td>
<td>955.2921</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>72.7607</td>
<td>98.1330</td>
<td>174.9981</td>
<td>223.9579</td>
<td>160.9561</td>
<td>2356.8723</td>
<td>1054.8664</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>74.3629</td>
<td>97.7615</td>
<td>174.9999</td>
<td>233.5208</td>
<td>170.7884</td>
<td>2493.5044</td>
<td>1131.0941</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>71.3549</td>
<td>95.8937</td>
<td>174.9896</td>
<td>215.8270</td>
<td>156.2441</td>
<td>2257.5813</td>
<td>1000.5506</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>73.3679</td>
<td>97.4869</td>
<td>174.8972</td>
<td>204.8661</td>
<td>149.2668</td>
<td>2158.9678</td>
<td>948.1610</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>74.5637</td>
<td>97.5761</td>
<td>172.5342</td>
<td>178.5867</td>
<td>139.5825</td>
<td>2145.3737</td>
<td>838.3936</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>74.2145</td>
<td>98.0255</td>
<td>143.7155</td>
<td>142.0552</td>
<td>128.9376</td>
<td>2005.1164</td>
<td>660.1850</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>69.7418</td>
<td>94.5680</td>
<td>136.7653</td>
<td>135.7584</td>
<td>127.5646</td>
<td>1935.5024</td>
<td>615.2870</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>72.3190</td>
<td>93.7304</td>
<td>160.3007</td>
<td>156.3516</td>
<td>132.9195</td>
<td>2141.8859</td>
<td>751.1907</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>73.0911</td>
<td>97.0757</td>
<td>172.4442</td>
<td>180.9130</td>
<td>139.3223</td>
<td>2147.8409</td>
<td>839.1757</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>74.5448</td>
<td>98.0857</td>
<td>174.9730</td>
<td>213.2317</td>
<td>153.7423</td>
<td>2202.8478</td>
<td>993.8403</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>74.0950</td>
<td>97.0570</td>
<td>174.7230</td>
<td>196.9716</td>
<td>146.7380</td>
<td>2168.2209</td>
<td>915.4888</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>74.4132</td>
<td>98.2060</td>
<td>156.3234</td>
<td>152.1687</td>
<td>131.4742</td>
<td>2104.6122</td>
<td>714.2379</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>62.7565</td>
<td>94.0297</td>
<td>122.8740</td>
<td>125.3848</td>
<td>125.8122</td>
<td>1702.1086</td>
<td>562.0723</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>33.8492</td>
<td>84.5942</td>
<td>111.3043</td>
<td>114.5845</td>
<td>123.1039</td>
<td>1674.6846</td>
<td>471.3589</td>
<td>0</td>
</tr>
</tbody>
</table>

The MONNDE algorithm was then applied to a 15 unit system. The purpose of this was to demonstrate the scalability of MONNDE to larger problems. Table 4.7 outlines the average cost and emissions of MONNDE and Particle Swarm Optimisation (PSO) for the 15 unit system. As this table shows, each algorithm here performs equally optimal, i.e. are non dominant with respect to one another. MONNDE produces a solution with lower emissions while PSO produces a solution with lower a cost. When the Pareto fronts of the two algorithms are compared to each other using the Two Set Coverage approach [406], it can be seen that there is no overlap between the two Pareto fronts. This shows that these two methods produce Pareto fronts that do not dominate one another and are truly equally optimal. This is an important result as it demonstrates that MONNDE scales well to larger problems and still performs
competitively. The best solution obtained for a 15 unit system is presented in Table 4.8.

Table 4.7: Average 24 Hour Cost and Emissions \((w = 0.5)\) for 15 Generator System

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cost ($)</th>
<th>Emissions (lbs)</th>
<th>CS(MONNDE, ALG)</th>
<th>CS (ALG, MONNDE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONNDE</td>
<td>711635</td>
<td>273906</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PSO [241]</td>
<td>706128</td>
<td>307726</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

What the experiments in this section demonstrate is that MONNDE is a viable approach for evolving a network capable of producing the Pareto front for a range of problem sizes. In every size of problem evaluated, MONNDE was able to produce Pareto fronts that are comparable with the state of the art. Figures 4.11 (a) and (b) illustrate the Pareto front for the 5 and 15 unit system. In no instances did it produce a solution that had any constraint violations or a Pareto front that had any dominated solutions.

![Figure 4.11: Best Pareto Front Over 24 Hours - 5 and 15 Units.](image)

4.6.6 New Power Demands Evaluation

The final experiment in this research was to evaluate the performance of the evolved network by giving it new unseen power demand data. This in an important experiment because it demonstrates the key advantage of MONNDE over the traditional optimisation based approach. In this experiment, the networks that were trained to produce Pareto fronts for the 5, 10 and 15 unit systems for power demands over 24 hours, were then given a new set of power demands (found in the Appendix) to produce a set of solutions for. What needs to be emphasised here is that no further optimisation of the networks was carried out. Figures 4.12 (a), (b) and (c) represent the 24 hour Pareto front produced by these networks when given this new power demand profile for the 5, 10 and 15 unit systems respectively. This validates the hypothesis...
Table 4.8: Best Generator Configuration for \( w = 0.5 - 15 \) Unit System

<table>
<thead>
<tr>
<th>Hour</th>
<th>Cost ($)</th>
<th>Emission [lbs]</th>
<th>Violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>159.7602</td>
<td>209.4521</td>
<td>150.3802</td>
</tr>
<tr>
<td>2</td>
<td>157.8459</td>
<td>154.6311</td>
<td>150.6261</td>
</tr>
<tr>
<td>3</td>
<td>179.8028</td>
<td>162.5739</td>
<td>125.5078</td>
</tr>
<tr>
<td>4</td>
<td>208.9452</td>
<td>189.1218</td>
<td>129.9900</td>
</tr>
<tr>
<td>5</td>
<td>196.6866</td>
<td>209.0088</td>
<td>129.9814</td>
</tr>
<tr>
<td>6</td>
<td>198.9000</td>
<td>238.6964</td>
<td>129.9800</td>
</tr>
<tr>
<td>7</td>
<td>224.2041</td>
<td>250.3168</td>
<td>129.9896</td>
</tr>
<tr>
<td>8</td>
<td>222.7658</td>
<td>266.5760</td>
<td>129.9992</td>
</tr>
<tr>
<td>9</td>
<td>239.2045</td>
<td>295.2334</td>
<td>129.9997</td>
</tr>
<tr>
<td>10</td>
<td>263.3597</td>
<td>314.3383</td>
<td>129.9998</td>
</tr>
<tr>
<td>11</td>
<td>271.9021</td>
<td>335.9412</td>
<td>129.9998</td>
</tr>
<tr>
<td>12</td>
<td>284.2074</td>
<td>345.2219</td>
<td>129.9998</td>
</tr>
<tr>
<td>13</td>
<td>263.3011</td>
<td>329.3127</td>
<td>129.9998</td>
</tr>
<tr>
<td>14</td>
<td>233.4062</td>
<td>299.9201</td>
<td>129.9998</td>
</tr>
<tr>
<td>15</td>
<td>221.5929</td>
<td>268.5027</td>
<td>129.9998</td>
</tr>
<tr>
<td>16</td>
<td>199.6040</td>
<td>224.8516</td>
<td>129.9982</td>
</tr>
<tr>
<td>17</td>
<td>194.7012</td>
<td>209.9629</td>
<td>129.9982</td>
</tr>
<tr>
<td>18</td>
<td>194.4596</td>
<td>238.7226</td>
<td>129.9981</td>
</tr>
<tr>
<td>19</td>
<td>241.4655</td>
<td>262.3569</td>
<td>129.9994</td>
</tr>
<tr>
<td>20</td>
<td>254.2278</td>
<td>302.8974</td>
<td>129.9998</td>
</tr>
<tr>
<td>21</td>
<td>262.4898</td>
<td>291.5412</td>
<td>129.9998</td>
</tr>
<tr>
<td>22</td>
<td>203.6545</td>
<td>240.6355</td>
<td>129.9925</td>
</tr>
<tr>
<td>23</td>
<td>174.4454</td>
<td>179.9473</td>
<td>129.7907</td>
</tr>
<tr>
<td>24</td>
<td>164.9523</td>
<td>159.6798</td>
<td>125.9084</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>P9</th>
<th>P10</th>
<th>P11</th>
<th>P12</th>
<th>P13</th>
<th>P14</th>
<th>P15</th>
</tr>
</thead>
<tbody>
<tr>
<td>125.4225</td>
<td>77.920</td>
<td>89.2172</td>
<td>150.3802</td>
<td>146.3275</td>
<td>138.2569</td>
<td>60.0000</td>
<td>25.0166</td>
<td>73.6182</td>
<td>80.0000</td>
<td>59.6217</td>
<td>25.3910</td>
<td>15.1413</td>
<td>15.0000</td>
<td>19077.5027</td>
</tr>
<tr>
<td>125.4225</td>
<td>77.920</td>
<td>89.2172</td>
<td>150.3802</td>
<td>146.3275</td>
<td>138.2569</td>
<td>60.0000</td>
<td>25.0166</td>
<td>73.6182</td>
<td>80.0000</td>
<td>59.6217</td>
<td>25.3910</td>
<td>15.1413</td>
<td>15.0000</td>
<td>19077.5027</td>
</tr>
<tr>
<td>125.4225</td>
<td>77.920</td>
<td>89.2172</td>
<td>150.3802</td>
<td>146.3275</td>
<td>138.2569</td>
<td>60.0000</td>
<td>25.0166</td>
<td>73.6182</td>
<td>80.0000</td>
<td>59.6217</td>
<td>25.3910</td>
<td>15.1413</td>
<td>15.0000</td>
<td>19077.5027</td>
</tr>
<tr>
<td>125.4225</td>
<td>77.920</td>
<td>89.2172</td>
<td>150.3802</td>
<td>146.3275</td>
<td>138.2569</td>
<td>60.0000</td>
<td>25.0166</td>
<td>73.6182</td>
<td>80.0000</td>
<td>59.6217</td>
<td>25.3910</td>
<td>15.1413</td>
<td>15.0000</td>
<td>19077.5027</td>
</tr>
<tr>
<td>125.4225</td>
<td>77.920</td>
<td>89.2172</td>
<td>150.3802</td>
<td>146.3275</td>
<td>138.2569</td>
<td>60.0000</td>
<td>25.0166</td>
<td>73.6182</td>
<td>80.0000</td>
<td>59.6217</td>
<td>25.3910</td>
<td>15.1413</td>
<td>15.0000</td>
<td>19077.5027</td>
</tr>
<tr>
<td>125.4225</td>
<td>77.920</td>
<td>89.2172</td>
<td>150.3802</td>
<td>146.3275</td>
<td>138.2569</td>
<td>60.0000</td>
<td>25.0166</td>
<td>73.6182</td>
<td>80.0000</td>
<td>59.6217</td>
<td>25.3910</td>
<td>15.1413</td>
<td>15.0000</td>
<td>19077.5027</td>
</tr>
<tr>
<td>125.4225</td>
<td>77.920</td>
<td>89.2172</td>
<td>150.3802</td>
<td>146.3275</td>
<td>138.2569</td>
<td>60.0000</td>
<td>25.0166</td>
<td>73.6182</td>
<td>80.0000</td>
<td>59.6217</td>
<td>25.3910</td>
<td>15.1413</td>
<td>15.0000</td>
<td>19077.5027</td>
</tr>
<tr>
<td>125.4225</td>
<td>77.920</td>
<td>89.2172</td>
<td>150.3802</td>
<td>146.3275</td>
<td>138.2569</td>
<td>60.0000</td>
<td>25.0166</td>
<td>73.6182</td>
<td>80.0000</td>
<td>59.6217</td>
<td>25.3910</td>
<td>15.1413</td>
<td>15.0000</td>
<td>19077.5027</td>
</tr>
<tr>
<td>125.4225</td>
<td>77.920</td>
<td>89.2172</td>
<td>150.3802</td>
<td>146.3275</td>
<td>138.2569</td>
<td>60.0000</td>
<td>25.0166</td>
<td>73.6182</td>
<td>80.0000</td>
<td>59.6217</td>
<td>25.3910</td>
<td>15.1413</td>
<td>15.0000</td>
<td>19077.5027</td>
</tr>
<tr>
<td>125.4225</td>
<td>77.920</td>
<td>89.2172</td>
<td>150.3802</td>
<td>146.3275</td>
<td>138.2569</td>
<td>60.0000</td>
<td>25.0166</td>
<td>73.6182</td>
<td>80.0000</td>
<td>59.6217</td>
<td>25.3910</td>
<td>15.1413</td>
<td>15.0000</td>
<td>19077.5027</td>
</tr>
</tbody>
</table>

Note: The table continues with similar entries for each hour.
that it is possible to evolve a network that is an effective control system for the DEED problem. The evolved network is capable of building internal representations of the problem characteristics and use these to control the system of power generators.

Table 4.9 shows the cost and emissions for the 5, 10 and 15 unit systems for both the training and test power demands. This table shows that the cost and emission for each set of power demands are in the same order of magnitude. Changes in cost and emissions are due to the new power demands requiring different generator configurations that have different associated costs and emissions.

Table 4.9: MONNDE 24 Hour Cost and Emissions ($w = 0.5$) for Training and Test Power Demands

<table>
<thead>
<tr>
<th>System (No. Units)</th>
<th>Training Cost ($)</th>
<th>Emissions (lbs)</th>
<th>Test Cost ($)</th>
<th>Emissions (lbs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>49884</td>
<td>18647</td>
<td>53259</td>
<td>20117</td>
</tr>
<tr>
<td>10</td>
<td>2559956</td>
<td>297818</td>
<td>2531945</td>
<td>311323</td>
</tr>
<tr>
<td>15</td>
<td>711635</td>
<td>273906</td>
<td>716522</td>
<td>266873</td>
</tr>
</tbody>
</table>

MONNDE has the advantage over other optimisation approaches (such as Non-dominated Sorting Genetic Algorithm (NSGA-II) [92], Pareto-frontier Differential Evolution (PDE) [3], Multi Objective Particle Swarm Optimisation (MOPSO) [79], etc.) in that MONNDE produces an approximate model of the problem after the
optimisation process is complete. This is advantageous for dynamic problems. The proposed MONNDE approach produce a Pareto front with no further training whenever the optimisation problem changes. In the case of the DEED problem, this refers to when the power demand changes.

Another feature of the proposed MONNDE algorithm is that it implements a single objective optimisation algorithm to produce the Pareto front, as it optimises the weights of the network rather than the variables directly. By using a linear combination of the objectives, the network can be trained to produce this set by simply varying the objective weight $w$. It is well known that there are of course disadvantages in using a linear combination of objective weights such as non uniformity in the set of solutions produced and the inability to find some solutions. This approach does however lend itself to the evolution of multi-objective neural networks. Future work would include investigating other methods of combining objectives such as $\epsilon$-constraint.

As would be expected, the computational cost of MONNDE is higher than for purely optimisation based methods. Given that the algorithm must set the weights of a network at each run, pass the power demand into the network and calculate the outputs of each neuron in the network and then change the network output into a power generator setting, it is understandable that it is higher than optimisation methods that do not use neural networks. However the extra computational cost associated with training the network pays off when passing in new power demands to the network after the training period is over. In Experiment 6, the network was able to produce Pareto fronts for new power demands in a fraction of a second. Using optimisation algorithms, the cost of producing new Pareto fronts is equally as high as for the previous set of power demands since they have built no model of the problem. The proposed evolutionary neural network approach could be viewed as a spend more now but save later in terms of computational cost.

Much research has already been conducted applying differential evolution to both feed forward [178] and recurrent neural networks [288]. There has been no research however applying differential evolution to multi-objective neural networks to address dynamic multi-objective problems. There is new research emerging exploring how neural networks can be combined with the multi-objective framework. In 2010, Schrum and Miikkulainen used NSGA-II to evolve a neural network in an environment with multiple objectives [324]. This differs from the research presented in this paper as the goal of this research is to evolve a neural network capable of producing the Pareto
optimal front in a dynamic environment. The work of Schrum and Miikulainen was concerned with evolving a network that produce a single output in a dynamic multi-objective environment rather than the full set. In 2016, Colby et al. explored how multi-objective optimisation could be used to determine a fitness function that can be used as input for a neural network controller [80]. Again the research presented in this paper involves evolving a neural network capable of producing the Pareto front of solutions rather than just a single solution. In 2017, Ravi et al. also explore the use of neural networks in multi-objective problems [303]. This study however utilizes neural networks for the time series prediction of financial data and utilizes that prediction in multi-objective optimisation. The neural network itself is not concerned with the multi-objective nature of the problem. The contribution that the proposed MONNDE makes that is distinct from other previously mentioned studies is that the purpose of MONNDE is to produce Pareto fronts for dynamic multi-objective optimisation problems. Previous studied implemented neural networks in multi-objective environments but did not aim to produce a Pareto front of equally optimal solutions for the decision maker. This is the key contribution of MONNDE.

There are many potential routes for future research that have arisen from this research. The first route would be to evaluate the performance of MONNDE on a wider range of problems and to compare the performance of this method to a greater range of problems. There are many examples of dynamic multi-objective optimisation problems which MONNDE would be applicable to, e.g. Stock portfolio optimisation: This problem involves selecting which stocks to invest capital into to maximize profit and minimize risk [14]. Watershed management: This problem involves allocating water to multiple parties each seeking to maximize their water consumption. It is also a dynamic problem due to the varying levels of water available [245]. Cloud computing: The task of placing virtual machines in the cloud to minimize energy consumption, waiting times and service level agreement violations is a dynamic multi-objective problem [391]. Future work would also include exploring different ways of handling multiple objectives and also to combine the proposed MONNDE framework with other multi-objective optimisation algorithms. This research only evaluated the performance of Differential Evolution to train the multi-objective neural network. Comparing and analyzing the performance of other optimisation algorithms for training multi-objective neural networks would therefore be insightful. An additional route for future research would also include combining MONNDE with other state of the art multi-objective optimisation algorithms by using the solutions provided by the
MONNDE function approximator as a seed or starting place for these optimisation algorithms to build upon.

4.7 Conclusion

The central aim of this research was to investigate evolving neural networks capable of producing the Pareto front for dynamic multi-objective optimisation problems. The results presented in this research clearly indicate that the proposed MONNDE is capable of evolving multi-objective neural networks that are comparable with many state of the art multi-objective optimisation algorithms. The MONNDE algorithm has the added advantage of developing a model of the problem that can be used to produce the Pareto front in a dynamic environment with no further optimisation required. It is also shown that the proposed fitness function with a novel Pareto penalty helps the network to produce the Pareto front. This research also evaluates the benefits of evolving fully and partially connected networks for multi-objective problems. It was observed that a partially connected network with 15% fewer connections can converge faster and perform statistically the same as a fully connected network. It was found that when training the network in an offline manner where the fitness function is not subject to any change, training just the weights of a network using Differential Evolution is advantageous due to its rapid convergence. In an online learning or online environment where the environment is subject to change, such as a generator failure, it is found that optimising the topology and weights of the neural network performs better than simply optimising the network weights. The continuous exploration and evaluation of new networks that topology and weight optimisation provides is the reason for its superior performance in an online learning environment. The scalability of MONNDE is demonstrated for problems of different size. The final experiment demonstrates the primary advantage of MONNDE, i.e. it can produce Pareto fronts for new power demands after training with no further optimisation required.

In summary, the contributions of this research are:

1. A novel Multi-Objective Neural Network evolved with Differential Evolution (MONNDE) is proposed.

2. The proposed fitness function with a novel Pareto penalty function enables the
MONNDE algorithm to approximate the Pareto front.

3. A partially connected network with 15% fewer connections performs statistically equal to the fully connected network and can converge faster.

4. For offline learning, evolving both the topology and weights during run time performs worse due to the computational demand of optimising the network topology.

5. In an online learning environment, evolving the topology and weights of the network during run time provides better performance and can react better to a generator failure.

6. The MONNDE algorithm scales well with no constraint violations for 5, 10 and 15 unit problems.

7. MONNDE can successfully schedule power generators for new power demands with no further optimisation needed.
Chapter 5

Forecasting in Ireland’s Energy Sector

The work outlined in this chapter was published in:


5.1 Introduction

Worldwide there is an ever increasing demand for energy due to population growth, increased living standards and industrial development. This ever increasing appetite for energy gives rise to a number of problems, namely: 1) Producing the energy necessary to meet the power demand, 2) Reducing the harmful atmospheric pollutants that result from the power generation process, 3) Incorporating renewable energy sources into the power generation process. In each of these problems facing the energy sector, it is vital to develop accurate forecasting methods.

When generating power it is crucial to be able to accurately forecast energy demands in both the short term and long term. Long term energy forecasting enables policy makers, planners and engineers to prepare for future energy needs by building and developing infrastructure to generate power. Short term energy forecasting is critical to energy production as it is vital to know how much energy will be needed in the
near future so that power generators can be scheduled to meet future energy needs.

Minimizing the carbon footprint of the power generation process is very important to the energy sector in recent years. It is well known that burning fossil fuels such as coal will produce harmful atmospheric pollutants such as Sulfur Dioxide ($\text{SO}_2$), Nitrogen Oxide ($\text{NO}_x$) and Carbon Dioxide ($\text{CO}_2$). This is a problem as these chemical compounds directly contribute to global warming. In the 2015 Paris Climate Conference (COP21) nearly 200 countries worldwide agreed to cut green house gas emissions in the coming years. In order to reduce the production of these harmful atmospheric pollutants, it is vital to be able to predict how much of these pollutants will be produced.

Due to the harmful atmospheric effects of burning fossil fuels, many countries worldwide have resorted to renewables as a source of energy, e.g. wind, wave and solar energy. The primary drawback with these renewable resources is that they do not provide consistent energy. The fact that renewables do not produce a consistent source of energy is a major hurdle that must be overcome. In order to incorporate these environmentally friendly resources into the power generation process, it is important to know in advance how much energy will be available from these sources. This further motivates the need for accurate forecasting techniques.

Recently neural networks have become a popular machine learning approach for forecasting problems. Neural networks are function approximators inspired by the brain. One of the main design considerations when implementing neural networks is how to optimise the network weights in order to produce the desired output for a given input. Traditionally these weights have been trained using the backpropagation algorithm. In recent years however there has been a large body of research conducted that focuses on the use of evolutionary algorithms to train these neural network weights. One of the most successful evolutionary algorithm is the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) algorithm [157]. This algorithm uses evolutionary principles such as selection and mutation and implements a covariance matrix to represent the dependencies between variables to evolve solutions to complex real valued optimisation problems. CMA-ES has never been applied to forecasting in the energy sector despite its impressive performance. It is hoped that the effectiveness of CMA-ES as an optimisation algorithm can benefit the energy sector by evolving more accurate neural network forecasting models.

The accuracy of a neural network trained using CMA-ES will be judged using Ireland’s
5.2. Energy Forecasting

This section will give an overview of energy demand forecasting, wind power generation forecasting and CO₂ level forecasting. This will include outlining the importance and motivation of each forecasting problem along with the prominent research conducted on each problem.
5.2.1 Energy Demand Forecasting

As the world’s population is continuously increasing, energy providers are faced with the task of generating energy sufficiently to meet this increased power demand. Forecasting electrical load plays a crucial role within the energy sector for problems such as power system planning and management. Load forecasting can be categorized into 3 groups: 1) Short term 2) Medium term and 3) Long term forecasting. Short term forecasting is concerned with time scales of hours (the focus of this research), medium term is concerned with time scales of months and long term is concerned with time scales of years. Each of these are important for different reasons. Accurate short term forecasting is crucial to ensure that an uninterrupted power supply is available [286]. Medium term forecasting is important as it is used to schedule maintenance and repairs of the power generation infrastructure [51]. Finally long term forecasting plays a crucial role in influencing policy and infrastructure planning [196].

Forecasting approaches can be subdivided into two groups: 1) Statistical techniques 2) Artificial intelligence (AI) techniques. As will be discussed later on, AI techniques such as neural networks and support vector machines have the advantage of making fewer assumptions about the data and are therefore more flexible. These are referred to as non-parametric methods. Examples of statistical techniques include linear regression and autoregressive integrated moving average (ARIMA). There are a number of drawbacks with the ARIMA approach [72]. It is more rigid and has lack of generality when there is a significant change in the environment [304]. It is also known that ARIMA models forecast well for a only short period of time after the model has been trained [207]. This research will train the forecast model using data corresponding to a time period of a month and then evaluate the performance of the forecast model on data corresponding to the following month. For these reasons, methods such as ARIMA will not be implemented in this research. A major area of research into forecasting load demand includes incorporating other environmental variables into the forecasting models, e.g. time of day. For predicting wind power generation this could be other meteorological information such as temperature and forecast wind speed [305]. This research will implement a purely univariate approach that solely utilizes historical time series data to make predictions.
5.2.2 Wind Generation Forecasting

Many countries are devoting resources into renewable energy infrastructure as a solution to the harmful effects that arise from burning fossil fuels. In 2015, 22.8% of Ireland’s energy was generated from wind power [82]. Wind power generation poses additional challenges when compared to conventional thermal power generation. This is due to the inherent stochasticity with wind speeds. Forecasting wind power generation accurately is therefore a crucial task when incorporating energy from wind farms to meet power demand. Accurate forecasts of wind power enable better economic dispatch, unit commitment and scheduling of thermal power generators [125]. In terms of economic dispatch, in order for power generators to be scheduled optimally to meet the power demand, it is important to know how much energy the power generators will need to produce in the future. This is because power generators take a significant amount of time to ramp up/down their power output [241]. If a significant amount of energy were to be available from wind turbines in the future, the power generators would need time to reduce their power output. It is therefore vital to have accurate wind power forecasting methods to schedule these generators to account for energy available from wind. This is currently an active area of research.

Wind generation forecasting algorithms can be subdivided into 2 groups: 1) Historical time series prediction methods. 2) Numerical Weather Prediction (NWP) models. NWP models incorporate weather data into the prediction of wind speed [7]. This research will focus on the former of the two groups using only historical time series wind data. AI methods are often referred to as grey box models. Grey box models combine mathematical based approaches with data to form the forecast model. Neural networks are typical examples of grey box models. There are many examples in the literature of applications of neural networks to wind energy forecasting. One such example is applying neural networks to predict wind power using local wind speed data [257]. Another application of neural networks to wind forecasting is implementing a multi neural network approach to first predict wind speed and to then map the wind speed to the power generated [44]. A comprehensive overview of the research surrounding wind power forecasting is given by Foley et al. [125].
5.2.3 Atmospheric Pollutant Forecasting

One of the major factors behind the rapid change in climate in recent years is the industrialization of the modern world. This is expected to increase as developing countries become more industrialized [259]. Climate change is considered to be one of the greatest threats facing modern society [357]. Man made climate change resulting from the emission of carbon dioxide emissions is responsible for many of the negative environmental changes observed in recent years. One of the most well known negative effects of climate change is the bleaching of coral reefs [174]. Increasing sea temperatures is believed to be responsible for this coral reef decline which is a direct result of climate change [227]. This warming of the oceans in turn has resulted in a decline in sea ice [223]. Less sea ice then results in increased sea levels [41]. One reason why this is problematic is that many of the worlds most populated cities are situated on the coast and are therefore more susceptible to flooding. The emissions of carbon dioxide is evidently a serious problem. It is therefore vital to develop accurate forecasting models to estimate CO$_2$ levels in both the short term and long term to enable better planning and control CO$_2$ levels.

In the context of power generation, a practical example of where the emission of greenhouse gasses is considered in the power generation process is the Dynamic Economic Emission Dispatch problem [241]. This is a multi-objective problem where a range of potential power generator configurations must be produced that minimize both cost and emissions to varying degrees. This range of solutions would then aid the engineer responsible for unit commitment and scheduling. This problem acknowledges that the environmental cost must also be considered in addition to the financial cost of power generation.

In terms of forecasting carbon dioxide levels, in 2015 Ganesan et al. implemented a neural network to forecast the exhaust emissions from a diesel electric generator [132]. A long term forecast of CO$_2$ levels from 1950 to 2050 was conducted by Schmalensee et al. that predicts how high global CO$_2$ levels will reach midway through the century [322]. Other studies have focused on the prediction of CO$_2$ levels at a national level in Brazil, Russia, India, China, and South Africa [385]. There are also many examples in the literature where other atmospheric pollutants are predicted. Bai et al. utilized neural networks to predict the daily air pollutant concentrations in China [23]. Krzywanski et al. predict the level of sulfur dioxide from boilers using a neural network [222]. Biancofiore et al. also utilized neural networks to predict hourly ozone
5.3 Neural Networks

Neural networks were first proposed in the 1960s. Juergen Schmidhuber provides a comprehensive overview of the history of neural networks [323]. Neural networks are computational models that are inspired by the network of neurons that biological brain is comprised of. The function of these networks is to read in an input signal and produce an output signal that corresponds to that input. This functionality is useful for a wide range of problems including: robotics, control, classification, regression and time series forecasting. Neural networks are comprised of layers of processing units, referred to as neurons. The first layer, referred to as the input layer, reads in a signal to the network. Next are a number of hidden layers of neurons that pass the signal from the input layer through the network through weighted connections until the signal is outputted through the output layer. Each hidden neuron receives input in the form of weighted signals from the previous layer. All neurons possesses an activation function, the most common activation function being the sigmoid or logistic function (Equation 5.1).

\[ a_j = \frac{1}{1 + \exp(-v_j)} \]  
(5.1)

This activation function is used to calculate the neurons output signal \( a_j \). The parameter \( v_j \) refers to the neurons input signal. This is calculated as shown in Equation 5.2.

\[ v_j = \sum_{i=1}^{N} w_{i,j} a_i \]  
(5.2)

Where \( v_j \) is the input to a neuron in the \( j^{th} \) layer, layer \( i \) is the preceding layer to layer \( j \) which contains \( N \) neurons, each neuron in the previous layer \( i \) has output \( a_i \) and each of these output signals are weighted by the value \( w_{i,j} \) as they are passed to each neuron in the current layer \( j \).

In terms of applying neural networks to forecasting, the input signal corresponds to

counters [45]. The problem of predicting the levels of harmful atmospheric pollutants is an important task due to the adverse affects that these pollutants have on both the environment and the overall health of the population.
Chapter 5. Forecasting in Ireland’s Energy Sector

the current and historic values in the time series. The network’s output corresponds to future predicted values. Parameter tuning revealed that for the Irish energy time series data used in this research, 2 inputs provided the best performance, i.e. the current data point at time $t$ and the previous data point at $t - 1$. The output corresponds to the prediction for time $t + 1$.

This research implements a particular form of neural network known as a Recurrent Neural Network (RNN) (Figure 5.1). In the standard feed forward neural network, each neuron receives input from the previous layer only. In recurrent networks however, the neurons in the hidden layer have connections joining neurons in the same layer. These are referred to as recurrent connections and give the system memory which is useful for problems with a temporal component such as time series forecasting.

![Recurrent Neural Network](image.png)

Figure 5.1: Recurrent Neural Network [240]. This figure depicts a fully connected recurrent neural network. Neurons are connected by weighted connections (known as synapses). These connections pass signals between neurons. The recurrent connections can be seen in the hidden layer of neurons. These recurrent connections give the network memory from the previous forward pass.

A forward pass refers to when a signal is passed through the neural network and an output is produced. One of the main issues surrounding neural network research is how to set the weights so that a forward pass produces the correct output. The next section will discuss the backpropagation algorithm which is the most common method of training neural network weights.

There are many examples of neural networks being applied to predicting national
power demands. In Iran, a multi-layered neural network was implemented to forecast monthly electrical energy consumption [18]. A neural network was applied to the task of forecasting long term energy forecasting in Greece [115]. The yearly energy demand was predicted in South Korea over a 27 year period using a neural network [136]. Another long term forecast study was conducted in Japan, where a neural network was trained on 20 years of data and then applied to forecast the subsequent 10 years [206]. In the USA, a neural network was utilized to forecast the energy consumption of the residential sector [208]. A hybrid neural network - linear model was implemented to forecast energy consumption in Taiwan [294]. There are also many examples of neural network forecasting in the literature with a focus on the environment. In 2018, Ye et al. explored the use of neural networks to predict CO$_2$ emissions from office buildings [397]. In 2017, Stamenkovic et al. utilized neural networks to predict nitrogen oxide emissions at a national level for 22 countries [344]. Another recent 2018 study has explored atmospheric dispersion prediction and estimated the source of hazardous gas using neural networks [302].

Many of the application of neural network to energy forecasting have been in the context of wind generation. In 2015, Osorio et al. implemented a neural network to forecast energy production of a wind farm in Portugal [287]. In 2016, Men et al. utilized ensemble neural networks to predict wind power and wind speed [257]. More recently, in 2017, Chang et al. proposed a radial basis function neural network-based model with an error feedback scheme to predict wind speed and power of a wind farm [66]. Each of these studies demonstrate the effectiveness of neural networks to predict the various factors associate with energy generation. The research presented in this paper builds upon each of these previous studies by applying neural networks to predict Irish power demand, wind energy generation and CO$_2$ levels.

In the context of research into Irish energy forecasting, there are no examples in the literature that utilize neural networks for time series forecasting. In 2004, Moehrlen explored the levels of uncertainty in wind energy forecasting using wind data from Ireland and Holland [268]. In 2005 Lang et al. implemented a Multi-Scheme Ensemble Prediction System to predict wind power generated from individual wind farms [224].
5.3.1 Backpropagation

This section will outline the Backpropagation algorithm which is the most common method of training neural networks. After a forward pass, the error $E$ is calculated based on the difference between the target output $t_k$ and the observed output $a_k$ of unit $k$. The error $E$ is calculated using Equation 5.3.

$$E = \frac{1}{2} \sum_k (t_k - a_k)^2 \quad (5.3)$$

By taking the partial derivatives of the error w.r.t. each output and the partial derivatives of each output w.r.t. the network, the error delta $(\delta)$ for the final layer which is to be propagated can be calculated using Equation 5.4.

$$\delta_k = (t_k - a_k)a_k(1 - a_k) \quad (5.4)$$

The weights between the final hidden layer and the output layer can then be updated using Equation 5.5.

$$w_{j,k} = w_{j,k} + \alpha \delta_k a_j \quad (5.5)$$

Where $\alpha$ is the learning rate. In order to propagate the error back through the hidden layers, the $\delta$ value for each hidden layer must be calculated. This can be done using the chain rule as the $\delta$ value for each neuron in layer $j$ depends on the $\delta$ values of the neurons in layer $j + 1$. The $\delta$ for each neuron in a hidden layer is calculated using Equation 5.6.

$$\delta_j = \sum_{j+1} \delta_{j+1}w_{j,j+1}a_j(1 - a_j) \quad (5.6)$$

The weights between the hidden layer $j$ and layer $j - 1$ can then be updated using Equation 5.7.

$$w_{j-1,j} = w_{j-1,j} + \alpha \delta_j a_{j-1} \quad (5.7)$$

Backpropagation is only suitable for problems that have target outputs, which is
one of the limitations of the algorithm. It is however perfectly suited to time series forecasting and will be implemented as a benchmark to compare evolutionary neural networks to.

5.4 Swarm and Evolutionary Methods for Energy Forecasting

The proposal of Genetic algorithms (GA) in the 1970s by John Holland was one of the first evolutionary algorithms proposed [169]. In the years since there have been a diverse range of proposed evolutionary algorithms. Some of the more popular and successful of these methods would include: Differential Evolution (DE), Particle Swarm Optimisation (PSO) and Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES). The advantage of evolutionary strategies such as those listed above is that they are applicable to a wide range of optimisation problems including neural network training. These algorithms make no assumptions about the search space of the problem and are therefore applicable to problems with a range of challenging properties such as: noisy, multi-modal, non separable and very large problems. Traditional mathematical optimisation methods, such as gradient descent, struggle with these problems types. As a result, evolutionary methods are a popular choice of optimisation algorithm. Based on the extensive literature review conducted, there are no examples in the literature of CMA-ES being applied to the problem of forecasting in the energy sector. This is the primary contribution of this research. It is also apparent that the existing studies do not give a comprehensive experimental review of the different methods of evolving neural networks in the energy sector. The distinct contribution that this research makes is to compare and contrast the leading evolutionary algorithms on multiple energy forecasting problems. This research also makes the contribution of applying evolutionary neural networks to forecasting Ireland’s energy needs.

5.4.1 Covariance Matrix Adaptation Evolutionary Strategy

One of the most studied optimisation algorithms in recent years is the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES). CMA-ES was first invented in 1996 by Hansen and Ostermeier [156]. The algorithm was proposed as a method to
solve non-convex, non-linear and complex global optimisation problems. CMA-ES is classed as an evolutionary algorithm. These are algorithms that incorporate some of the various operators used in the evolutionary process that occurs in nature, e.g. crossover, mutation and selection. The CMA-ES algorithm begins by randomly sampling a number of solutions to the optimisation problem. For the task of optimising a neural network for forecasting problems, these solutions correspond to sets of network weights. These solutions are then ranked in accordance to their fitness. The mean solution (m), covariance matrix (C) and step size (σ) are then updated by increasing the chances of sampling solutions with better fitness scores. The pseudocode in Algorithm 8 outlines how this process is carried out. For a problem where the number of dimensions = n, the covariance matrix is an $n \times n$ matrix that determines the variance around the mean. For neural network optimisation, $n =$ the number of weights in the network to be optimised. In Algorithm 8, $p_\sigma$ is the path for $\sigma$, $p_c$ is the path for $C$, $\omega$ are the recombination weights such that $\omega_1 \geq \omega_2 \geq ... \geq \omega_\mu > 0$ sum to 1, $y_\omega$ is the move of the population mean, $\lambda = 10$ is the population size, $\mu$ is the number of samples selected for the update, I is the identity matrix and $N(m, \sigma^2 C)$ is the multivariate normal distribution. Parameter tuning found that $\sigma_0 = 0.1$ and $\mu = 0.5\lambda = 5$ gave the best performance. Standard values for other constants were implemented: $c_c = 4/n$, $c_\sigma = 4/n$, $c_1 = 2/n^2$, $c_\mu = \mu\omega/n^2$, $d \approx 1$ and $\alpha = 1.5$

Initialize: Sample $\lambda$ random solutions  
Initialize: $C_0 = I$, $p_c = 0$ & $p_\lambda = 0$

while Evaluation $e < E_{max}$ do  
    for Solution $i = 1$ to $\lambda$ do  
        $x_i = m + \sigma y_i \sim N(m, \sigma^2 C)$  
        $f_i = \text{fitness}(x_i)$
    end
    Sort solutions $x$ according to fitness
    $m \leftarrow m + \sigma \sum_{i=1}^{\lambda} \omega_{p(i)} y_i := m + \sigma y_\omega$
    $p_\sigma \leftarrow (1 - c_\sigma)p_\sigma + \sqrt{1 - (1 - c_\sigma)^2} \sqrt{\mu\omega} C^{-0.5} y_\omega$
    $\sigma \leftarrow \sigma \times \exp\left(\frac{1}{\sqrt{2\pi\sqrt{\sigma^2}}} - 1\right)$
    $p_c \leftarrow (1 - c_c)p_c \frac{\|p_\sigma\|}{\sqrt{\|N(0,1)\|} - 1}\|p_\sigma\|^2 \frac{\sqrt{1 - (1 - c_\sigma)^2}}{\sqrt{\mu\omega} y_\omega}$
    $C \leftarrow (1 - c_1 - c_\mu)C + c_\mu \sum_{i=1}^{\lambda} \omega_{p(i)} y_i y_i^T + c_1 p_c p_c^T$
end
Return best solution

**Algorithm 8:** Covariance Matrix Adaptation - Evolutionary Strategy Algorithm

CMA-ES has had a great deal of success in its application to many real world problems since its first proposal. CMA-ES has been applied to optimising building placement for solar energy [198]. A more high dimensional task that CMA-ES has proved to be successful at is the problem of laser pulse shaping [335]. The problem of antenna
design has also benefited from the application of CMA-ES [191]. It has also been shown that a neural network trained with CMA-ES is an effective approach for reinforcement learning problems [176]. In 2013, Corne et al. hint at using CMA-ES for short term wind forecasting as future work [83]. The research presented in this paper pursues this suggested avenue of research. Based on the extensive literature review conducted, there are no examples of CMA-ES being applied to neural networks to forecast wind generation, energy demand or CO$_2$ levels.

5.4.2 Particle Swarm Optimisation

The next algorithm that will be applied to train the neural network is Particle Swarm Optimisation (PSO). The PSO algorithm dates back to 1995 when it was first proposed by James Kennedy [205]. The algorithm consists of a number $N = 50$ of particles. Each particle has a position $\vec{x}_t$ and velocity $\vec{v}_t$. A particle’s position corresponds to its position in the search space, i.e. a candidate solution to the optimisation problem. For the task of neural network training, a position corresponds to a set of network weights. The velocity of the particle describes how the particle will change its position between iterations of the algorithm. For minimization problems, positions with a lower fitness are considered to be better. The fitness of a particle’s position is determined by an objective function. For the problem of training a neural network for forecasting, this objective function corresponds to the Mean Squared Error. At each iteration, each particle will update its velocity based on its personal best position $\vec{p}_b$ and the best overall position $\vec{g}_b$. This process is carried out until a stopping criteria is met such as a threshold fitness being achieved or a predetermined number of evaluations have been carried out. Equation 5.8 describes how the particles update their position and velocity.

\begin{align}
\vec{v}_{t+1} &= \chi (\vec{v}_t + r_1 c_1 (\vec{p}_b - \vec{x}_t) + r_2 c_2 (\vec{g}_b - \vec{x}_t)), \\
\vec{x}_{t+1} &= \vec{x}_t + \vec{v}_t
\end{align}

(5.8a)

(5.8b)

Where $c_1$ and $c_2 = 2.05$ are acceleration coefficients, $r_1$ and $r_2$ are random numbers between 0 and 1 and finally $\chi \approx 0.72984$ is the constriction factor. The constriction factor guarantees that as the particles move around the problem space evaluating can-
Chapter 5. Forecasting in Ireland’s Energy Sector

didate solutions, they will eventually converge. The operation of the PSO algorithm is described in Algorithm 9.

\begin{algorithm}
\begin{algorithmic}
\State Initialize N particles with random $\vec{v}_0$ and $\vec{x}_0$
\While {Evaluation $e < E_{\text{max}}$}
\For {Particle = 1 to N}
\State Update personal best position $p_{b_t}$
\State Update neighbourhood best position $g_{b_t}$
\State Evaluate particle’s current position $x_t$
\State $v_{t+1} = \chi(\vec{v}_t + r_1c_1(p_{b_t} - \vec{x}_t) + r_2c_2(g_{b_t} - \vec{x}_t))$
\State $x_{t+1} = \vec{x}_t + \vec{v}_t$
\EndFor
\EndWhile
\State Return best solution
\end{algorithmic}
\end{algorithm}

Algorithm 9: Particle Swarm Optimisation Algorithm

PSO has been successfully applied to many real world problem domains. A PSO trained neural network has shown to be a successful strategy for the task of watershed management [245]. It has also been shown that PSO is very effective at the task of power generator scheduling [241]. In 2009, Welch et al. implemented PSO to train neural networks for wind speed forecasting with great success [376]. In 2011, Catalao et al. applied PSO in conjunction with fuzzy logic and a wavelet transform to predict wind power generation in Portugal [64]. PSO has also been applied to predicting power demand in Iran in 2014 by Bahrami et al. [22]. This study did not however implement neural networks as the predictive model. Similarly in 2014, Selakov et al. combined PSO with Support Vector Machines (SVM) to predict power demand for the city of Burbank in the USA [328]. In 2009, PSO was implemented with neural networks for short term load forecasting for an area in New York [28]. The authors concluded that PSO was in fact an effective neural network training algorithm. There has been much less research in applying PSO to forecasting CO$_2$ levels. In 2016, Ozceylan applied PSO to CO$_2$ emissions forecasting in Turkey [290]. This study did not investigate combining PSO with neural networks, which is element of this research. The research presented in this paper builds on these previous studies by applying PSO to neural networks and applying the resulting forecasting algorithm to wind, energy and CO$_2$ forecasting in Ireland. This research will also compare PSO to other popular evolutionary algorithms that were not evaluated in many of these previous studies.
5.4.3 Differential Evolution

The final evolutionary algorithm that will be implemented in this study is Differential Evolution (DE). Like CMA-ES, DE uses evolutionary operators such as selection, crossover and mutation to optimise complex global optimisation problems. The algorithm was first proposed in 1997 by Storn and Price [348]. Like all evolutionary algorithms, DE does not rely on problem gradients or need any prior information about the problem at hand. This makes it a highly robust and widely applicable optimisation algorithm that can be applied to problems that traditional mathematical optimisation algorithms would either take too much computational time to solve or would converge onto a local optimum. DE is also a more straightforward optimisation algorithm to implement compared to CMA-ES. The ease of implementation and effectiveness of DE makes it a popular optimisation algorithm and suitable for the task of optimising network weights for forecasting problems.

The DE algorithm begins by creating a number of agents with random position in the problem space. Like the particles in PSO, the agents in DE each have a position $\vec{x}$ that represents a potential solution. The solution here represents a set of network weights. At each iteration of the algorithm, the current agent’s position is combined with the positions of three other distinct agents’ positions to produce a new position $\vec{y}_i$. This is outlined in the pseudo-code in Algorithm 10. If the new position $\vec{y}_i$ of the agent is better than its previous position $\vec{x}_i$, the agent moves to the new position. The quality of the position is judged based on its fitness which is calculated using the objective function. In this research, the fitness function is the MSE of the neural network when evaluated on the training data. This is repeated for each of the agents until a predetermined number of problem evaluations has been conducted or until some fitness threshold has been reached. In Algorithm 10 the parameter $CR = 0.7$ is the crossover probability, $F = 0.5$ is the differential weight and $N$ is the number of agents ($N =$the number of network weights).

Within the context of energy forecasting, Meeyappan et al. applied DE to neural networks for wind power prediction for economic dispatch in 2015 [260]. In 2016, Yang et al. applied DE to neural networks to predict the short term demand in New South Wales, Australia [393]. These previous studies are built upon in this paper by applying DE to forecast power demand, wind generation and CO$_2$ for Ireland. Based on the extensive literature review conducted, there are no examples in the literature of DE being applied to predict CO$_2$ levels.
Initialize N agents with random positions $\vec{x}_0$

\[\text{while } \text{Evaluation } e < E_{\text{max}} \text{ do} \]

\[\text{for Agent} = 1 \text{ to } N \text{ do} \]

Select 3 other agents A, B and C

Select random dimension index R

\[\text{for dimension } i = 1 \text{ to } D \text{ do} \]

generate random number $r \in [0,1]$

\[\text{if } r < CR \text{ Or } i = R \text{ then} \]

new position $y_i = a_i + F \times (b_i - c_i)$

else

$y_i = x_i$

end if

end if

if fitness$(y) < \text{fitness}(x)$ then

replace $x$ with $y$

end if

end for

end if

end

Return best solution

\textbf{Algorithm 10:} Differential Evolution Algorithm

5.5 Experimental Methods

This section will outline the various experiments conducted and the implementation details of each algorithm. The experiments described in this research were implemented in Java.

5.5.1 Forecast Data

Each forecasting method will be evaluated on data sets relating to Ireland’s: 1) Energy demand. 2) Wind power generation. 3) CO$_2$ intensity levels. In each of these three forecasting problems, each forecasting algorithm will be trained on a month of data from the 16$^{th}$ of August 2017 until the 14$^{th}$ of September 2017. The performance of each model will then be tested on completely separate month of time series data sets from the 18$^{th}$ of September 2017 until the 17$^{th}$ of October 2017. Each of these data sets consist of time series data in increments of 15 minutes, with a total of 2878 data points. This data was sourced from Eirgrid [114].

5.5.2 Network Parameter Selection

A recurrent neural network was implemented in this research for time series prediction. Recurrent neural networks were selected over feed forward networks as the recurrent connections give the network a memory of previous forward passes. This is beneficial for time series problems as there is a strong temporal element in time series data.
5.5. Experimental Methods

sets, i.e. a correlation between the current state time step and previous time step. Each network is fully connected and has six hidden neurons. The networks have 2 inputs corresponding to the current (time t) and previous (time t-1) time step values. This was implemented for all data sets. Parameter sweeps revealed that 6 was the optimum number of hidden neurons. More than 6 hidden neurons did not lead to any increase in performance due to the increased number of weights that must be trained as the network increased in size. Parameter sweeps also established that more than two network inputs did not lead to any increase in performance. The networks have one output corresponding to the network’s prediction at time step t+1.

5.5.3 Network Training

Each network with 6 neurons has a corresponding set of 54 weights. All neural network training algorithms optimised the set of weights for 10E6 evaluations of the training data. The fitness of the network was determined by the Mean Squared Error (MSE) of networks prediction accuracy on the training data. After a network is trained for 10E6 evaluations on the training data, it is then evaluated on the test data. This process is repeated over 10 runs to ensure statistically significant results.

5.5.4 Comparative Forecasting Methods

In total 7 algorithms will be evaluated on each of the 3 forecasting problems. These are:

1. Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES).
2. Particle Swarm Optimisation (PSO).
5. Moving Average (MA).
6. Random walk forecasting (RWF).
7. Linear Regression (LR).

Implementing this diverse set of 7 algorithms gives a comprehensive comparison of forecasting algorithms for energy time series problems. Forecasting methods 1-4 all
Chapter 5. Forecasting in Ireland’s Energy Sector

involve training a neural network to output the predicted values. The implementation
details of each of these algorithms has been outlined in detail in Sections 5.3 and 5.4.

The final three algorithms: Moving Average, Random Forest Walk and Linear Regression will be implemented to give further context to the neural network based forecasting methods. The moving average method is a commonly used forecasting approach. This method consists of predicting a future value as an average of \( n \) previous values. The forecast value at time \( t+1 \), \( V_{t+1} \) is calculated using equation 5.9.

\[
V_{t+1} = \frac{\sum_{i=0}^{n-1} V_{t-n}}{n}
\]  

(5.9)

Random walk forecasting is the next forecasting method that will be implemented.
This approach simply consists of predicting the next future value \( V_{t+1} \) as equal to the current observed value \( V_t \).

Linear regression (LR) is a well known machine learning algorithm that is routinely used for long term electricity consumption forecasting [203]. Linear regression consists of constructing a linear model whereby the parameters of the model are estimated using the data [269]. Equation 5.10 describes the linear model that LR implements to forecast the future value \( V_{t+1} \) at time \( t + 1 \). The parameters \( C \) are constants that are adjusted by the LR algorithm so that the model produces the lowest forecasting error.

\[
V_{t+1} = C_1 V_{t-1} + C_2 V_t + C_3
\]  

(5.10)

5.5.5 Forecast Accuracy Evaluation

Four standard metrics will be used to evaluate the performance of each forecasting algorithm. The first is the Mean Absolute Error (MAE). This is a easy to understand metric which has the added benefit of having the same units as the value being predicted. The MAE is calculated by Equation 5.11.

\[
MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}
\]  

(5.11)

The next metric is the Mean Squared Error (MSE). This metric does not have the
same units as the value being predicted. The MSE is more sensitive to outliers in the
data and is calculated using Equation 5.12.

\[
MSE = \frac{\sum_{i=1}^{n} |y_i - x_i|^2}{n} \tag{5.12}
\]

The Root Mean Squared Error (RMSE) is the next metric that will be evaluated. Like
the MSE, it is more sensitive to outliers than the MAE, however it is more readable
than the MSE. The RMSE is calculated using Equation 5.13.

\[
RMSE = \sqrt{\frac{\sum_{i=1}^{n} |y_i - x_i|^2}{n}} \tag{5.13}
\]

The final metric that will be evaluated is the Mean Absolute Percentage Error (MAPE).
This metric gives the forecasting error as a percentage of the actual value being es-
timated, this means that the MAPE can be used to compare forecasting approaches
across problems with different units. The MAPE is calculated using Equation 5.14.

\[
MAPE = \frac{100}{n} \sum_{i=1}^{n} \frac{|y_i - x_i|}{|y_i|} \tag{5.14}
\]

Where \( n \) is the number of forecast values, \( y_i \) is the actual value and \( x_i \) is the forecast
value.

5.5.6 Experiments Conducted

There will be three experiments conducted in this research paper. The first experiment
will involve evaluating the performance of each forecasting algorithm on each set of
training data. All algorithms will be judged based on their rate of convergence,
prediction accuracy, performance consistency and ease of implementation.

The second experiment will evaluate the performance of each trained network on a
month of previously unseen data for each of the three forecasting problems. The
purpose of this is to establish the generality of each forecasting method and ensure
they are capable of providing good performance outside of the training data.

The third and final experiment will evaluate how far into the future the evolved
network can predict. In terms of planning, it would be advantageous to have accurate
predictions farther into the future. This would aid the scheduling and operation of the power generation process.

5.6 Results

This section presents the results of each of the experiments outlined above followed by a discussion in order to gain insight and highlight their significance. The two tailed $t$ test with a significance level of $\alpha = 0.05$ was conducted to determine significant performance differences when comparing algorithms. All values were rounded to 4 significant figures.

5.6.1 Convergence

The convergence of CMA-ES, PSO and DE can be seen in Figures 5.2(a), (b) and (c) for the wind power generation, power demand and CO$_2$ emissions respectively. In each of these graphs, it can be seen that CMA-ES and PSO converge the fastest, however PSO stops improving after approximately $10^5$ evaluations for each problem. CMA-ES continuously improves for the entire training period. These improvements are much more incremental after approximately $10^5$ evaluations. DE is the slowest converging algorithm of the evolutionary approaches. DE offers much more gradual improvements in the accuracy of the network and eventually converges to a better solution than PSO on the power demand prediction problem (Figure 5.2 (b)). Of these three evolutionary algorithms, CMA-ES and PSO provide the fastest convergence. Overall CMA-ES provides the best performance in terms of rate of convergence and final network accuracy on the training data.

In terms of complexity, DE and PSO are more straightforward to implement than CMA-ES. Although these algorithms are more simple in their design, Figure 5.2 illustrates that the added complexity of CMA-ES does result in a more accurate neural network for forecasting. It can be seen that PSO does provide rapid convergence. Therefore if limited time or computational resources are available, PSO could be a viable alternative. Although DE is slower to converge, it is apparent from the convergence graphs that it has not yet converged on an optimum solution. Given more computational time, DE could perhaps converge on a network with an accuracy comparable with CMA-ES.
5.6. Results

5.6.2 Accuracy

The accuracy of each evolutionary method and other benchmark algorithms for both training and test data is presented in Table 5.1 for the wind power generation, power demand and CO$_2$ emissions. In this table, the average and standard deviation of the MAE, MSE, RMSE and MAPE are displayed for both the training and test sets. It is evident from this table that a network trained with CMA-ES performs significantly better than other methods on the training data. CMA-ES provides the highest prediction accuracy on all three sets of training data. On the test data sets, CMA-ES performs best on two of the three forecasting problems (wind power generation and power demand). CMA-ES performs third best on the CO$_2$ intensity level data set, where it is outperformed by LR and RW. Of the neural network based approaches, CMA-ES performs best on all data sets, for training and testing. In terms of performance consistency, BP had the smallest standard deviation followed by CMA-ES, PSO then DE. This is the same across all forecasting problems. Of the non neural network based approaches, LR had the highest forecasting accuracy on the wind power generation and CO$_2$ intensity prediction problems, followed by RW.
then MA. On the power demand prediction problem, LR performed the worst. The reasons for this will be discussed in detail in Section 5.6.5.

One observation from Table 5.1 worth noting is the difference between the MAPE and MAE, MSE and RMSE on the wind generation prediction problem. CMA-ES performs significantly better than all other approaches when compared using the MAE, MSE or RMSE. CMA-ES performs slightly worse than LR however when compared using the MAPE. This points to a common criticism of the MAPE metric. The MAPE metric gives higher significance to errors when the target value is near 0 than values that are not. For example a forecast error of 10 MW would correspond to a percentage error of 0.5 % if the value being forecast \( y_i = 2000 \text{MW} \). However a forecast error of 10 MW would correspond to a percentage error of 100 % if the value being forecast \( y_i = 10 \text{MW} \). The wind power prediction problem contains values that are near zero. This means that MAPE gives a higher significance to these lower values than others. Since CMA-ES has a higher MAPE than LR on this problem, it indicates that LR is slightly better at predicting these lower values. However if it is assumed that all time series values have equal significance, which it is believed to be the case, then the MAE, MSE and RMSE are better indicators of forecast accuracy. These three metrics reveal that CMA-ES has a higher accuracy than LR. MAPE does indicate that the accuracy of CMA-ES might slightly decrease when predicting lower values.

When inspecting the differences between the accuracy of each forecast model on the training and test data, it is evident that each model generalizes well to unseen data. As Table 5.1 shows, the accuracy of each model is only slightly worse on the test data than on the training data for the wind and power demand forecast problems. The accuracy of most models actually improves on the CO\(_2\) forecasting test data. A possible reason for this is that lower CO\(_2\) values are observed overall on the month of test data than the month of training data. It appears to be the case that the the forecast models are better at forecasting these lower values for the CO\(_2\) data set. The PSO and DE trained networks seem to be an exception to this however as they have higher MSE for the CO\(_2\) test data than for the training data.

Figure 5.3 illustrates the spread of each forecasting algorithm. The most salient observation from these graphs is that each of the evolutionary neural networks has much higher variance in its performance. CMA-ES has the smallest spread, followed by PSO then DE. This is due to the stochastic nature of these algorithms. The performance of DE on the CO\(_2\) forecasting problem is one of the more noticeable
features in Figure 5.3 (c). DE has a large amount of variation in its accuracy on the test data for each run. The worst run is significantly worse than all other approaches. Its best MAE however is comparable with CMA-ES, LR and RW which have the highest accuracy. CMA-ES is also very inconsistent with its performance on the CO\textsubscript{2} test data. This is also the only problem where CMA-ES does not have the best average MAE, MSE or RMSE. The reason for this could be the previously mentioned lower test CO\textsubscript{2} values. This will be discussed further in Section 5.6.5.

Table 5.1: Forecast Accuracy

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Wind Power Generated</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>CO\textsubscript{2} Intensity Level</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training MAE (StDev)</td>
<td>MAE (StDev)</td>
<td>MSE (StDev)</td>
<td>RMSE (StDev)</td>
<td>MAPE (StDev)</td>
<td>Testing MAE (StDev)</td>
<td>MSE (StDev)</td>
<td>RMSE (StDev)</td>
</tr>
<tr>
<td>CMA-ES</td>
<td>8.879 (0.052)</td>
<td>133.6 (1.472)</td>
<td>11.56 (0.064)</td>
<td>2.056 (0.012)</td>
<td>8.290 (0.393)</td>
<td>120.5 (17.33)</td>
<td>10.98 (0.743)</td>
<td>2.258 (0.152)</td>
</tr>
<tr>
<td>PSO</td>
<td>9.216 (0.101)</td>
<td>142.1 (3.653)</td>
<td>11.92 (0.152)</td>
<td>2.130 (0.018)</td>
<td>9.035 (0.157)</td>
<td>143.2 (6.098)</td>
<td>11.96 (0.251)</td>
<td>2.558 (0.058)</td>
</tr>
<tr>
<td>DE</td>
<td>9.146 (0.265)</td>
<td>150.0 (7.605)</td>
<td>12.44 (0.310)</td>
<td>2.231 (0.064)</td>
<td>9.669 (0.872)</td>
<td>165.5 (36.17)</td>
<td>12.79 (1.360)</td>
<td>2.747 (0.339)</td>
</tr>
<tr>
<td>BP</td>
<td>9.178 (0.000)</td>
<td>140.4 (0.024)</td>
<td>11.85 (0.001)</td>
<td>2.124 (0.005)</td>
<td>8.569 (0.021)</td>
<td>125.1 (0.014)</td>
<td>11.18 (0.021)</td>
<td>2.386 (0.008)</td>
</tr>
<tr>
<td>MA</td>
<td>9.290 (0.000)</td>
<td>145.1 (0.000)</td>
<td>12.04 (0.000)</td>
<td>2.145 (0.000)</td>
<td>8.316 (0.000)</td>
<td>120.1 (0.000)</td>
<td>10.96 (0.000)</td>
<td>2.236 (0.000)</td>
</tr>
<tr>
<td>RW</td>
<td>9.384 (0.000)</td>
<td>148.5 (0.000)</td>
<td>12.20 (0.000)</td>
<td>2.182 (0.000)</td>
<td>8.911 (0.000)</td>
<td>111.3 (0.000)</td>
<td>10.55 (0.000)</td>
<td>2.168 (0.000)</td>
</tr>
<tr>
<td>LR</td>
<td>8.993 (0.000)</td>
<td>137.0 (0.000)</td>
<td>11.70 (0.000)</td>
<td>2.082 (0.000)</td>
<td>7.924 (0.000)</td>
<td>100.1 (0.000)</td>
<td>10.45 (0.000)</td>
<td>2.149 (0.000)</td>
</tr>
</tbody>
</table>

5.6.3 Predictions

The forecast predictions of each evolutionary algorithm are plotted along with the actual values for the training and test sets in Figures 5.4, 5.5 and 5.6 of the wind, power demand and CO\textsubscript{2} data sets respectively. As these graphs illustrate, each forecasting algorithm is capable of forecasting future values with a high degree of accuracy. Differences between the forecast values and the actual values can only be observed at the peaks of the time series data. This highlights that the error of the forecasting methods is largest when there is a rapid change in the time series data. This can be observed in all of the forecasting problems in Figures 5.4 to 5.6.

There is also an interesting distinction between the forecast values in the training set.
Figure 5.3: Forecast Accuracy Spread of Each Algorithm. This figure illustrates the spread in accuracy of each forecasting algorithm. Graphs a, b and c depict the wind power, power demand and CO₂ levels respectively.

and the test set. As Figure 5.6 illustrates, there is a much more noticeable difference between the forecast values and the actual values in the test data (Graph (b)) than in the training data (Graph (a)) for the PSO and DE trained networks. This is particularly evident for the lower CO₂ intensity values. As previously mentioned, DE and PSO had the highest error on this data set, which highlight that they do not generalize to unseen CO₂ data as well as the other models. The suspected reason for this is that there are significantly lower CO₂ values in the test set than in the training set. This seems to favor other forecast models including the CMA-ES and BP trained networks, however the PSO and DE trained networks produce less accurate forecasts for lower CO₂ levels. As could be seen in the convergence graphs in Figure 5.2, PSO and DE converge to produce a network with a much lower accuracy than CMA-ES. It is thought that these higher CO₂ forecast errors in the test set is a manifestation of this.

The generalization of each algorithm is discussed in more detail in Section 5.6.5. When inspecting the differences between the forecast and actual values in the CO₂ test data (Figure 5.6 (b)), it is observed that DE has the largest difference from the
5.6. Results

actual CO$_2$ values, followed by PSO then CMA-ES. This reflects the overall accuracy of each algorithm in Table 5.1. Unlike PSO and DE, CMA-ES has a higher overall accuracy in the test set than the training set for the CO$_2$ prediction problem. This can be seen in Figure 5.6, where the predictions of CMA-ES is much closer to the actual values than PSO and DE.

The error at each time step for each forecasting problem can be seen in Figures 1, 2 and 3 in Appendix .3. These graphs highlight that the primary source of error for CMA-ES evolved neural network is when there are sudden extreme changes in the value being predicted. As expected the network has the highest accuracy when there is a shallow gradient in the time series data. The wind power forecasting error in Figure 1 (Appendix .3) clearly illustrates this. At approximately time step 1500 of the training data and 2000 of the test data, the error on the lower plot is very low. These points correspond to locations on the upper time series plots of the wind power where there is very little change in the power generated from wind. The error spikes in the lower plots correspond to locations in the time series data where there are sudden changes in wind power generation. There is a particularly large error at approximately time step 2700 in the training set. The wind power generated at this time changes from a steep decline to a steep incline. The stochastic nature of the power available from wind means that it is very difficult for the neural network forecasting model to anticipate and adapt to these sudden random changes. For this reason many forecast models include wind speed forecasts into the prediction of wind power generation. As previously stated however, this is outside of the scope of this research.

As Figure 2 (Appendix .3) illustrates, there is much less stochasticity in the power demand profile. It is periodic in nature unlike the wind power generation and CO$_2$ level profiles. The error in the power demand predictions is also therefore regular. Every day at approximately 6:00 am, when the power demand changes from decreasing to increasing, there is a spike in the prediction error. Despite the fact that this is a regular occurrence every day, the accuracy of the neural network prediction is at its lowest at this point. The CMA-ES trained neural network does however have a higher accuracy than all other approaches evaluated for power demand forecasting. One possible way to increase the accuracy of the neural network at forecasting the power demand would be to incorporate the time of day as an input to the network. This would give the network a sense of the periodic nature of the problem and potentially
improve performance.

Finally, Figure 3 (Appendix .3) graphs the error at each time step for the CO$_2$ intensity prediction problem. Similar to the wind power generation graph, there is no periodicity in the CO$_2$ data. Similar observations can be made from this graph as from the wind power graph, the prediction errors correspond to points in time when there is the a rapid change in CO$_2$ intensity. There is a particularly high forecast error at time step 627 of the test data. As expected, this is due to a very rapid change in the CO$_2$ levels. From time step 622 to 626 the CO$_2$ intensity level was steadily decreasing from approximately 485 to 472 g CO$_2$/kWh, which the neural network was able to predict with a reasonable degree of accuracy. Within the 15 minutes between time point 626 and 627, the CO$_2$ intensity increases from 472 to 546 g CO$_2$/kWh. However the network had predicted that the CO$_2$ levels would decrease at a similar steady rate that it had observed for the previous 4 time steps. This results in the relatively large prediction error observed in test predictions.

![Figure 5.4: Wind Power Generation Predictions for Training and Test Data.](image)

Figure 5.4: Wind Power Generation Predictions for Training and Test Data. This figure illustrates the predicted wind power generated at each time step using the evolved neural networks. Graph (a) illustrates the predictions for the training data while graph (b) presents the test data predictions.

![Figure 5.5: Power Demand Predictions for Training and Test Data.](image)

Figure 5.5: Power Demand Predictions for Training and Test Data. This figure illustrates the predicted power demand at each time step using the evolved neural networks. Graph (a) illustrates the predictions for the training data while graph (b) presents the test data predictions.
5.6. Results

Figure 5.6: CO$_2$ Intensity Predictions for Training and Test Data. This figure illustrates the predicted CO$_2$ level at each time step using the evolved neural networks. Graph (a) illustrates the predictions for the training data while graph (b) presents the test data predictions.

5.6.4 Multi-step Ahead Prediction

The aim of the experiment conducted in this section was to evaluate how the forecast accuracy of the neural network changes as it attempts to predict farther into the future. Figure 5.7 illustrates how the forecasting accuracy decreases as it predicts 1 time step into the future (15 minutes) versus 10 time steps (2.5 hours). As this graph demonstrates, there is a predominantly linear increase in forecasting error (MAE) as the network trained with CMA-ES predicts farther into the future. This is observed for all data sets. The CO$_2$ data set has different units to the wind and power demand data sets, which is why the increase appears to be much less than for the other two data sets. The prediction accuracy for the CO$_2$ data set increases from $MAE_{Train} = 8.879$ and $MAE_{Test} = 8.200$ for 1 step ahead prediction to $MAE_{Train} = 25.106$ and $MAE_{Test} = 31.964$ for 10 steps ahead prediction. The accuracy for the wind forecasting problem changes from $MAE_{Train} = 29.701$ and $MAE_{Test} = 32.819$ for 1 step ahead prediction to $MAE_{Train} = 149.176$ and $MAE_{Test} = 221.482$ for 10 steps ahead prediction. Finally the forecasting accuracy for predicting the power demand changes from $MAE_{Train} = 22.495$ and $MAE_{Test} = 24.466$ for 1 step ahead prediction to $MAE_{Train} = 170.584$ and $MAE_{Test} = 248.785$ for 10 steps ahead prediction. It is to be expected that the forecasting accuracy decreases as the algorithm attempts to predict farther into the future. Given that values that are to be predicted vary by 1000s of MWs for the power and wind forecasting problems, the 2.5 hour ahead forecasting accuracy achieved by the CMA-ES is still considered relatively accurate.

Another observation that can be made from Figure 5.7 is that difference between the forecasting accuracy on the training and test data becomes larger as the network predicts farther ahead. The forecasting accuracy difference is almost indistinguishable.
for 1 time step ahead but is significantly higher for the training data on all three problems when predicting 10 steps ahead. This indicates that the networks are able to more accurately forecast previously unseen data when predicting only 1 step ahead.

![Multi Step Prediction](image)

**Figure 5.7**: MAE of CMA-ES Neural Network When Forecasting Further Into the Future. This figure illustrates how the forecasting accuracy of the neural network degrades as it predicts further into the future for each problem.

### 5.6.5 Discussion

The results presented in the previous section demonstrate that evolving neural networks using CMA-ES is an effective approach to developing accurate forecasting models. In terms of accuracy, convergence and robustness, CMA-ES outperforms all other approaches evaluated for each of the three energy forecasting problems. CMA-ES provides the highest forecasting accuracy for the training data in all three forecasting problems. When evaluated using previously unseen test data, CMA-ES performs best on two of the three forecasting problems. When contrasting the performance differences between the training and test data, it is insightful to look at the training and testing data sets. CMA-ES had the third highest MAE on the test data for the CO₂ intensity forecasting problem. It is thought that the reason CMA-ES was outperformed in this instance by RW and LR is that the test data set contained CO₂ levels that were significantly lower than any values in the training set. Although every model aside from DE and PSO trained networks produced a lower forecast MSE on the CO₂ test data, it appears that LR is slightly more robust to unseen data that is significantly different from data experienced during training than the neural network based approaches.

Figure 5.6 illustrates the increased error when the PSO and DE trained neural net-
works predict these lower values in the test set. The training set didn’t contain any
values lower than 250 g CO$_2$/kWh, while the test set contained three separate in-
stances where the CO$_2$ level drops below 250 g CO$_2$/kWh. The PSO and DE trained
neural networks produced higher errors when predicting these values. This is a well
known problem in machine learning research however. The performance of any func-
tion approximators will suffer if it is given new values that are significantly different
from those previously experienced in training. A way to mitigate this issue would be
to train the networks on a larger data set containing every possible values that the
network could possibly face when implemented to forecast for new and unseen data.
Such a data set is currently unavailable to the authors and is therefore differed as a
topic for future research. CMA-ES did still perform third best on the test data for
the CO$_2$ data but it is worth highlighting this issue. LR appears to be more robust
to significantly different values for the CO$_2$ forecasting problem.

This issue was not observed in the test data for the wind power generation and
power demand prediction problems. As is evident from Figures 5.4 and 5.5, the
range of values observed in the training and testing data sets for these problems is
fairly consistent. One of the most prominent observations made from Table 5.1 is the
performance of the neural network based forecasting methods when compared to MA,
RW and LR. The MAE of each neural network forecasting method is between 20 and
45 MW for both the training and test data, while the MAE of MA, RW and LR is in
the range of 45 to 105 MW. It is thought that the reason for this large gap in forecast
accuracy of the neural network versus non neural network based approaches is due
to the nature of the power demand data. The power demand data is very periodic
in nature, as illustrated by Figure 5.5. The data set contains many steep gradients
where the power demand increases and decreases day to day. These rapid increases
and decreases in power demand mean that naïve approaches such as MA and RW are
unable to make accurate forecasts. These approaches make future predictions purely
based on the current value (and historic values for MA) and do not form any model of
the problem. These approaches can work well for problems that contain fewer steep
gradients such as predicting wind power generated and CO$_2$ levels, i.e. RW performs
third best on the wind and CO$_2$ prediction problems. Unlike MA and RW, LR does
form a model that is used for prediction. As Table 5.1 shows however, this linear
model is inadequate for accurately predicting the power demand data as LR performs
worst of all algorithms at forecasting future power demands. Neural networks perform
much better on this problem as they can adapt to the steep and frequent gradients in
the power demand data and can accurately represent complex non linearities in the data that other methods cannot.

The results presented in the previous section indicate that CMA-ES can train networks to predict multiple time steps into the future with a reasonable accuracy. Figure 5.7 shows that the accuracy of the networks diminishes as the networks predict further into future, as would be expected. These graphs show that the accuracy of the network predictions still remains relatively high even when predicting 2.5 hours. This is important when scheduling power generators. Power generators can only increase/decrease their power outputs by a limited amount from hour to hour [246]. This task is even more complex when wind power is incorporated to the power generation process due to its stochastic nature [165]. It is vital to know in advance what the power demand and wind power generation will be so that thermal power generators can be scheduled optimally. The accuracy of the CMA-ES neural network forecasting model evaluated in this research can therefore greatly benefit this process by predicting future power demand and wind power generation with a high degree of accuracy.

An interesting side note is that the final peak at the end of the wind power generation test data corresponds to the amount of wind power generated when Storm Ophelia hit Ireland on the 16th of October 2017. As Figure 5.4 illustrates, despite the high winds that were experienced in Ireland during this time period, there was a relatively low amount of power generated from wind energy when compared to earlier in the month. This is because wind turbines are not designed to operate in exceedingly strong winds and are shut down to avoid damage. This research utilized a univariate approach whereby predictions were made solely based on the wind power time series data and not meteorological data. Although it was outside of the scope of the research conducted in this paper, the use of meteorological data can enhance the accuracy of wind power generation forecasting models [7]. In the case of the power generated from wind during Storm Ophelia, incorporating wind speed data into the forecast model would have the adverse effect of reducing the accuracy of the model. This is because the high wind speeds would lead the model to believe that more power will be generated from wind energy when in reality there will be less wind energy available as the turbines are not operational. For the majority of the time however, i.e. when there are no extreme events such as storms, incorporating meteorological data such as wind speed will increase the accuracy of the forecast model.

Previous studies have demonstrated how swarm and evolutionary methods such as
PSO and DE can be of benefit to forecasting problems. In 2011, Catalao et al. demonstrated that a PSO trained neural network is an effective approach to short term wind power prediction [64]. In 2016, Yang et al. successfully implemented a DE trained neural network for short term power prediction [393]. Özceylan utilized a PSO trained network to predict CO\textsubscript{2} emissions in 2016 [290]. There is also a wealth of literature outlining successful applications of neural networks to forecasting power demand in many countries, e.g. Iran [18]. Neural networks have also successfully been applied to the task of wind power generation prediction [44]. In terms of air pollutants, neural networks have been successfully applied to the task of CO\textsubscript{2} forecasting [293]. There are also examples of forecasting other air pollutants such as nitrogen oxide [344]. The research presented in this paper has a positive impact on each of these previous studies as it shows that CMA-ES is an effective method for evolving neural networks for forecasting. CMA-ES could therefore be applied to neural networks for each of these forecasting problems and give high forecasting accuracy. Corne et al. even suggest using CMA-ES for short term wind forecasting as future work as they suspected that it would provide a high accuracy forecasts [83]. The research presented in this paper confirm that this is the case. The high forecasting accuracy of a CMA-ES can benefit previous studies that have sought to predict wind power generation from local wind farms in Ireland [224].

The neural networks implemented in this research do not consider any other data when making predictions. The predictions are made solely using historic time series data. Many forecast models incorporate other information such as time of day or estimated wind speed [305]. The results presented here would be directly transferable to instances where the network is trained using time series data and other data such as those mentioned. This research demonstrates that CMA-ES is a very effective approach for training neural networks for energy prediction. Studies that utilize data other than time series data to train a neural network for forecasting would also benefit from training the network using the CMA-ES algorithm. CMA-ES converges faster to a network with higher accuracy than PSO, DE and backpropagation. A larger network with more inputs for other data would therefore benefit further from the use of CMA-ES to train its weights as it is the most effective algorithm out of those evaluated here.
5.7 Conclusion

The primary aim of this research was to investigate if a neural network train with CMA-ES is capable of accurately predicting Ireland’s power demand, wind power generation and CO$_2$ levels. The results obtained indicate that CMA-ES can in fact produce accurate predictions for each of these problems. Moreover CMA-ES performs very competitively when compared to other state of the art approaches. CMA-ES performs significantly better than other evolutionary methods, i.e. PSO and DE, on all problems in terms of accuracy, convergence speed and consistency. The results show that the largest sources of forecast errors come from when there are large changes in the time series data. The neural networks also appear to perform worse when faced with data that it outside of the range that it has trained on, i.e. the CO$_2$ test data. On this problem, methods such as LR and RW are more robust to these new data points. When predicting power demands, the CMA-ES trained neural network performed significantly better than all other methods on both training and test data. CMA-ES was able to adapt best to the periodic nature of the time series data. When evaluated predicting multiple time steps into the future, the CMA-ES trained network was able to maintain a reasonably high level of accuracy even when predicting 10 time steps (2.5 hours) into the future.

In summary, the contributions of this research are:

1. The novel application of CMA-ES to evolve networks to forecast wind power generation, power demand and CO$_2$ levels.

2. The application of evolutionary neural networks to Ireland’s energy sector.

3. CMA-ES significantly outperformed PSO and DE in terms of convergence speed, accuracy and robustness. CMA-ES also outperformed other state of the art approaches on 5 out of 6 data sets.

4. The accuracy of the evolved network’s predictions decreases in a mostly linear fashion the further into the future the network attempts to predict. The evolved network does still posses a reasonable level of accuracy when predicting 2.5 hours ahead of time.
5.7.1 Future Work

There are many potential routes for future research that have arisen from this research. One such subject of future work would be to evolve networks that take into account wind data when making time series predictions.

Another promising avenue for future research would be to utilize the evolve forecasting model by implementing the model as part of a model predictive controller for scheduling power generators.
Chapter 6

Forecasting In The Cloud

The work outlined in this chapter was published in:


6.1 Introduction

Predicting resource utilization has been listed as one of the ten biggest obstacles facing the growth of cloud computing [15]. One of the major difficulties for prediction algorithms in cloud computing is that cloud resources are constantly changing and exhibit a complex dynamic behavior. Artificial Intelligence (AI) and Machine learning (ML) algorithms such as Neural Networks have been shown to improve upon traditional models, e.g. ARIMA. Forecasting methods such as ARIMA rely on patterns in historical data to make future predictions. These approaches are not suitable when there is not a distinct pattern in the data or if there is significant amount of random variation in the data [207]. AI and ML methods are much more adaptable and robust than these traditional approaches. The robustness of ML approaches, such as neural networks, is due to the few assumptions made and requirements they have about the data set.

It is estimated that by 2020, there will be 51,974 GB of internet traffic generated per second [181]. This staggering amount of internet traffic combined with the trend of technology companies moving towards offering computing as a service via cloud
computing will result in the generation of massive volumes of data being processed by hosts and Virtual Machines (VMs). The advantage of ML methods is that they can make use of this increased volume of data to make more accurate prediction models for cloud resources.

CPU utilization is one of the most important metrics for measuring the performance of host machines and is a popular metric for researchers to determine when predicting future host performance [404, 97, 43]. In virtualized environments, CPU is usually the resource with the highest level of demand and is therefore a major cause of resource shortages on cloud host machines. Many of these studies have examined one-step ahead forecasting using methods such as LOESS and Feed-forward Neural Network to predict CPU utilization at the next time step. However this one step ahead prediction time frame (usually 5 minutes ahead) leaves little time for the cloud resources to be adjusted. It has been shown in the literature that predicting a workload pattern with a high degree of accuracy on a short time scale (i.e., time steps of 5 minutes) is more difficult than long term forecasting (i.e., time steps of days or weeks) as resource in these short time scale can be extremely random [40]. The ability to accurately predict the demand on cloud resources further into the future in the short term is critical to a data centre’s efficiency and performance. Islam et al., have shown that the process of instantiating a new virtual machine takes between 5-15 minutes [184]. This is problematic as VMs need to be migrated from over-utilized hosts quickly. Leaving a VM to run on an over-utilized host for an extended period of time will result in the deterioration of the performance of that VM. Therefore it would be advantageous for the cloud management system to have the information of when the host machine will become over-utilized in advance to enable it to start up another host before the first host becomes over-utilized. This is one of the key ideas that has motivated this research. It is thought that some of the novel and powerful algorithms within the machine learning paradigm could be promising candidates to predict CPU utilization with greater accuracy.

In recent years machine learning algorithms have received a lot of attention and are a very active area of research due to the increased power of modern computers. New applications of machine learning algorithms are routinely being discovered. One of the most effective and diverse machine learning methods is the neural network [47, 160], which is inspired by the brain. Neural networks act as function approximators which makes them widely applicable to a broad range of problems from regression
to robotics. Neural networks are of interest in this research due to their ability to accurately make predictions for time series problems. There are numerous examples in the literature of successful applications of neural networks to time series problems from river flow forecasting [186] to electrical load forecasting [295]. It is for these reasons that neural networks will be implemented in this research to predict CPU utilization levels. One of the main issues when implementing neural networks is training the network weights. Training the weights to enable the network to produce the correct output for a given input is a complex optimisation problem. This is what motivates the use of swarm and evolutionary optimisation algorithms. Since the proposal of Genetic Algorithms (GA) by John Holland in the 1970’s [169], a large body of research has developed surrounding evolutionary algorithms due to their effectiveness as optimisation problem solvers. The task of optimising the weights of neural networks is a problem with continuous variables. There have been a number of highly effective swarm and evolutionary algorithms for solving continuous optimisation problems: Particle Swarm Optimisation by James Kennedy in 1995 [205], Differential Evolution in 1997 by Storn and Price [348] and Covariance Matrix Adaptation Evolutionary Strategy by Hansen and Ostermeier in 1996 [156, 157] have been shown to be the most effective of these optimisation algorithms. The advantage of these methods over traditional mathematical methods is that they are more robust and suitable to non-separable, noisy and non-convex optimisation problems as they do not rely on any problem gradient [400]. They are also capable of approximating very large NP-Hard problems (Non-deterministic in Polynomial time), i.e. problems that grow exponentially with the number of variables [81]. For these reasons these algorithms are widely used as problem solvers for complex continuous optimisation problems and will be used to train the neural network for predicting host CPU utilization in this research. To the best of our knowledge this is the first paper to apply these optimisation algorithms to train recurrent neural networks to predict host CPU utilization.

In this paper, we predict host CPU utilization using several different recurrent neural networks. The aims of this research is to:

1. Investigate the accuracy of evolving neural networks for predicting CPU utilization using PSO, CMA-ES and DE.

2. To establish the generality of the evolved networks by testing its accuracy using unseen data from the same and different hosts.
3. To determine how far into the future the networks can accurately predict host CPU utilization levels.

The outline of the paper is as follows. Section 2 gives an overview of forecasting in cloud computing, neural networks and swarm evolutionary optimisation algorithms. The experimental procedure will be explained in Section 3. Section 4 will present the experimental results. These results will then be discussed in Section 5. Finally, Section 6 will highlight what conclusions can be drawn from the research conducted and also outline some avenues for future research that have arisen from this paper.

6.2 Related Work

With resource usage in a constant state of flux, CPU utilization is difficult to predict for time scales in the range of minutes. This means that forecasting approaches such as one-step-ahead prediction strategy leave little time for the data centre to adjust resources required when bursts of high traffic occur. Predicting 20 or 30 minutes into the future with a high accuracy could inform data centre management systems to perform appropriate measures such as live migration or turning hosts on/off in time to deal with future cloud demand. The objective of this paper is to use evolutionary optimisation algorithms to evolve recurrent neural networks to predict host machines CPU utilization with a high degree of accuracy.

6.2.1 Forecasting in Cloud Computing

Predicting resource usage is an important tool for effective planning and to aid in counteracting future uncertainty. Forecasting and prediction techniques have been widely studied and applied to several different areas. Enke and Thawornwong successfully utilized neural network to forecast stock market returns [119] in 2005. In 2008, Jones et al. utilized neural networks to forecast patient volumes in the emergency department of a hospital [192].

In cloud computing one of the most well studied metric to forecast is CPU performance as it is one of the major causes of resource shortages on cloud host machines. Dinda and O’Hallaron used several forecasting models to predict tasks running times, based on CPU load predictions [97]. Beloglazov et al. proposed the implementation of
the Local Regression, Median Absolute Deviation, Inter-quartile Range and Robust Local Regression methods to determine when servers are likely to become over-utilized based on CPU utilization [38]. The focus of their work was the optimal scheduling of VM and hosts. The evolutionary neural network approach proposed in this paper for forecasting host CPU utilization could enhance the performance of the VM placement algorithm proposed by Beloglazov et al. and make for a more efficiently run data center. Zhang et al. presented a prediction method for grid tasks by providing a resource-oriented approach to predict the future performance of the resources and also a multi-step ahead CPU load prediction approach [404].

In recent years there has been a move towards integrating machine learning techniques to improve cloud computing efficiency. Machine learning approaches provide a unique way for cloud systems to adapt to sudden changes in resource consumption to improve resource management and develop highly scalable IT infrastructure. Several works demonstrate that ML algorithm can improve resource scaling, VM live migration [108, 107, 109, 106] and resource allocation [26, 25] in cloud computing. Neural networks are one of the most effective and versatile machine learning algorithms and have been successfully applied to areas within cloud computing such as scheduling [112], intrusion detection [371], DDoS attack defence [193] and load forecasting [299]. The ability of neural networks to approximate non linear functions enable them to perform well for complex function approximation problems. They are robust to noise in data and can accurately model intricate and complex patterns. This makes them ideal candidates for forecasting CPU utilization which is a highly irregular and noisy problem with very little pattern. Neural networks have previously been used to forecast resource demands in cloud computing. Huang et al. has previously applied the NARX neural network to improve scheduling decisions in grid environments [173]. This method uses a Nonlinear AutoRegressive with eXogenous (NARX) input model in combination with a neural network. Duy et al. employs a neural network predictor for optimising server power consumption in a data centre [112]. The feed forward neural network predicts future load demands based on historical demands to turn off unused servers and restarts servers to minimize the number of running servers, thus minimizing the energy usage. Prevost et al. implement neural networks and a linear predictor algorithms to forecast future workloads [299]. Kasabov and Song designed a fuzzy inference system using both online and offline knowledge for a time series prediction [199]. Their dynamic neural-fuzzy inference system can learn based on a set of rules. Bey et al. uses several models for time series prediction [43]. They use an
6.2. Related Work

Adaptive network to estimate the future value of CPU load for distributed computing. Their hybrid predictors were designed to perform for one-step-ahead prediction. The work presented in this paper builds on this work by forecasting both one-step and multi-steps ahead. Duggan et al. used a recurrent neural network trained with the algorithm known as back-propagation through time to predict CPU utilisation of a host on the Google Cluster trace data-set [111]. The work presented in this paper builds up this work to improve CPU prediction accuracy.

All of the examples highlighted above outline how neural networks are effective at addressing many of the problems in cloud computing, in particular CPU forecasting. The research presented in this paper makes the novel contribution of applying swarm and evolutionary algorithms to evolve neural networks for CPU forecasting. Based on the extensive literature conducted, there are no examples in the literature that utilize PSO, DE and CMA-ES to evolve neural network weights to predict CPU utilization.

PSO is a well known swarm intelligence optimisation algorithm. There are many examples in the literature where PSO has been successfully applied to cloud computing. Zhan et al., applied the PSO algorithm to address the problem of job scheduling in the cloud and found that PSO provided superior results than genetic algorithms, simulated annealing and ant colony optimisation [401]. Liu et al. use a variation of PSO to solve the load balancing problem in VMs [228]. They use their MAPSO algorithm which incorporated a novel mutation mechanism to give improved performance for job scheduling. Pandey et al., employs PSO as a scheduling heuristic to minimize the total cost of execution of application work-flows on cloud environments [291]. They compared their results with a Best Resource Selection heuristic and their results showed that the PSO based task-resource mapping achieved three times the cost savings when compared to the BRS based mapping for the application work-flow.

The differential evolution optimisation algorithm has also been applied to optimisation problems within the domain of cloud computing. Tsai et al. uses an improved differential evolution algorithm for multi-objective task scheduling and resource allocation in a cloud environment [359]. Their improved differential evolution algorithm performed better than state of the art multi-objective optimisation algorithms at obtaining Pareto-optimal solutions. CMA-ES has also been applied to optimise migration in a distributed cloud computing setting [383].

All of the studies outlined above implement PSO, DE and CMA-ES to optimise job scheduling, mapping work-flows or resource allocation. This paper utilizes these
algorithms within a cloud environment to evolve the weights of a neural network for CPU prediction. We found no examples in the literature where this has previously be done.

### 6.2.2 State of the Art

It is thought that the results presented in this paper could have a significant impact on the related research that applies neural networks in the cloud. Some recent studies include the implementation of a fuzzy neural network to predict cloud resource demand by Chen et al. [74]. This work utilizes backpropagation to train the network. The results presented here indicate the performance advantage of using evolutionary methods over backpropagation. It is thought that Chen et al. could further improve their fuzzy neural network by training it using evolutionary algorithms. The same is true for recent work by Song et al. that utilizes long short term memory (LSTM) for predicting host resource demand [341]. The authors here use back propagation to train their LSTM network. It is likely that the authors can improve the performance of their network using evolutionary approaches to train the network. Qazi et al. combine neural networks with ARIMA modelling to produce an auto regressive neural network (AR-NN) [405]. This network is trained using backpropagation which is demonstrated in this research to perform worse than evolutionary methods. Their proposed AR-NN could likely be improved if evolutionary methods were used to train the AR-NN. Duggan et al. trained a recurrent neural network using back-propagation through time to predict CPU utilisation [111]. Their results could also be improved upon by using evolutionary neural networks.

### 6.2.3 Neural Networks

Neural Networks is one of the most prominent fields of Machine Learning research. Neural networks are function approximators that are inspired by nature, in particular, the biological brain [47, 160]. Since they were first proposed in the 1960’s, neural networks have been applied to a wide range of problem domains including classification, regression, forecasting, control, learning and robotics. The standard feed forward network consists of an input layer of neurons, one or more hidden layer of neurons and an output layer of neurons. A neuron is a signal processing unit that reads in a number of signals as input and then outputs a signal using the sigmoid (or logistic)
function. The network senses its environment by reading a normalized signal into the input layer. In the research presented in this paper, this signal corresponds to the CPU demand on the host from previous time steps. This signal is propagated forward through the connected layers of neurons via weighted connections. Once the signal is propagated through the network, the network outputs the signal through the output layer of neurons. In terms of the host utilization prediction problem, this output signal corresponds to the CPU demand on the host in future time steps. This research will implement a particular type of neural network known as a Recurrent Neural Network (RNN), illustrated in Figure 6.1. Recurrent networks differ from the standard feed forward networks due to the recurrent connections between hidden neurons. These recurrent connections differ from the other network connections as they connect neurons within the same hidden layer. This gives the neural network memory which makes it particularly well suited to the problem of predicting CPU demand.

A signal is passed into the network through the input layer of neurons, i.e. the previous CPU demands for the host. This signal is then passed through the layers of hidden neurons and is then outputted from the final output layer. This process is referred to as a forward pass. Recurrent connections retain memory as a result of the hidden neurons retaining information from the previous forward pass. By conducting parameter sweeps, it was found that a network configuration of 3 hidden neurons provided the best performance. A fully connected recurrent neural network with 3 hidden neurons, 2 input neurons and 1 output neuron consists of 18 network weights which must be optimised.

As the signal is propagated through the network, it is adjusted by the weights of the connections between neurons. Aside from the input layer, a neuron in any other layer will have as input, the sum of the weighted signals that are outputted from other connected neurons. A neurons input signal is described by Equation 6.1.

\[ v_j = \sum_{i=1}^{N} w_{i,j} a_i \]  

(6.1)

where \( v_j \) is the input to a neuron in the \( j^{th} \) layer, layer \( i \) is the preceding layer to \( j \) that contains \( N \) neurons, each neuron in layer \( i \) has output \( a_i \) and each of these output signals are weighted by the value \( w_{i,j} \) as they are passed to each neuron in layer \( j \).
Figure 6.1: Recurrent Neural Network [240]. This figure illustrates a fully connected recurrent neural network. Neurons are connected by weighted synapses (or connections) that pass signals between neurons. The recurrent connections can be seen in the hidden layer of neurons. This gives the system memory. Image taken with permission from Mason et al. [240]

Each neuron $a_i$ outputs a value between 0 and 1. This output value is determined by the activation function of the neuron. The most commonly used activation function is the sigmoid function. This is described by Equation 6.2

$$a_j = \frac{1}{1 + \exp(-v_j)}$$

Neural networks have been implemented for the task of CPU demand time series prediction in this paper due to the past successful applications of neural networks for time series prediction. These include: financial forecasting [195], river flow forecasting [186], electrical load forecasting [295] and rainfall forecasting [127]. Based on the extensive literature review conducted there has been no research where neural networks have been applied to predict host CPU utilization in the cloud. This is the primary contribution of the research presented in this paper.

6.2.4 Swarm and Evolutionary Algorithms

This section will present the three state of the art optimisation algorithms that will be used to optimise the weights of the neural networks outlined in the previous section. There are many examples in the literature where Particle Swarm Optimisation,
Covariance Matrix Adaptation Evolutionary Strategy and Differential Evolution have been applied to train neural networks \[402, 178, 176, 242\]. One of the main contributions of this paper is to compare and contrast the performance of each of these algorithms on the task of forecasting CPU utilization using neural network.

**Particle Swarm Optimisation**

Of the three optimisation algorithms evaluated in this research, Particle Swarm Optimisation (PSO) is the only algorithm that falls under the heading of swarm intelligence. The PSO algorithm was first proposed by Kennedy in 1995 \[205\]. There exists many examples in the literature that successfully train neural networks using PSO \[245\]. The algorithm consists of a number of particles that move throughout the problem space evaluating potential solutions and move towards the best solutions. The algorithm is initialized by creating N particles with random positions \(\vec{x}_t\) and velocities \(\vec{v}_t\). At each iteration \(t\), each particle moves to a new position (i.e. solution) based on its velocity and previous position. In contrast, evolutionary algorithms typically use the operator’s crossover and mutation to evaluate new solutions. The fitness of a position is calculated based on an objective function. This fitness determines which positions are good candidate solutions. Each particle will update its position based on its best personal position \(\vec{p}_b\) and that of its neighbourhood \(\vec{g}_b\). After a predetermined number of problem evaluations, the particles should converge on the best-found location using the equations of motion outlined in Equation 6.3.

\[
\begin{align*}
    \vec{v}_{t+1} & = \chi \left( \vec{v}_t + r_1 c_1 (\vec{p}_b - \vec{x}_t) + r_2 c_2 (\vec{g}_b - \vec{x}_t) \right), \\
    \vec{x}_{t+1} & = \vec{x}_t + \vec{v}_t
\end{align*}
\]  

(6.3a) (6.3b)

where \(c_1\) and \(c_2 = 2.05\) are acceleration coefficients, \(r_1\) and \(r_2\) are random numbers between 0 and 1, \(\vec{p}_b\) is the best location of the current particle and \(\vec{g}_b\) is the best location found within the particle’s neighbourhood. The \(\chi\) term is the constriction
factor and is defined in Equation 6.4.

\[
\chi = \frac{2}{2 - \varphi - \sqrt{\varphi^2 - 4\varphi}},
\]

(6.4a)

\[
\varphi = c_1 + c_2.
\]

(6.4b)

where \( \chi \approx 0.72984 \), and \( c_1 = c_2 = 2.05 \) [76]. As the particles move around the problem space and evaluate candidate solutions, they should eventually converge on the best position. The constriction factor guarantees convergence. The PSO parameters outlined here are considered to be standard [54]. Algorithm 11 describes the functionality of the PSO algorithm. In this research \( N = 50 \) was found to give the best performance as is commonly used [251].

Create \( N \) particles with random position and velocity

while Evaluation e < Emax do

<table>
<thead>
<tr>
<th>for Particle = 1 to N do</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update personal best position ( p_{b_t} )</td>
</tr>
<tr>
<td>Update neighbourhood best position ( g_{b_t} )</td>
</tr>
<tr>
<td>Evaluate particle’s current position ( x_t )</td>
</tr>
<tr>
<td>Update particle’s velocity ( v_t )</td>
</tr>
<tr>
<td>Update particle’s position ( x_t )</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

end

Return best solution

Algorithm 11: PSO Algorithm. Pseudocode taken with permission from Mason et al. [241]

Covariance Matrix Adaptation Evolutionary Strategy

The Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) was first proposed in 1996 by Hansen and Ostermeier [156, 157]. Similar to PSO, CMA-ES was proposed as a method to solve global and non-linear optimisation problems. The CMA-ES algorithm falls within a class of algorithms known as Evolutionary Algorithms (EAs) which are inspired by the biological process of evolution and uses evolutionary operators such as crossover, mutation and selection. CMA-ES is initialized by randomly sampling a number of solutions to the optimisation problem. The solutions are then ranked according to their fitness. The mean \( (m) \), covariance matrix \( (C) \) and step size \( (\sigma) \) are then updated by increasing the likelihood of sampling good solutions (based on the pseudocode in Algorithm 12). The covariance matrix is an \( n \times n \) matrix that determines the variation around the mean, where \( n \) is the number
of dimensions in the problem. In Algorithm 12, $p_\sigma$ is the path for $\sigma$, $p_c$ is the path for $C$, $\omega$ are the recombination weights such that $\omega_1 \geq \omega_2 \geq ... \geq \omega_\mu > 0$ sum to 1, $y_\omega$ is the move of the population mean, $\lambda = 10$ is the population size, $\mu$ is the number of samples selected for the update, $I$ is the identity matrix and $N(m, \sigma^2 C)$ is the multivariate normal distribution. Parameter tuning found that $\sigma_0 = 0.1$ and $\mu = 0.5\lambda = 5$ gave the best performance. Standard values for other constants were implemented: $c_c = 4/n$, $c_\sigma = 4/n$, $c_1 = 2/n^2$, $c_\mu = \mu_\omega/n^2$, $d \approx 1$ and $\alpha = 1.5$.

Initialize: Sample $\lambda$ random solutions
Initialize: $C_0 = I$, $p_c = 0$ & $p_\lambda = 0$

while Evaluation $e < E_{\text{max}}$ do
  for Solution $i = 1$ to $\lambda$ do
    $x_i = m + \sigma y_i \sim N(m, \sigma^2 C)$
    $f_i = \text{fitness}(x_i)$
  end
  Sort solutions $x$ according to fitness
  $m \leftarrow m + \sigma \sum_{i=1}^\lambda \omega_{p(i)} y_i$ = $m + \sigma y_\omega$
  $p_\sigma \leftarrow (1 - c_\sigma)p_\sigma + \sqrt{1 - (1 - c_\sigma)^2 \mu_\omega} C^{-0.5} y_\omega$
  $\sigma \leftarrow \sigma \times \exp\left(\frac{\sigma}{\sigma\|N(0,1)\|} - 1\right)$
  $p_c \leftarrow (1 - c_c)p_c \|0, \alpha\sqrt{\mu}\| \|\|p_c\|\|^2 \sqrt{1 - (1 - c_c)^2 \mu_\omega} y_\omega$
  $C \leftarrow (1 - c_1 - c_\mu)C + c_\mu \sum_{i=1}^\lambda \omega_{p(i)} y_i y_i^T + c_1 p_c p_c^T$
end
Return best solution

**Algorithm 12**: Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) Algorithm

CMA-ES has been applied to many real world problems since its first proposal including optimising building placement for solar energy [198], laser pulse shaping [335], antenna design [191] and neural networks [176].

**Differential Evolution**

Similar to CMA-ES, Differential Evolution (DE) is also a global optimisation algorithm that falls under the heading of evolutionary computing. The DE algorithm was first developed in 1997 by Storn and Price [348]. DE does not rely on any gradient information about the problem and is therefore suitable for optimisation problems that contain noise. DE is a relatively simple optimisation algorithm to implement when compared to CMA-ES. The simplicity and robustness of DE make it a very popular optimisation algorithm and suitable for optimising network weights. Since its creation DE has been applied to many real world problems such as robotics [65] and energy systems [246]. There are many examples applications of DE to neural network weight optimisation [178, 2, 103, 288].
At each iteration, the current agent’s position (i.e, solution) is combined with three other distinct agents’ positions to produce a new position \( \mathbf{y}_i \). The functionality of DE is outlined in detail in Algorithm 13. If the new position of an agent has a better fitness than that agent’s previous position, the agent moves to the new position. This is then repeated for each of the agents until a predetermined number of problem evaluations has been conducted. In Algorithm 13 the parameter \( CR = 0.9 \) is the crossover probability, \( F = 0.5 \) is the differential weight and \( N = 20 \) is the number of agents.

```
Initialize N agents with random positions
while Evaluation e < Emax do
    for Agent = 1 to N do
        Select 3 other agents A,B and C
        Select random dimension index R
        for dimension i = 1 to D do
            generate random number \( r \in [0,1] \)
            if \( r < CR \) Or \( i = R \) then
                new position \( y_i = a_i + F \times (b_i - c_i) \)
            else
                \( y_i = x_i \)
            end if
        end
        if fitness(\( \mathbf{y} \)) < fitness(\( \mathbf{x} \)) then
            replace \( \mathbf{x} \) with \( \mathbf{y} \)
        end if
    end
end
Return best solution
```

**Algorithm 13:** Differential Evolution (DE) Algorithm. Pseudocode taken with permission from Mason et al. [240]

### 6.3 Experimental Setup

This section will outline the various experiments that were conducted and the implementation of each algorithm.

#### 6.3.1 Network Parameter Selection

A recurrent neural network was implemented to predict CPU utilization. This was selected over a feed forward neural network because the recurrent connections give the system memory which is advantageous for the task of CPU utilization prediction. The network has three hidden neuron and two inputs for the current and previous CPU utilization. Parameter sweeps revealed that any more than three hidden neurons did not lead to any increase in performance and resulted in a longer training time due
to the additional number of weights to be trained. Parameter sweeps also found that more than two inputs did not lead to any increase in performance. The network had one output that corresponded to the network’s prediction of future CPU utilization.

6.3.2 Network Training

The weights of the network corresponded to a potential solution for each of the optimisation algorithms outlined in the previous section. A potential solution is referred to a particle’s position in PSO, a candidate solution in CMA-ES and an agent’s position in DE. Each algorithm will evaluate 10,000 network weight configurations before the network with the highest prediction accuracy is returned and evaluated on the test data. This process is repeated over 10 runs to ensure statistically significant results.

6.3.3 Data Models

The CPU utilization training and testing data were generated by the PlanetLab files [60]. These files contain CPU utilization values measured every 5 minutes in PlanetLab’s VMs. Each file contains 288 values and each value relates to five minutes in a 24 hour day. There are 10 folders, each folder contains between 898 to 1516 files. The planet lab data trace files can be obtained from their github [37]. These are used by the CloudSim simulator [61, 60].

Three data sets were generated from the planet lab files. The first data set generated was the training data which contained 2296 values or 8 days worth of CPU data. The second data set (shown in Figure 6.6) contained 288 CPU values (1 days workload) and was used to test the Neural Networks after the training phase had been completed. The training and the second (testing) data sets are apart of the same host. However the third data set which was used to also test the Neural Networks has come from a different host 2296 CPU values (8 days worth of CPU data). The reasoning behind having two test data sets from two different hosts was the test the generality and adaptability of the Neural Networks on completely new host CPU data.

6.3.4 Comparative Forecasting Methods

In order to put the performance of evolutionary neural networks in context, the neural networks trained with PSO, DE and CMA-ES will be compared to:
1. Random walk forecasting (RWF).

2. Moving Average (MA).

3. Linear Regression (LR).


These methods are utilized as a benchmark to compare swarm and evolutionary methods to as they are state of the art forecasting algorithms.

Random walk forecasting is the most basic forecasting method that will be implemented. This approach consists of predicting the next future value as equal to the current observed value.

The moving average method is another commonly using forecasting approach. This method consists of predicting a future value by averaging $n$ previous values.

Linear regression is one of the most commonly used machine learning algorithm for forecasting. Linear regression consists of forming a linear model of a data set whereby the parameters of the model are estimated using the data [269].

The final method that will be used to predict CPU utilization is a neural network with backpropagation [382]. Backpropagation is the most popular method of neural network training and is suitable for supervised learning problems. The algorithm works by calculating the error between the target output and the observed output. This error is then propagated back through the network and is used to update the weights. Backpropagation is implemented in this research on a network with 2 inputs, 3 hidden neurons and 1 output, as with all other neural network implementations.

### 6.3.5 Experiments Conducted

The first experiment will involve comparing the performance of each optimisation on the training data. All evolutionary neural networks will be judged based on their rate of convergence, prediction accuracy, performance consistency and ease of implementation.

The second experiment will evaluate the performance of each trained network on previously unseen data. The purpose of this is to test if the trained networks are
capable of giving a good general performance and do not only perform well on the training data.

The third experiment will evaluate how far into the future the network can predict. It would be advantageous in data centres to know well in advance how much CPU will be required of a given host. This experiment will evaluate the accuracy of the network for predicting CPU utilization further than one step into the future.

Finally, the fourth experiment will further test the generality of the trained networks. Once the network has been trained on a host’s CPU utilization data set, the network will then be tested on new data from a completely separate host. The distinction between this experiment and experiment two is that in experiment two, the network was evaluated on new data from the same host. Testing the performance of the network on a completely new host will give insight into how general the networks are at predicting CPU levels and also establish if there are commonalities in the CPU utilization data from different hosts.

6.4 Experimental Results

This section presents the results of each of the experiments outlined above followed by a discussion in order to gain insight and highlight their significance. The two tailed t test with a significance level of $\alpha = 0.05$ was conducted to determine significant performance differences when comparing algorithms. All values were rounded to 4 decimal places.

6.4.1 Training Data

The convergence of each algorithm evaluated can be seen in Figure 6.2. This figure illustrates the best fitness of each algorithm as they evaluate the training data over 10,000 evaluations. It is obvious from this graph that PSO performs the worst of the 3 algorithms. CMA-ES performs best of the 3, however, DE provides very similar performance in terms of both rates of convergence and final fitness.

The predictions of each algorithm are plotted along the actual CPU utilization in Figure 6.3 for the first 500 training steps. The predictions for the remaining training data can be seen in the Appendix. There is an interesting difference in the prediction
values for the network trained with PSO than those trained with CMA-ES or DE. The network trained using PSO performs significantly worse at predicting the lower CPU values than either CMA-ES or DE, as seen in Figure 6.3.

When predicting the higher CPU values, neither CMA-ES or DE trained networks predict CPU utilization values higher than approximately 0.8. It is thought that the reason for this is that when the CPU utilization is high, the CPU values oscillate rapidly between 0.6 and 1.0. Each of the networks trained seems to struggle with this rapid change in CPU utilization and as a result found that predicting approximately 0.8 when the CPU utilization is high is the best overall policy.

It appears that the PSO trained network does in parts of the data match these changing high CPU values. Table 6.1 however shows that the PSO trained network performs worse overall.

Table 6.1 displays the Mean Absolute Error (MAE) and Mean Squared Error (MSE) for each of the algorithms tested. As is clearly evident from Table 6.1, the random walk, moving average, linear regression and backpropagation neural network performs significantly worse than all evolutionary neural network approaches. Random walk performed the worst overall followed by moving average, linear regression and then backpropagation. This further validates the choice of evolutionary neural networks for CPU utilization time series prediction. When comparing the performance of each of the neural networks, statistical testing reveals that CMA-ES performs significantly better than DE and PSO. PSO performs statistically worse than both CMA-ES and DE. CMA-ES provides the best solution and also converges the fastest on the training
6.4. Experimental Results

Figure 6.3: Host Utilization Predictions for First 500 Steps of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for the first 500 training data time steps. The predictions for the remaining training data can be found in the appendix.

data. CMA-ES and DE also have the lowest standard deviation meaning that they give the most consistent prediction accuracy. The box plots in Figure 6.4 illustrate the spread in the MAE produced by each algorithm. From this diagram it is clear that the evolutionary methods evaluated here perform significantly better than moving average, random walk, linear regression and backpropagation.

Table 6.1: Training Data Accuracy

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MAE (Std Dev)</th>
<th>MSE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO</td>
<td>0.1521 (0.0021)</td>
<td>0.0481 (0.0003)</td>
</tr>
<tr>
<td>DE</td>
<td>0.1422 (0.0008)</td>
<td>0.0454 (0.0004)</td>
</tr>
<tr>
<td>CMA-ES</td>
<td>0.1414 (0.0008)</td>
<td>0.0452 (0.0005)</td>
</tr>
<tr>
<td>Random Walk</td>
<td>0.1746 (0.0000)</td>
<td>0.0594 (0.0000)</td>
</tr>
<tr>
<td>Moving Avg</td>
<td>0.1715 (0.0000)</td>
<td>0.0572 (0.0000)</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>0.1641 (0.0000)</td>
<td>0.0498 (0.0000)</td>
</tr>
<tr>
<td>Backpropagation</td>
<td>0.1509 (0.0010)</td>
<td>0.0462 (0.0004)</td>
</tr>
</tbody>
</table>

6.4.2 Test Data

The next set of experiments involved evaluating the performance of the networks trained with each algorithm on previously unseen test data. The goal of this experiment was to establish how each network performs on new data from the same host as it was trained on. Figure 6.5 compares the predictions of each network with the actual CPU values. Similar observations can be made from this graph as can be made from the training data in Figure 6.3, i.e. CMA-ES and DE can better predict lower CPU values. The ability of each network to perform well on the unseen test data
Chapter 6. Forecasting In The Cloud

Figure 6.4: Forecasting Accuracy Spread. These figures depict box plots for each of the algorithms implemented. These graphs highlight the statistical differences in the performance of every algorithm.

confirms the generality of the trained neural networks.

The accuracy of each algorithm of the test data can be seen in Table 6.2. As with the training data, the non evolutionary methods perform statistically worse. Backpropagation performed the worst overall followed by moving average, random walk and finally linear regression. Interestingly backpropagation performed the worst on the test data but performed third worst on the training data in the previous section. This indicates that the network trained using backpropagation does not generalize well to unseen data. Of the evolutionary neural networks, the network trained using PSO also performed worst of the three optimisation algorithms. A more interesting observation, however, is the performance of DE and CMA-ES. When evaluated on the training data CMA-ES performs statistically better than DE, however when evaluated on the test data DE performs statistically equal to CMA-ES. This points to the well-known problem in machine learning research of over training. The reason CMA-ES performs significantly better than DE on the training data but equal on the test data is that CMA-ES over trains the network on the training data to the point where gains in accuracy on the training data do not correspond to gains in accuracy in
6.4. Experimental Results

Figure 6.5: Host Utilization Predictions for Test Data. This figure illustrates the predicted host utilization of each evolved neural network on unseen test data.

The final observation to be made from Table 6.2 is that all of the standard deviations for each of the networks are higher than they were for the training data. This is to be expected however due to the fact that the networks are being evaluated on unseen data so more deviation in the accuracy is to be expected.

Table 6.2: Test Data Accuracy

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MAE (Std Dev)</th>
<th>MSE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO</td>
<td>0.1564 (0.0023)</td>
<td>0.0483 (0.0002)</td>
</tr>
<tr>
<td>DE</td>
<td>0.1495 (0.0018)</td>
<td>0.0465 (0.0013)</td>
</tr>
<tr>
<td>CMA-ES</td>
<td>0.1498 (0.0013)</td>
<td>0.0468 (0.0008)</td>
</tr>
<tr>
<td>Random Walk</td>
<td>0.1733 (0.0000)</td>
<td>0.0602 (0.0000)</td>
</tr>
<tr>
<td>Moving Avg</td>
<td>0.1756 (0.0000)</td>
<td>0.0577 (0.0000)</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>0.1704 (0.0000)</td>
<td>0.0509 (0.0000)</td>
</tr>
<tr>
<td>Backpropagation</td>
<td>0.1892 (0.0123)</td>
<td>0.0558 (0.0050)</td>
</tr>
</tbody>
</table>

6.4.3 Multi-step Ahead Prediction

The aim of the experiment conducted in this section was to evaluate how far into the future the neural network could predict CPU utilization and to establish how much the accuracy of the prediction decreases. Since CMA-ES trained neural network had the best accuracy in the two previous results sections, the CMA-ES neural network
was implemented to predict CPU utilization for multiple steps into the future in this experiment. This experiment involved predicting the CPU utilization at 1, 2, 3 and 4 steps into the future where each step corresponds to 5 minutes. Table 6.3 presents the accuracy of the prediction at each of the future points in time. As is expected, the accuracy of the future predictions decreases steadily the further into the future the network attempts to predict. This is true for both the training and testing data.

Table 6.3: Multi-Step Prediction Accuracy

<table>
<thead>
<tr>
<th>Number of Steps</th>
<th>Training MAE (Std Dev)</th>
<th>Training MSE (Std Dev)</th>
<th>Test MAE (Std Dev)</th>
<th>Test MSE (Std Dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Step Ahead</td>
<td>0.1414 (0.0008)</td>
<td>0.0452 (0.0005)</td>
<td>0.1498 (0.0013)</td>
<td>0.0468 (0.0008)</td>
</tr>
<tr>
<td>2 Step Ahead</td>
<td>0.1794 (0.0011)</td>
<td>0.0669 (0.0008)</td>
<td>0.1817 (0.0020)</td>
<td>0.0647 (0.0013)</td>
</tr>
<tr>
<td>3 Step Ahead</td>
<td>0.2033 (0.0011)</td>
<td>0.0816 (0.0004)</td>
<td>0.2087 (0.0023)</td>
<td>0.0825 (0.0011)</td>
</tr>
<tr>
<td>4 Step Ahead</td>
<td>0.2206 (0.0008)</td>
<td>0.0917 (0.0003)</td>
<td>0.2289 (0.0029)</td>
<td>0.0919 (0.0021)</td>
</tr>
</tbody>
</table>

Figure 6.6 displays the graphs. The top graph shows the actual CPU demand for the host. The bottom graph displays the absolute error of the prediction at each time step for both 1 and 4 steps ahead on the test data. The purpose of this graph is to illustrate at what points in the CPU utilization data does the accuracy of the network increase and decrease. Figure 6.6 reveals that the largest prediction errors occur when there are large and instantaneous changes in the CPU utilization of the host. An example of this is just before the 100th time step. At this point on the top graph, it can be seen that the utilization drops from 1 to 0 very suddenly. At the corresponding point on the bottom graph, it can be seen that there is a very high spike in the error of both the 1 and 4 step ahead prediction. This is because the network finds it very difficult to cope with these instantaneous and drastic changes in CPU utilization. At other time steps where the change is less severe or less instantaneous, the corresponding prediction error is much lower. The final observation to note from Figure 6.6 is the difference between the prediction error of the 1 and 4 step ahead predictions. Whenever there is a large error spike such as the spike in error midway between the 200th and 250th-time step, the 1 step ahead error spike is much shorter in duration than the 4 step ahead error. The reason for this is simply that for the 1 step ahead prediction, the large error is discovered at the next time step and the network can adjust its prediction to accommodate for the sudden change in CPU utilization. For the 4 step ahead prediction, however, the sudden change in CPU utilization is not discovered for another 4 time steps and therefore the prediction error stays larger for a longer duration. This is to be expected however given the nature of the CPU utilization data.
6.4. Experimental Results

Figure 6.6: Host Utilization Predictions for First 500 Steps of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for the first 500 training data time steps. The predictions for the remaining training data can be found in the appendix.

6.4.4 New Host Evaluation

The final experiment involved testing the performance of a network on separate host data than that which it was trained on. It was already established that the trained network can generalise to predict new CPU data with similar accuracy when tested using new data from the same host. This experiment aims to test if the trained network is general enough to accurately predict new and unseen CPU values from a completely different host.

The accuracy obtained testing the trained network on CPU utilization data from a new host was: $\text{MAE} = 0.1442 \pm 0.0009$, $\text{MSE} = 0.0475 \pm 0.0007$. When these results are compared to those in Table 6.2, it is evident that the network trained on one host is in fact general enough to be applied to other hosts and still achieve a similar level of accuracy. This result implies that there are similarities between different hosts in a data centre. The CPU demand profile for the various hosts has similar characteristics that the network can exploit these when making predictions on a separate host than
the host on which it was trained. This is beneficial from a practical point of view. It
would clearly be advantageous to not have to train a neural network for each host in
the data centre to make accurate predictions. The results presented in this section
indicate that training multiple neural networks may not be necessary.

6.5 Discussion

The results show that the evolutionary neural networks have the capabilities to im-
prove upon traditional prediction methods to predict CPU utilization with a high
degree of accuracy. This is true for both one-step and multi-step prediction.

The purpose of the first experiment conducted was to demonstrate how evolutionary
neural networks could outperform tradition forecasting methods such as random walk,
moving average, linear regression and backpropagation neural networks. The results
show that despite the large amount of noisy present in the CPU utilization data,
the CMA-ES, PSO and DE trained neural networks could produce more accurate
forecasts than other. The CMA-ES had the best accuracy when evaluated on the
training data.

The second experiment conducted was to establish the generality of each method
by applying the forecasting models to previously unseen data. In this experiment,
the evolutionary methods again provided the best forecasting accuracy. The best
performing algorithms here were both DE and CMA-ES which performed equally
optimally.

The third experiment conducted was to establish how much the prediction accuracy
of the evolved neural network decreases as they attempt to predict further into the
future. The results show that it is indeed possible to predict multiple time steps into
the future while still retaining a reasonable degree of accuracy. The CMA-ES trained
neural network obtained a reasonably high level of accuracy when predicting 4 time
steps into the future (20 minutes). As expected the further into the future results in
a significant decrease in prediction accuracy.

Multi-time step a head forecasting has proven to be difficult area in time series re-
search. The additional noise present in the CPU utilization data makes forecasting
far into the future even more difficult. The evolutionary neural network forecasting
method presented in this research could potentially be incorporated with many of the
other sub fields of cloud computing that involve host migration and VM scheduling to improve overall performance.

It is known that instantiating a new virtual machine takes between 5-15 minutes [184]. The results presented in this paper demonstrate that evolutionary neural networks are capable of predicting CPU data 20 minutes ahead with a relatively high degree of accuracy despite the noise in the data set. The evolved networks could be implemented to inform the cloud management system as to when a host is going to become over-utilized so the management system can take action and boot up new VM instances on different hosts prior to initiating live migration from the over-utilized host. This in-turn, would lead to smoother transitions of VMs being moved from a source host to a destination, reducing live migration times and decrease the occurrences of SLAs on host machines. This is a promising avenue of future research that has arisen from this paper.

Power consumption is another area where accurate forecasting could enhance the performance of cloud data centre. Research by Koomey has stated that in 2010 1.3% of all power consumed worldwide was due to data centre usage [217]. Gartner et al. highlighted that the ICT industry contributed to about 2% of global CO2 emitted each year, aligning itself on the same level with the aviation industry [179]. More effective optimisation of cloud data centres could dramatically decrease energy consumption. It has been shown in data centres that a significant portion of the host machines operate at 10-50% of their full capacity [27]. This results in a considerable increase in energy costs. Duy et al. have shown how neural networks can be utilized as a predictor to reduce energy consumption in a data centre to turn off host when the traffic load is light [112]. The results presented in this paper demonstrate that evolving neural networks can result in more accurate forecasts. These more accurate forecasts could therefore have the additional benefit of reducing the overall energy consumption of the data center if incorporated with VM migration. For example, host machines which are predicted to be idling at 0-10% utilization for the next 20-30 minutes could be shut down to reduce energy consumption and by extension CO2 emissions from powering the cloud data centres. Companies such as Google have recently implemented their own DeepMind neural network tool to reduce their data centre energy cost by 40% [133].

The results from the final experiment highlight the generality of the proposed evolved neural networks. When evaluated on previously unseen data from a completely new
host, the evolved neural network performed with an accuracy comparable to the original host. This is advantageous as it implies that it may not be necessary to evolve a neural network for every host in the data center to make accurate forecasts. Evolving just a single neural network can provide accurate forecasts for future CPU utilization for each.

6.6 Conclusion

The primary aim of this research was to investigate if neural networks are capable of accurately predicting CPU utilization for short time. The results obtained indicate that it is, in fact, possible to predict CPU utilization with a high degree of accuracy for short time periods and on data sets that have sudden extreme changes. All three algorithms (PSO, DE and CMA-ES) were able to train a network to accurately predict CPU utilization within 10,000 evaluations of the training data. Of the three algorithms evaluated CMA-ES performed the best on the training data followed by DE and lastly PSO. On the test data, however, CMA-ES and DE performed equally well. It was discovered however that the prediction of the CPU utilization is a difficult task due to the occasional sudden extreme change in CPU utilization. All prediction algorithms struggled to predict these rapid changes. It was also shown that the evolved networks were also capable of accurately predicting were given new CPU utilization data from both the same and new hosts. This indicates that there are some reoccurring patterns and regularities in the CPU utilization data that the networks are capable of exploiting to give a good general performance. It was also found that the networks prediction accuracy does decrease as it predicts further into the future however on average the network is capable of predicting with a reasonable level of accuracy 4 steps (20 minutes) into the future.

In summary, the contributions of this research are:

1. It is possible to evolve a network capable of accurately predicting host CPU utilization.

2. CMA-ES converged the fastest and gave the best network prediction accuracy on the training data. On the test data, CMA-ES and DE provide a statistically equal network prediction accuracy. CMA-ES would be the algorithm of choice if rapid convergence is desired however DE is far simpler to implement and gives
an equal performance on the test data.

3. The evolved networks are capable of accurately predicting CPU utilization on unseen data from both the same and also new hosts. This demonstrates a high degree of generality.

4. The accuracy of the network predictions decrease in a linear fashion as the networks attempt to predict further into the future. This is due to occasional severe changes in CPU utilization.

6.6.1 Future Work

There are many potential routes for future research that have arisen from this research. The first route would be to evaluate the evolved neural network predictors in other areas of cloud computing, e.g. to predict the amount of RAM and disk utilization in hosts.

This research was conducted using the Planet Lab data set for CPU utilization. It would be advantageous to further validate the proposed evolutionary neural network approach using other CPU utilization data sets. It would also be beneficial to test the prediction accuracy of a network trained on the Planet Lab data set using a new data set from a new data centre.
Chapter 7

Conclusion

The hypothesis stated in the introduction was that evolutionary neural networks can be successfully applied to address control and forecasting problems in the areas of energy systems and the environment.

Each experimental chapter in this thesis explored the application of evolutionary neural networks to a problem in energy systems or the environment. In Chapters 2 and 3, it was demonstrate that a network trained with PSO, NDE, NEAT and ESP can successfully allocate water to each interested party without any constraint violations. This was a new application domain for evolutionary neural networks.

Chapter 4 demonstrated that evolutionary neural networks are capable of learning to generate Pareto fronts for the multi-objective task of power generation. This is the first application of evolutionary neural networks to the DEED problem.

When forecasting Ireland’s energy needs in Chapter 5, it is shown that evolutionary neural networks can outperform many state of the art forecasting algorithms. In particular, a network trained with CMA-ES outperforms all other forecasting methods on 5 out of 6 data sets when predicting Ireland’s power demand, wind power generation and carbon dioxide levels.

The final set of experimental results in Chapter 6 echos those of Chapter 5 and reveals that a network trained with CMA-ES outperforms all other forecasting methods when predicting CPU utilization in data centers. This is a crucial task as data centers consume vast amounts of energy. Accurately predicting CPU utilization can allow more efficient use of servers and therefore reduce energy consumption.
7.1 Summary of Thesis Achievements

In every chapter, it was found the evolving neural networks can be of benefit to whatever energy systems or environmental problem evaluated. This confirms the initial hypothesis at the beginning of the thesis that evolutionary neural networks can have a positive impact on energy systems and the environment.

**7.1 Summary of Thesis Achievements**

**Chapter 2:**

1. The largest meta optimisation analysis of PSO velocity update equations to date.
2. The first application of a PSO trained neural network to the problem of watershed management is conducted.
3. Commonly used PSO parameters do not always result in the best performance.
4. The proposal of novel hybrid PSO velocity update equations such as the Attractive Repulsive PSO with Avoidance of Worst Locations.

**Chapter 3:**

1. The proposal of a combined GA - DE evolutionary strategy for neural network topology and weight optimisation, i.e. Neuro Differential Evolution (NDE). NDE outperforms both the state of the art NEAT and ESP algorithms.
2. The first application evolutionary neural networks to watershed management is conducted. NDE produces a network with the best overall fitness.

**Chapter 4:**

1. A Multi-Objective Neural Network trained with Differential Evolution (MON-NDE) is proposed for dynamic multi-objective optimisation problems.
2. The first application evolutionary neural networks to DEED is presented.
3. A new fitness function with Pareto penalty is proposed. It is experimentally demonstrated that this fitness function places evolutionary pressure on the algorithm to produce viable Pareto fronts.
4. A comparison between neural network topology and weight optimisation versus solely weight optimisation is conducted using DE and a variant of NDE. This comparison shows that optimising the topology leads to slower convergence in a fixed offline learning environment. When a power generator failure is simulated during the evolutionary process, topology and weight optimisation with NDE can better adapt to the changed environment than simply optimising with weights with DE.

5. MONNDE can produce Pareto fronts with no further optimisation required when given new power demands different to the power demands it was trained on.

Chapter 5:

1. The first application of a CMA-ES trained neural network to forecast wind power generation, power demands and carbon dioxide is conducted.

2. The first application of evolutionary neural networks to Ireland’s energy sector is also presented.

3. A CMA-ES evolved network performs better than all other forecasting methods on 5 out of 6 data sets.

4. The evolved network maintains a reasonable forecast accuracy when predicting further into the future (2.5 hours).

Chapter 6:

1. The first application of a CMA-ES trained neural network to the task of CPU utilization prediction is presented and performs better than all other methods on the training data but performs equal to DE on the test data.

2. The evolved networks perform equally well on unseen data from the same and from a new host entirely. This indicates that they generalize well.

3. Due to the sudden severe changes in host CPU utilization, the accuracy of the network decreases in a near linear manner as the networks predict further into the future.
7.2 State of the Art

This thesis builds upon recent research in a number of areas. Chapter 2 presents the most comprehensive meta optimisation analysis to date implementing 20 PSO variants evaluated over 8 problems. This is significantly larger than those in the current literature [256, 297]. The NDE algorithm proposed in Chapter 3 takes inspiration from the NEAT [347] and EPNet [396] algorithms for design features such as network growth, speciation and topology and weight optimisation. The MONNDE algorithm proposed in Chapter 4 builds on previous studies that evolve neural networks in multi-objective environments for games [324] and hybrid fuel cell turbine control [80]. Chapter 5 demonstrates that networks evolved with CMA-ES can provide significantly higher accuracy for forecasting in the energy sector than other approaches in the literature [393, 18]. This chapter pursues the future work suggested by Corne et al. that suggests using CMA-ES for short term wind forecasting [83]. Similarly, the results presented in Chapter 6 demonstrate that a CMA-ES trained network can provide higher prediction accuracy than a backpropagation trained network, which is used in many previous studies [74, 341, 405].

7.3 Limitations

7.3.1 Computational Cost of Meta Optimisation

Meta optimisation is a useful tool for finding good parameter settings for optimisation algorithms such as PSO. The drawback with using meta optimisation however is that it is very computationally expensive. If many evaluations of the problem are required or if each problem evaluation is computationally expensive, meta optimisation may not be viable. Of course meta optimisation does not guarantee that the optimum parameters will be found as is the case for all optimisation problems.

7.3.2 Choice of Meta Optimisation Algorithm

When implementing meta optimisation, the question arises of what optimisation algorithm should be implemented for the task of meta optimisation? This is not clear, there are advantages and disadvantages of using evolutionary approaches such as PSO or DE versus traditional gradient based methods. Traditional gradient based methods
may provide faster convergence which is desirable since meta optimisation is computa-
tionally expensive. Global search algorithms are less prone to converging onto a local
optima however which increases the chances of finding the best parameters. This
is after all the purpose of meta optimisation. Another concern when implementing
meta optimisation is how to choose the parameters of the meta optimisation algorithm
that is optimising the base optimisation algorithm. Would this ultimately result in
an infinite number of optimisation algorithms applied to one another? The way to
circumvent this issue is to select parameters for the meta optimiser that are known
to provide good performance, albeit not optimal. This is however a fundamental
limitation of meta optimisation.

7.3.3 Neuro Differential Evolution and Large Networks

The experiments presented in the thesis only evaluate the proposed NDE algorithm
on problems that require shallow networks. Evolving large deep networks was outside
of the scope of this research however. It is thought that NDE would needed to
be extended in a similar manner as the NEAT algorithm was. The hyperNEAT
algorithm extends NEAT to evolve larger networks by implementing compositional
pattern-producing network (CPPNs). A similar approach may be applicable to NDE.
In its current form however NDE is limited to shallow networks.

7.3.4 MONNDE and Large Networks

Similar the NDE experiments, those for MONNDE do not evaluate the algorithm for
large scale networks. Again it is likely that MONNDE will need to be modified to
scale to larger problems. This is outside of the scope of this thesis but is a limitation
of the current MONNDE algorithm.

7.3.5 Diversity of Pareto fronts produced by MONNDE

Another drawback of the MONNDE algorithm in its current form is the lack of diver-
sity in the Pareto front. As the purpose of multi-objective optimisation is to produce
a range of solutions, it is desirable to produce a wide range of solutions. A possible
way to address this would be to explore the use of a crowding distance metric to
increase the diversity of the set produced by MONNDE. It is also known that there
are limitations to using the linear combination of objectives method used to train the MONNDE algorithm. These include the inability to handle concavities or discontinuities in the Pareto front. To address this, future work would include applying the $\epsilon$-constraint to the MONNDE algorithm which has been shown to address these issues.

### 7.3.6 Univariate Forecasting of Ireland’s Energy Sector

This research does not consider any meteorological data or time of day when making predictions of Ireland’s wind power generation or power demand. This would likely significantly increase the prediction accuracy of the evolved networks as knowledge of future wind speeds can inform prediction of future availability of wind energy. Similarly the power demand profile has a strong temporal regularity pattern based on the time of day, this information would beyond doubt improve the accuracy of forecasts. This is differed for future research as the purpose of Chapter 5 was to simply demonstrate the effectiveness of evolutionary neural networks for these forecasting problems. A caveat to this that would partially justify the univariate forecasting implemented is that when predicting wind power generation during storm Ophelia, our univariate model would outperform models that use wind speed data as wind turbines are shut down in high wind to avoid damage. In the normal case however, the univariate approach implemented here is a limitation of this thesis.

### 7.3.7 Implementing Resource Predictions in Data Center Management

It is not clear what way incorporating the host CPU utilization predictions into the management of data centers would affect the CPU utilization profile. For example, if a virtual machine is moved based on the prediction that the current host machine will become over utilized in the future, this will result in the host machine having one less virtual machine demanding resources in the future. This will mean that the future CPU utilization will be less than predicted. The research in this thesis does not predict disk or RAM utilization which are also important in data center operations.
7.4 Future Work

7.4.1 PSO Parameters

The results in Chapter 2 indicate that PSO parameters that are not commonly used often lead to the best performance. A more in depth theoretical analysis of the effect that these parameters have on the algorithm is needed. Also the Adaptive Velocity PSO performs the best on many of the benchmark equations but not on the watershed management neural network training problem. Further investigation is needed to explain why this is the case.

7.4.2 Scaling Up NDE

As already stated, the current form of the NDE algorithm is best suited to shallow networks. Future work would combine NDE with CPPNs to generate larger networks.

7.4.3 Applications of MONNDE

MONNDE has proved to perform competitively with the state of the art multi-objective optimisation algorithms while providing the added benefits of producing an approximate model that can be used for future changes to the environment. It is hoped that MONNDE will be applied to more dynamic multi-objective optimisation problems so that researchers in different fields can benefit from it, e.g. in multi-objective stock portfolio optimisation and cloud computing. It is also hoped to scale MONNDE up to problems requiring deep networks.

7.4.4 Building on Energy Predictions for Ireland

It has been demonstrated that evolutionary methods such as CMA-ES can effectively train networks for forecasting in the energy sector. This research can be extended in many ways. One such extension would be to evolve networks for multivariate forecasting as previously mentioned. Another extension would be to combine this research with the power generator scheduling problem. Future power demands effect how power generators are scheduled. In systems that use thermal power generators and power from wind turbines, predictions of wind power generation are also crucial.
7.5. Final Remarks

for unit commitment. Future research would apply these predictions for model predictive control of such systems. This research also explored the forecasting of carbon dioxide. It would also be the aim of future research to apply these evolved forecasting networks to other air born pollutants such as nitrogen oxide from diesel engines.

7.4.5 Cloud Predictions and Data Center Management

As has already been alluded to, future research would involve making predictions for disk and RAM usage and also to use these predictions for more efficient host migration. Data centers use vast amounts of energy. It would therefore be hoped that more efficient virtual machine migration using model predictive control could reduce this energy consumption. Since service level agreements must also be considered when migrating virtual machines, this migration process could be framed as a dynamic multi-objective optimisation problem. It is therefore hoped that the MONNDE algorithm could be utilized to produce the range of solutions that optimise energy consumption and service level agreements to varying degrees.

7.5 Final Remarks

Whether its the geometry of the honeycomb structure or Fibonacci sequence observed in plants, there are many examples of where nature has already figured out the most efficient solution to problems found in the natural world. Evolutionary neural networks take inspiration from two of the most fascinating phenomena in nature: Darwinian Evolution and the biological brain. It is hoped that this thesis has demonstrated that this nature inspired approach to function approximation can be an effective tool for solving many problems faced in the energy and environmental sectors.
Chapter 8

Appendix

1. Appendix A - Watershed Management Data
Table 1: Training States

<table>
<thead>
<tr>
<th>State</th>
<th>$Q_1 (L^3)$</th>
<th>$Q_2 (L^3)$</th>
<th>$S(L^3)$</th>
<th>State</th>
<th>$Q_1 (L^3)$</th>
<th>$Q_2 (L^3)$</th>
<th>$S(L^3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>92</td>
<td>39</td>
<td>11</td>
<td>51</td>
<td>81</td>
<td>35</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>129</td>
<td>53</td>
<td>13</td>
<td>52</td>
<td>84</td>
<td>37</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>129</td>
<td>53</td>
<td>13</td>
<td>53</td>
<td>154</td>
<td>63</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>143</td>
<td>59</td>
<td>14</td>
<td>54</td>
<td>96</td>
<td>42</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>84</td>
<td>36</td>
<td>10</td>
<td>55</td>
<td>129</td>
<td>53</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>116</td>
<td>48</td>
<td>12</td>
<td>56</td>
<td>113</td>
<td>47</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>86</td>
<td>37</td>
<td>10</td>
<td>57</td>
<td>125</td>
<td>52</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>141</td>
<td>58</td>
<td>14</td>
<td>58</td>
<td>105</td>
<td>44</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>117</td>
<td>49</td>
<td>12</td>
<td>59</td>
<td>135</td>
<td>56</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>137</td>
<td>56</td>
<td>14</td>
<td>60</td>
<td>134</td>
<td>55</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>114</td>
<td>48</td>
<td>12</td>
<td>61</td>
<td>86</td>
<td>37</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>85</td>
<td>37</td>
<td>10</td>
<td>62</td>
<td>146</td>
<td>60</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>141</td>
<td>58</td>
<td>14</td>
<td>63</td>
<td>114</td>
<td>48</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>139</td>
<td>57</td>
<td>14</td>
<td>64</td>
<td>159</td>
<td>65</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>94</td>
<td>40</td>
<td>11</td>
<td>65</td>
<td>102</td>
<td>43</td>
<td>11</td>
</tr>
<tr>
<td>16</td>
<td>113</td>
<td>47</td>
<td>12</td>
<td>66</td>
<td>124</td>
<td>51</td>
<td>13</td>
</tr>
<tr>
<td>17</td>
<td>154</td>
<td>63</td>
<td>15</td>
<td>67</td>
<td>97</td>
<td>41</td>
<td>11</td>
</tr>
<tr>
<td>18</td>
<td>152</td>
<td>62</td>
<td>14</td>
<td>68</td>
<td>106</td>
<td>45</td>
<td>12</td>
</tr>
<tr>
<td>19</td>
<td>87</td>
<td>37</td>
<td>10</td>
<td>69</td>
<td>90</td>
<td>39</td>
<td>11</td>
</tr>
<tr>
<td>20</td>
<td>88</td>
<td>38</td>
<td>10</td>
<td>70</td>
<td>107</td>
<td>45</td>
<td>12</td>
</tr>
<tr>
<td>21</td>
<td>107</td>
<td>45</td>
<td>12</td>
<td>71</td>
<td>136</td>
<td>56</td>
<td>14</td>
</tr>
<tr>
<td>22</td>
<td>127</td>
<td>53</td>
<td>13</td>
<td>72</td>
<td>83</td>
<td>36</td>
<td>10</td>
</tr>
<tr>
<td>23</td>
<td>111</td>
<td>47</td>
<td>12</td>
<td>73</td>
<td>85</td>
<td>37</td>
<td>10</td>
</tr>
<tr>
<td>24</td>
<td>140</td>
<td>57</td>
<td>14</td>
<td>74</td>
<td>125</td>
<td>52</td>
<td>13</td>
</tr>
<tr>
<td>25</td>
<td>140</td>
<td>58</td>
<td>14</td>
<td>75</td>
<td>125</td>
<td>52</td>
<td>13</td>
</tr>
<tr>
<td>26</td>
<td>82</td>
<td>36</td>
<td>10</td>
<td>76</td>
<td>110</td>
<td>46</td>
<td>12</td>
</tr>
<tr>
<td>27</td>
<td>122</td>
<td>51</td>
<td>13</td>
<td>77</td>
<td>153</td>
<td>62</td>
<td>15</td>
</tr>
<tr>
<td>28</td>
<td>133</td>
<td>55</td>
<td>13</td>
<td>78</td>
<td>138</td>
<td>57</td>
<td>14</td>
</tr>
<tr>
<td>29</td>
<td>102</td>
<td>43</td>
<td>11</td>
<td>79</td>
<td>103</td>
<td>44</td>
<td>11</td>
</tr>
<tr>
<td>30</td>
<td>134</td>
<td>55</td>
<td>13</td>
<td>80</td>
<td>138</td>
<td>57</td>
<td>14</td>
</tr>
<tr>
<td>31</td>
<td>132</td>
<td>54</td>
<td>13</td>
<td>81</td>
<td>145</td>
<td>59</td>
<td>14</td>
</tr>
<tr>
<td>32</td>
<td>110</td>
<td>46</td>
<td>12</td>
<td>82</td>
<td>144</td>
<td>59</td>
<td>14</td>
</tr>
<tr>
<td>33</td>
<td>122</td>
<td>51</td>
<td>13</td>
<td>83</td>
<td>144</td>
<td>59</td>
<td>14</td>
</tr>
<tr>
<td>34</td>
<td>115</td>
<td>48</td>
<td>12</td>
<td>84</td>
<td>100</td>
<td>43</td>
<td>11</td>
</tr>
<tr>
<td>35</td>
<td>140</td>
<td>57</td>
<td>14</td>
<td>85</td>
<td>86</td>
<td>37</td>
<td>10</td>
</tr>
<tr>
<td>36</td>
<td>129</td>
<td>54</td>
<td>13</td>
<td>86</td>
<td>101</td>
<td>43</td>
<td>11</td>
</tr>
<tr>
<td>37</td>
<td>97</td>
<td>41</td>
<td>11</td>
<td>87</td>
<td>154</td>
<td>63</td>
<td>15</td>
</tr>
<tr>
<td>38</td>
<td>116</td>
<td>49</td>
<td>12</td>
<td>88</td>
<td>88</td>
<td>38</td>
<td>10</td>
</tr>
<tr>
<td>39</td>
<td>152</td>
<td>62</td>
<td>15</td>
<td>89</td>
<td>156</td>
<td>64</td>
<td>15</td>
</tr>
<tr>
<td>40</td>
<td>159</td>
<td>65</td>
<td>15</td>
<td>90</td>
<td>132</td>
<td>54</td>
<td>13</td>
</tr>
<tr>
<td>41</td>
<td>116</td>
<td>48</td>
<td>12</td>
<td>91</td>
<td>137</td>
<td>56</td>
<td>14</td>
</tr>
<tr>
<td>42</td>
<td>115</td>
<td>48</td>
<td>12</td>
<td>92</td>
<td>91</td>
<td>39</td>
<td>11</td>
</tr>
<tr>
<td>43</td>
<td>126</td>
<td>52</td>
<td>13</td>
<td>93</td>
<td>110</td>
<td>46</td>
<td>12</td>
</tr>
<tr>
<td>44</td>
<td>86</td>
<td>37</td>
<td>10</td>
<td>94</td>
<td>108</td>
<td>45</td>
<td>12</td>
</tr>
<tr>
<td>45</td>
<td>87</td>
<td>38</td>
<td>10</td>
<td>95</td>
<td>150</td>
<td>61</td>
<td>14</td>
</tr>
<tr>
<td>46</td>
<td>159</td>
<td>64</td>
<td>15</td>
<td>96</td>
<td>118</td>
<td>49</td>
<td>12</td>
</tr>
<tr>
<td>47</td>
<td>95</td>
<td>41</td>
<td>11</td>
<td>97</td>
<td>151</td>
<td>62</td>
<td>14</td>
</tr>
<tr>
<td>48</td>
<td>124</td>
<td>51</td>
<td>13</td>
<td>98</td>
<td>123</td>
<td>51</td>
<td>13</td>
</tr>
<tr>
<td>49</td>
<td>85</td>
<td>37</td>
<td>10</td>
<td>99</td>
<td>101</td>
<td>43</td>
<td>11</td>
</tr>
<tr>
<td>50</td>
<td>107</td>
<td>45</td>
<td>12</td>
<td>100</td>
<td>115</td>
<td>48</td>
<td>12</td>
</tr>
</tbody>
</table>
Table 2: Test States

<table>
<thead>
<tr>
<th>State</th>
<th>$Q_1(L^3)$</th>
<th>$Q_2(L^3)$</th>
<th>$S(L^3)$</th>
<th>State</th>
<th>$Q_1(L^3)$</th>
<th>$Q_2(L^3)$</th>
<th>$S(L^3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>125</td>
<td>52</td>
<td>13</td>
<td>26</td>
<td>146</td>
<td>60</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>151</td>
<td>62</td>
<td>14</td>
<td>27</td>
<td>104</td>
<td>44</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>116</td>
<td>49</td>
<td>12</td>
<td>28</td>
<td>126</td>
<td>52</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>160</td>
<td>65</td>
<td>15</td>
<td>29</td>
<td>148</td>
<td>61</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>148</td>
<td>60</td>
<td>14</td>
<td>30</td>
<td>83</td>
<td>36</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>153</td>
<td>62</td>
<td>15</td>
<td>31</td>
<td>82</td>
<td>36</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>148</td>
<td>60</td>
<td>14</td>
<td>32</td>
<td>121</td>
<td>50</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>149</td>
<td>61</td>
<td>14</td>
<td>33</td>
<td>145</td>
<td>59</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>122</td>
<td>51</td>
<td>13</td>
<td>34</td>
<td>145</td>
<td>60</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>86</td>
<td>37</td>
<td>10</td>
<td>35</td>
<td>89</td>
<td>37</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>84</td>
<td>36</td>
<td>10</td>
<td>36</td>
<td>98</td>
<td>42</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>112</td>
<td>47</td>
<td>12</td>
<td>37</td>
<td>133</td>
<td>55</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>106</td>
<td>45</td>
<td>12</td>
<td>38</td>
<td>95</td>
<td>40</td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>90</td>
<td>39</td>
<td>11</td>
<td>39</td>
<td>158</td>
<td>64</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>134</td>
<td>55</td>
<td>13</td>
<td>40</td>
<td>102</td>
<td>43</td>
<td>11</td>
</tr>
<tr>
<td>16</td>
<td>145</td>
<td>59</td>
<td>14</td>
<td>41</td>
<td>100</td>
<td>43</td>
<td>11</td>
</tr>
<tr>
<td>17</td>
<td>87</td>
<td>37</td>
<td>10</td>
<td>42</td>
<td>131</td>
<td>54</td>
<td>13</td>
</tr>
<tr>
<td>18</td>
<td>155</td>
<td>63</td>
<td>15</td>
<td>43</td>
<td>111</td>
<td>48</td>
<td>13</td>
</tr>
<tr>
<td>19</td>
<td>149</td>
<td>61</td>
<td>14</td>
<td>44</td>
<td>109</td>
<td>46</td>
<td>12</td>
</tr>
<tr>
<td>20</td>
<td>126</td>
<td>52</td>
<td>13</td>
<td>45</td>
<td>157</td>
<td>64</td>
<td>15</td>
</tr>
<tr>
<td>21</td>
<td>105</td>
<td>44</td>
<td>12</td>
<td>46</td>
<td>115</td>
<td>48</td>
<td>12</td>
</tr>
<tr>
<td>22</td>
<td>117</td>
<td>49</td>
<td>12</td>
<td>47</td>
<td>80</td>
<td>35</td>
<td>10</td>
</tr>
<tr>
<td>23</td>
<td>83</td>
<td>36</td>
<td>10</td>
<td>48</td>
<td>104</td>
<td>44</td>
<td>11</td>
</tr>
<tr>
<td>24</td>
<td>102</td>
<td>43</td>
<td>11</td>
<td>49</td>
<td>136</td>
<td>56</td>
<td>13</td>
</tr>
<tr>
<td>25</td>
<td>121</td>
<td>50</td>
<td>13</td>
<td>50</td>
<td>126</td>
<td>52</td>
<td>13</td>
</tr>
</tbody>
</table>
## Appendix B - Power Generator Data

### Table 3: 5 Unit Generator Coefficients

<table>
<thead>
<tr>
<th>Unit</th>
<th>$P_{min}$</th>
<th>$P_{max}$</th>
<th>UR</th>
<th>DR</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>α</th>
<th>β</th>
<th>γ</th>
<th>δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>75</td>
<td>30</td>
<td>30</td>
<td>25</td>
<td>2.0</td>
<td>0.00080</td>
<td>100</td>
<td>0.042</td>
<td>80</td>
<td>-0.805</td>
<td>0.0380</td>
<td>0.65750</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>125</td>
<td>30</td>
<td>30</td>
<td>60</td>
<td>1.8</td>
<td>0.00300</td>
<td>140</td>
<td>0.040</td>
<td>50</td>
<td>-0.555</td>
<td>0.0550</td>
<td>0.57773</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>175</td>
<td>40</td>
<td>40</td>
<td>100</td>
<td>2.1</td>
<td>0.00012</td>
<td>160</td>
<td>0.038</td>
<td>60</td>
<td>-1.355</td>
<td>0.0105</td>
<td>0.49682</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>250</td>
<td>50</td>
<td>50</td>
<td>120</td>
<td>2.0</td>
<td>0.00010</td>
<td>180</td>
<td>0.037</td>
<td>45</td>
<td>-0.600</td>
<td>0.0080</td>
<td>0.4840</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>300</td>
<td>50</td>
<td>50</td>
<td>40</td>
<td>1.8</td>
<td>0.00015</td>
<td>200</td>
<td>0.035</td>
<td>30</td>
<td>-0.555</td>
<td>0.0120</td>
<td>0.56353</td>
</tr>
</tbody>
</table>

### Table 4: 5 Unit B Matrix

-0.000049 0.000014 0.000015 0.000015 0.00002
0.000014 0.000045 0.000016 0.000012 0.000018
0.000015 0.000016 0.000039 0.000011 0.000012
0.000015 0.000002 0.000001 0.000002 0.000014
0.00002 0.000018 0.000012 0.000014 0.000035

### Table 5: 10 Unit Generator Coefficients

<table>
<thead>
<tr>
<th>Unit</th>
<th>$P_{min}$</th>
<th>$P_{max}$</th>
<th>UR</th>
<th>DR</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>α</th>
<th>β</th>
<th>γ</th>
<th>δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
<td>470</td>
<td>80</td>
<td>80</td>
<td>786.7988</td>
<td>38.5397</td>
<td>0.1524</td>
<td>450</td>
<td>0.041</td>
<td>103.3908</td>
<td>-2.4444</td>
<td>0.0312</td>
<td>0.5035</td>
</tr>
<tr>
<td>2</td>
<td>135</td>
<td>470</td>
<td>80</td>
<td>80</td>
<td>451.3251</td>
<td>46.1584</td>
<td>0.1058</td>
<td>600</td>
<td>0.036</td>
<td>103.3908</td>
<td>-2.4444</td>
<td>0.0312</td>
<td>0.5035</td>
</tr>
<tr>
<td>3</td>
<td>73</td>
<td>340</td>
<td>80</td>
<td>80</td>
<td>1049.9977</td>
<td>40.3965</td>
<td>0.0280</td>
<td>320</td>
<td>0.028</td>
<td>300.3910</td>
<td>-4.0695</td>
<td>0.0509</td>
<td>0.4968</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>300</td>
<td>50</td>
<td>50</td>
<td>1243.5311</td>
<td>38.3055</td>
<td>0.0374</td>
<td>260</td>
<td>0.052</td>
<td>300.3910</td>
<td>-4.0695</td>
<td>0.0509</td>
<td>0.4968</td>
</tr>
<tr>
<td>5</td>
<td>73</td>
<td>243</td>
<td>50</td>
<td>50</td>
<td>1658.5906</td>
<td>36.3278</td>
<td>0.0211</td>
<td>280</td>
<td>0.063</td>
<td>320.0000</td>
<td>-3.8132</td>
<td>0.0344</td>
<td>0.4972</td>
</tr>
<tr>
<td>6</td>
<td>57</td>
<td>160</td>
<td>50</td>
<td>50</td>
<td>1356.6592</td>
<td>38.2704</td>
<td>0.0179</td>
<td>310</td>
<td>0.048</td>
<td>320.0000</td>
<td>-3.8132</td>
<td>0.0344</td>
<td>0.4972</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>130</td>
<td>30</td>
<td>30</td>
<td>1450.7045</td>
<td>36.5104</td>
<td>0.0121</td>
<td>300</td>
<td>0.086</td>
<td>330.0056</td>
<td>-3.9623</td>
<td>0.0465</td>
<td>0.5163</td>
</tr>
<tr>
<td>8</td>
<td>47</td>
<td>120</td>
<td>30</td>
<td>30</td>
<td>1450.7045</td>
<td>36.5104</td>
<td>0.0121</td>
<td>340</td>
<td>0.082</td>
<td>330.0056</td>
<td>-3.9623</td>
<td>0.0465</td>
<td>0.5163</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>80</td>
<td>30</td>
<td>30</td>
<td>1455.6056</td>
<td>39.5804</td>
<td>0.1090</td>
<td>270</td>
<td>0.098</td>
<td>350.0056</td>
<td>-3.9524</td>
<td>0.0465</td>
<td>0.5475</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>55</td>
<td>30</td>
<td>30</td>
<td>1469.4026</td>
<td>40.5407</td>
<td>0.1295</td>
<td>380</td>
<td>0.094</td>
<td>360.0012</td>
<td>-3.9864</td>
<td>0.0470</td>
<td>0.5475</td>
</tr>
</tbody>
</table>
Table 6: 10 Unit B Matrix

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000049</td>
<td>0.000014</td>
<td>0.000015</td>
<td>0.000016</td>
<td>0.000017</td>
<td>0.000018</td>
</tr>
<tr>
<td>0.000014</td>
<td>0.000045</td>
<td>0.000016</td>
<td>0.000016</td>
<td>0.000017</td>
<td>0.000018</td>
</tr>
<tr>
<td>0.000015</td>
<td>0.000016</td>
<td>0.000039</td>
<td>0.000047</td>
<td>0.000018</td>
<td>0.000018</td>
</tr>
<tr>
<td>0.000015</td>
<td>0.000016</td>
<td>0.000012</td>
<td>0.000014</td>
<td>0.000017</td>
<td>0.000018</td>
</tr>
<tr>
<td>0.000017</td>
<td>0.000015</td>
<td>0.000012</td>
<td>0.000014</td>
<td>0.000015</td>
<td>0.000017</td>
</tr>
<tr>
<td>0.000017</td>
<td>0.000015</td>
<td>0.000015</td>
<td>0.000015</td>
<td>0.000018</td>
<td>0.000019</td>
</tr>
<tr>
<td>0.000019</td>
<td>0.000018</td>
<td>0.000016</td>
<td>0.000016</td>
<td>0.000017</td>
<td>0.000018</td>
</tr>
<tr>
<td>0.000019</td>
<td>0.000018</td>
<td>0.000019</td>
<td>0.000015</td>
<td>0.000016</td>
<td>0.000017</td>
</tr>
</tbody>
</table>

Table 7: 15 Unit Generator Coefficients

| Unit | P^{min} | P^{max} | UR | DR | a | b | c | d | e | α | β | γ | η | δ | \( \gamma \) | \( \eta \) | \( \delta \) | \( \gamma \) | \( \eta \) | \( \delta \) | \( \gamma \) | \( \eta \) | \( \delta \) |
| 1    | 150     | 455     | 80 | 120| 671| 10.1| 0.000299| 100| 0.042| 80.0000| -0.8050| 0.0180| 0.655| 0.02846 |
| 2    | 150     | 455     | 80 | 120| 574| 10.2| 0.000183| 140| 0.040| 50.0000| -0.5550| 0.0150| 0.5773| 0.02446 |
| 3    | 20      | 130     | 130| 130| 374| 8.8 | 0.001126| 160| 0.038| 60.0000| -1.3550| 0.0105| 0.4968| 0.02270 |
| 4    | 20      | 130     | 130| 130| 374| 8.8 | 0.001126| 180| 0.037| 45.0000| -0.6000| 0.0080| 0.4868| 0.01948 |
| 5    | 150     | 470     | 80 | 120| 461| 10.4| 0.000205| 200| 0.035| 30.0000| -0.5550| 0.0120| 0.5035| 0.02975 |
| 6    | 135     | 460     | 80 | 120| 630| 10.1| 0.000301| 450| 0.041| 103.3908| -2.4444| 0.0312| 0.5035| 0.02970 |
| 7    | 135     | 465     | 80 | 120| 548| 9.8 | 0.000364| 600| 0.036| 103.3908| -2.4444| 0.0312| 0.5035| 0.02970 |
| 8    | 60      | 380     | 65 | 160| 227| 11.2| 0.000338| 320| 0.028| 300.3910| -4.0905| 0.0509| 0.4968| 0.02920 |
| 9    | 25      | 162     | 60 | 160| 173| 11.2| 0.000867| 260| 0.052| 300.3910| -4.0905| 0.0509| 0.4968| 0.02920 |
| 10   | 25      | 160     | 60 | 180| 175| 10.7| 0.001203| 280| 0.063| 320.0006| -3.8132| 0.0344| 0.4972| 0.02920 |
| 11   | 20      | 80      | 80 | 80 | 186| 10.2| 0.003586| 310| 0.048| 320.0006| -1.8132| 0.0344| 0.4972| 0.02920 |
| 12   | 20      | 80      | 80 | 80 | 230| 9.9 | 0.005513| 300| 0.086| 330.0056| -1.9023| 0.0465| 0.5163| 0.02140 |
| 13   | 25      | 85      | 80 | 80 | 225| 11.1| 0.003721| 340| 0.082| 330.0056| -1.9023| 0.0465| 0.5461| 0.02140 |
| 14   | 15      | 55      | 55 | 55 | 309| 12.1| 0.001929| 270| 0.098| 350.0056| -1.9224| 0.0465| 0.5475| 0.02340 |
| 15   | 15      | 55      | 55 | 55 | 323| 12.4| 0.004447| 380| 0.094| 360.0012| -1.9864| 0.0470| 0.5475| 0.02340 |
Table 8: 15 Unit B Matrix

<table>
<thead>
<tr>
<th></th>
<th>0.00014</th>
<th>0.00012</th>
<th>0.00007</th>
<th>-0.00001</th>
<th>-0.00003</th>
<th>-0.00001</th>
<th>-0.00001</th>
<th>-0.00003</th>
<th>-0.00001</th>
<th>-0.00003</th>
<th>-0.00002</th>
<th>0.00004</th>
<th>0.00003</th>
<th>-0.00001</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00012</td>
<td>0.00015</td>
<td>0.00013</td>
<td>0.00000</td>
<td>-0.00005</td>
<td>-0.00002</td>
<td>0.00000</td>
<td>0.00001</td>
<td>-0.00002</td>
<td>-0.00004</td>
<td>-0.00004</td>
<td>0.00000</td>
<td>0.00004</td>
<td>0.00010</td>
<td>-0.00002</td>
</tr>
<tr>
<td>0.00007</td>
<td>0.00013</td>
<td>0.00076</td>
<td>-0.00001</td>
<td>-0.00013</td>
<td>-0.00009</td>
<td>-0.00001</td>
<td>0.00000</td>
<td>-0.00008</td>
<td>-0.00012</td>
<td>-0.00017</td>
<td>0.00000</td>
<td>-0.00026</td>
<td>0.00111</td>
<td>-0.00022</td>
</tr>
<tr>
<td>-0.00001</td>
<td>0.00000</td>
<td>-0.00001</td>
<td>0.00034</td>
<td>-0.00007</td>
<td>-0.00004</td>
<td>0.00011</td>
<td>0.00050</td>
<td>0.00032</td>
<td>-0.00011</td>
<td>0.00000</td>
<td>0.00001</td>
<td>-0.00026</td>
<td>0.00001</td>
<td>-0.00026</td>
</tr>
<tr>
<td>-0.00003</td>
<td>-0.00005</td>
<td>-0.00013</td>
<td>-0.00007</td>
<td>0.00090</td>
<td>0.00014</td>
<td>-0.00003</td>
<td>-0.00012</td>
<td>-0.00010</td>
<td>-0.00013</td>
<td>0.00007</td>
<td>-0.00002</td>
<td>-0.00002</td>
<td>-0.00024</td>
<td>-0.00003</td>
</tr>
<tr>
<td>-0.00001</td>
<td>-0.00002</td>
<td>-0.00009</td>
<td>-0.00004</td>
<td>0.00014</td>
<td>0.00016</td>
<td>0.00000</td>
<td>-0.00006</td>
<td>-0.00005</td>
<td>-0.00008</td>
<td>0.00011</td>
<td>-0.00001</td>
<td>-0.00002</td>
<td>-0.00017</td>
<td>0.00003</td>
</tr>
<tr>
<td>-0.00001</td>
<td>0.00000</td>
<td>-0.00001</td>
<td>0.00011</td>
<td>-0.00003</td>
<td>0.00000</td>
<td>0.00015</td>
<td>0.00017</td>
<td>0.00015</td>
<td>0.00009</td>
<td>-0.00005</td>
<td>0.00000</td>
<td>-0.00002</td>
<td>-0.00008</td>
<td>0.00003</td>
</tr>
<tr>
<td>-0.00001</td>
<td>0.00001</td>
<td>0.00000</td>
<td>0.00050</td>
<td>-0.00012</td>
<td>-0.00006</td>
<td>0.00017</td>
<td>0.00168</td>
<td>0.00082</td>
<td>0.00079</td>
<td>-0.00023</td>
<td>-0.00036</td>
<td>0.00003</td>
<td>0.0005</td>
<td>-0.00078</td>
</tr>
<tr>
<td>-0.00003</td>
<td>-0.00002</td>
<td>-0.00008</td>
<td>0.00029</td>
<td>-0.00010</td>
<td>-0.00005</td>
<td>0.00015</td>
<td>0.00082</td>
<td>0.00129</td>
<td>0.00116</td>
<td>-0.00021</td>
<td>-0.00025</td>
<td>0.00007</td>
<td>-0.00012</td>
<td>-0.00072</td>
</tr>
<tr>
<td>-0.00005</td>
<td>-0.00004</td>
<td>-0.00012</td>
<td>0.00032</td>
<td>-0.00013</td>
<td>-0.00008</td>
<td>0.00009</td>
<td>0.00079</td>
<td>0.00116</td>
<td>0.00200</td>
<td>-0.00027</td>
<td>-0.00034</td>
<td>0.00009</td>
<td>-0.00011</td>
<td>-0.00088</td>
</tr>
<tr>
<td>-0.00003</td>
<td>-0.00004</td>
<td>-0.00017</td>
<td>-0.00011</td>
<td>0.00007</td>
<td>0.00011</td>
<td>-0.00005</td>
<td>-0.00023</td>
<td>-0.00021</td>
<td>-0.00027</td>
<td>0.00140</td>
<td>0.00001</td>
<td>0.00004</td>
<td>-0.00038</td>
<td>0.00168</td>
</tr>
<tr>
<td>-0.00002</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>-0.00002</td>
<td>-0.00001</td>
<td>0.00007</td>
<td>-0.00036</td>
<td>-0.00005</td>
<td>-0.00004</td>
<td>0.00001</td>
<td>0.00054</td>
<td>-0.00001</td>
<td>-0.00004</td>
<td>0.00028</td>
</tr>
<tr>
<td>0.00004</td>
<td>0.00004</td>
<td>-0.00026</td>
<td>0.00001</td>
<td>-0.00002</td>
<td>-0.00002</td>
<td>0.00000</td>
<td>0.00001</td>
<td>0.00007</td>
<td>0.00009</td>
<td>0.00004</td>
<td>-0.00001</td>
<td>0.00103</td>
<td>-0.00101</td>
<td>0.00028</td>
</tr>
<tr>
<td>0.00033</td>
<td>0.00010</td>
<td>0.00111</td>
<td>0.00001</td>
<td>-0.00024</td>
<td>-0.00002</td>
<td>0.00005</td>
<td>-0.00012</td>
<td>-0.00011</td>
<td>-0.00038</td>
<td>-0.00004</td>
<td>-0.00101</td>
<td>0.000578</td>
<td>-0.00094</td>
<td></td>
</tr>
<tr>
<td>-0.00001</td>
<td>-0.00002</td>
<td>-0.00028</td>
<td>-0.00026</td>
<td>-0.00003</td>
<td>-0.00008</td>
<td>-0.00078</td>
<td>-0.00072</td>
<td>-0.00088</td>
<td>0.00168</td>
<td>0.00028</td>
<td>0.00028</td>
<td>-0.00094</td>
<td>0.01283</td>
<td></td>
</tr>
</tbody>
</table>
Table 9: 24 Hour Power Demands

<table>
<thead>
<tr>
<th>Hour</th>
<th>Normal Power Demand (MW)</th>
<th>Test Power Demand (MW)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5 Units</td>
<td>10 Units</td>
</tr>
<tr>
<td>1</td>
<td>410</td>
<td>1036</td>
</tr>
<tr>
<td>2</td>
<td>435</td>
<td>1110</td>
</tr>
<tr>
<td>3</td>
<td>475</td>
<td>1258</td>
</tr>
<tr>
<td>4</td>
<td>530</td>
<td>1406</td>
</tr>
<tr>
<td>5</td>
<td>558</td>
<td>1480</td>
</tr>
<tr>
<td>6</td>
<td>608</td>
<td>1628</td>
</tr>
<tr>
<td>7</td>
<td>626</td>
<td>1702</td>
</tr>
<tr>
<td>8</td>
<td>654</td>
<td>1776</td>
</tr>
<tr>
<td>9</td>
<td>690</td>
<td>1924</td>
</tr>
<tr>
<td>10</td>
<td>704</td>
<td>2022</td>
</tr>
<tr>
<td>11</td>
<td>720</td>
<td>2106</td>
</tr>
<tr>
<td>12</td>
<td>740</td>
<td>2150</td>
</tr>
<tr>
<td>13</td>
<td>704</td>
<td>2072</td>
</tr>
<tr>
<td>14</td>
<td>690</td>
<td>1924</td>
</tr>
<tr>
<td>15</td>
<td>654</td>
<td>1776</td>
</tr>
<tr>
<td>16</td>
<td>580</td>
<td>1554</td>
</tr>
<tr>
<td>17</td>
<td>558</td>
<td>1480</td>
</tr>
<tr>
<td>18</td>
<td>608</td>
<td>1628</td>
</tr>
<tr>
<td>19</td>
<td>654</td>
<td>1776</td>
</tr>
<tr>
<td>20</td>
<td>704</td>
<td>1972</td>
</tr>
<tr>
<td>21</td>
<td>680</td>
<td>1924</td>
</tr>
<tr>
<td>22</td>
<td>605</td>
<td>1628</td>
</tr>
<tr>
<td>23</td>
<td>527</td>
<td>1332</td>
</tr>
<tr>
<td>24</td>
<td>463</td>
<td>1184</td>
</tr>
</tbody>
</table>
Figure 1: Wind Power Generation Error Per Time Step. This figure illustrates the wind power generated along with the error of the Covariance Matrix Adaptation - Evolutionary Strategy trained neural network at each time step.

Figure 2: Power Demand Error Per Time Step. This figure illustrates the power demand along with the error of the Covariance Matrix Adaptation - Evolutionary Strategy trained neural network at each time step.
Figure 3: Carbon Dioxide Level Error Per Time Step. This figure illustrates the carbon dioxide level along with the error of the Covariance Matrix Adaptation - Evolutionary Strategy trained neural network at each time step.
Figure 4: Host Utilization Predictions for Time Steps 500 - 1000 of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for time steps 500 - 1000 of training data.

Figure 5: Host Utilization Predictions for Time Steps 1000 - 1500 of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for time steps 1000 - 1500 of training data.
Figure 6: Host Utilization Predictions for Time Steps 1500 - 2000 of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for time steps 1500 - 2000 of training data.

Figure 7: Host Utilization Predictions for Time Steps 2000 - 2300 of Training Data. This figure illustrates the predicted host utilization of each evolved neural network for time steps 2000 - 2300 of training data.
Bibliography


