Problem Decomposition and Adaptation in Cooperative Neuro-evolution

by

Rohitash Chandra

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2012

Abstract

One way to train neural networks is to use evolutionary algorithms such as cooperative coevolution - a method that decomposes the network's learnable parameters into subsets, called subcomponents. Cooperative coevolution gains advantage over other methods by evolving particular subcomponents independently from the rest of the network. Its success depends strongly on how the problem decomposition is carried out.

This thesis suggests new forms of problem decomposition, based on a novel and intuitive choice of modularity, and examines in detail at what stage and to what extent the different decomposition methods should be used. The new methods are evaluated by training feedforward networks to solve pattern classification tasks, and by training recurrent networks to solve grammatical inference problems.

Efficient problem decomposition methods group interacting variables into the same subcomponents. We examine the methods from the literature and provide an analysis of the nature of the neural network optimization problem in terms of interacting variables. We then present a novel problem decomposition method that groups interacting variables and that can be generalized to neural networks with more than a single hidden layer.

We then incorporate local search into cooperative neuro-evolution. We present a memetic cooperative coevolution method that takes into account the cost of employing local search across several sub-populations.

The optimisation process changes during evolution in terms of diversity and interacting variables. To address this, we examine the adaptation of the problem decomposition method during the evolutionary process.

The results in this thesis show that the proposed methods improve performance in terms of optimization time, scalability and robustness.

As a further test, we apply the problem decomposition and adaptive cooperative coevolution methods for training recurrent neural networks on chaotic time series problems. The proposed methods show better performance in terms of accuracy and robustness.

Dedication

I dedicate this thesis to my mom, Prakash Wati. She has been the main inspiration behind this thesis.

Acknowledgements

I wish to acknowledge my supervisors, Dr. Marcus Frean and Prof. Mengjie Zhang for their guidance during the course of the PhD program.

I would like to thank Prof. Christian Omlin of Middle East Technical University and Mohammad Nabi Omidvar of RMIT University. I acknowledge the comments from the anonymous reviewers on the papers published from this thesis. I also acknowledge the examination committee.

My sincere gratitude to Siva Kumar Dorairaj and Rajeswari Nadarajan for their support and motivation. I also thank Prema Ram and Mani Nambayah for their support.

I also thank my sisters, Kumari Ranjeeni and Ranjita Kumari for their support and my wife Ronika Kumar for the inspiration. My sincere gratitude to all my family members and friends for their support.

Publications

Journal Articles

1. R. Chandra, M. Frean, M. Zhang, Crossover-based Local Search in Cooperative Co-evolutionary Feedforward Networks, Applied Soft Computing, Elsevier, Conditionally Accepted, March 2012 .

2. R. Chandra, M. Frean , M. Zhang, On the Issue of Separability in Cooperative Co-evolutionary Feedforward Networks , Neurocomputing, Elsevier, In press, 2012.
(http://dx.doi.org/10.1016/j.neucom.2012.02.005)

3. R. Chandra, M. Zhang, Cooperative Coevolution of Elman Recurrent Neural Networks for Chaotic Time Series Prediction, Neurocomputing, Elsevier, In Press, 2012.
(http://dx.doi.org/10.1016/j.neucom.2012.01.014)

4. R. Chandra , M. Frean, M. Zhang, Adapting Modularity During Learning in Cooperative Co-evolutionary Recurrent Neural Networks, Soft Computing, In Press, 2012 (DOI: 10.1007/s00500-011-0798-9).

5. R. Chandra M. Frean, M. Zhang and Christian Omlin, Encoding Subcomponents in Cooperative Coevolutionary Recurrent Neural Networks , Neurocomputing, Elsevier, Vol. 74, 2011, pp. 3223-3234.

Refereed Conference Proceedings

1. R. Chandra, M. Frean, M. Zhang, Modularity Adaptation in Cooperative Co-evolutionary Feedforward Neural Networks , Proceedings of the International Joint Conference on Neural Networks, San Jose, USA, 2011, pp. 681-688. .

2. R. Chandra, M. Frean, M. Zhang, A Memetic Framework for Cooperative Coevolution of Recurrent Neural Networks , Proceedings of the International Joint Conference on Neural Networks, San Jose, USA, 2011, pp. 673-680.

3. R. Chandra, M. Frean, M. Zhang, An Encoding Scheme for Cooperative Co-evolutionary Neural Networks. Proceedings of the 23rd Australasian Joint Conference on Artificial Intelligence. AI 2010: Advances in Artificial Intelligence. Lecture Notes in Artificial Intelligence. Vol. 6464. Springer. Adelaide, Australia, 2010. pp. 253-262.

4. R. Chandra, M. Frean, L. Rolland, A Hybrid Meta-Heuristic Paradigm for Solving the Forward Kinematics of 6-6 General Parallel Manipulator , Proceedings of 8th IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA 2009), Daejeon, Korea, 2009, pp. 171-176.

5. R. Chandra, M. Zhang, L. Rolland, Solving the Forward Kinematics of the 3RPR Planar Parallel Manipulator using a Hybrid Meta-Heuristic Paradigm , Proceedings of 8th IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA 2009), Daejeon, Korea, 2009, pp. 177-182.

Other Related Publications

1. R. Chandra and L. Rolland, "On Solving the Forward Kinematics of 3RPR Planar Parallel Manipulator using Hybrid Metaheuristics",

Applied Mathematics and Computation, Elsevier, Vol 217, No. 22, 2011 pp. 8997-9008 , (doi: 10.1016/j.amc.2011.03.106)

2. L. Rolland and R. Chandra, " On Solving the Forward Kinematics of the 6-6 General Parallel Manipulator with an Efficient Evolutionary Algorithm", In ROMANSY 18 - Robot Design, Dynamics and Control, Series: CISM International Centre for Mechanical Sciences, Vol. 524 Springer, Berlin, 2010, pp. 117-124.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Premises

Neural networks are nature-inspired computational methods that are loosely modelled after biological neural systems [140]. Neural networks are characterized into *feedforward* and *recurrent* architectures. In contrast to feedforward networks, recurrent neural networks are dynamical systems whose next state and output(s) depend on the present network state and input(s). This feature has made them successful in applications to speech recognition, time series prediction, language learning and control [178, 191]. Gradient descent has been used for training neural networks with variation in the form of *backpropagation* for feedforward networks and *backpropagation-through-time* [226] for recurrent networks.

Evolutionary algorithms are a family of optimisation methods that have shown promising results for NP [1] and other optimisation problems [142, 18]. A major contribution of evolutionary algorithms has been in the training and design of neural networks in order to achieve better performance when compared to traditional approaches [2]. Evolutionary algorithms address the convergence problems associated with algorithms such as gradient descent. The field of using evolutionary computation for evolving

[1] Nondeterministic polynomial time

neural networks is known as *neuro-evolution*.

Cooperative coevolution [169] is a nature-inspired evolutionary computation framework that divides a problem into subcomponents, where each subcomponent employs a separate evolutionary algorithm. The evolutionary algorithms that optimise the subcomponents are genetically isolated and cooperation takes place during fitness evaluation. The use of cooperative coevolution in training neural networks is referred to as *cooperative neuro-evolution* in this thesis. The major advantage of cooperative neuro-evolution is the diversity that it promotes through the subpopulations [168]. It also provides a mechanism for breaking down the neural network architecture and encoding it into different subcomponents. In this way, information associated with the respective groups of weights in the network can be preserved during learning. This information is often lost in neuro-evolution with conventional evolutionary algorithms due to the reproduction operators such as *crossover*. Problem decomposition has been a major issue in cooperative neuro-evolution of both feedforward [62, 77, 27] and recurrent networks [75, 77]. A problem decomposition is also referred as an *encoding scheme*, as it determines how the neural network is broken down and encoded into sub-populations.

In cooperative neuro-evolution of recurrent neural networks, *synapse* and *neuron* level problem decomposition have been used. In the method of *enforced sub-populations* (ESP) [76, 78] a subcomponent consists of the input, output and recurrent connections to a hidden neuron. By contrast, the synapse level decomposes the neural network into the lowest level where a subcomponent is a synapse (weight connection). Synapse level decomposition has shown better performance than ESP and other neuro-evolutionary methods for pole balancing problems [77].

In the original cooperative coevolution approach, the problem is decomposed by having a separate subcomponent for each variable [169]. It was later found that the strategy was only effective for problems that are termed *separable* [127]. In separable problems, there is no interdependency

between the variables whereas in non-separable problems, interdependencies exist. A more precise definition of separability will be given in Chapter 2. Cooperative coevolution naturally appeals to separable problems as there is no interaction among the subcomponents during evolution in its orginal framework [169]. The decomposition of the problem influences the performance of the optimisation algorithm being employed to solve it. In most problems, groups of interacting variables exist. In this thesis, the *degree of non-separability* is a term used to determine the level of interdependencies among variables.

Memetic algorithms [149] typically combine population-based evolutionary algorithms with local search in order to improve the search process. The search for more efficient local search techniques has been a major focus of research in memetic algorithms. It has been shown that evolutionary algorithms can be used as effective local search techniques [109, 131]. In crossover-based local search [131, 147], efficient crossover operators that have local search properties are used for local refinement with a population of a few individuals. They have shown promising results in comparison with other evolutionary approaches for function optimization problems with high dimensions [147].

Feedforward neural networks have been classically trained using backpropogation [181], which has limitations due to convergence problems [36]. In recent years, neuro-evolutionary methods have gained attention for training feedforward and recurrent network architectures on pattern classification, time series and control problems [2, 190, 53, 204] . Backpropagation also falls short in training feedforward networks for control problems such as the pole balancing problem where neuro-evolution has been successfully deployed [204]. Backpropagation through-time [226] has problems in training recurrent neural networks due to the vanishing gradient problem [92], and evolutionary methods can play a significant role in training them in practice [76, 77].

Time series prediction involves the study of the present and past be-

haviour of the system for a prediction of the future. In chaotic systems [128, 205], the initial conditions can make large differences in the behaviour of the system. This is known as the *butterfly effect* and not surprisingly, it makes long-term prediction difficult. The prediction of chaotic time series has a wide range of applications such as in finance [38], signal processing [110], power load [108], weather forecast [129], and sunspot prediction [117, 182, 67]. Neural networks and related computational intelligence methods have been successfully used to solve chaotic time series problems [4, 67, 25, 67, 179, 67, 144, 3]. Chaotic time series will be used as a practical example to examine the proposed methods in this thesis.

1.2 Motivations

1. **Neuro-evolution:** It has been noted that backpropagation has limitations in training feedforward and recurrent networks. Backpropagation has problems in convergence and learning long-term dependencies using recurrent networks. This forms the main motivation for using neuro-evolutionary methods, in particular, cooperative coevolution, which has the feature of decomposing the given neural network architecture into smaller problems and coevolving them. Neuro-evolution provides a learning mechanism that is independent of the network architecture, i.e. an evolutionary algorithm can be deployed easily when given any network topology as no gradient information is required in the optimisation process. This is of advantage for training neural networks in control problems and recurrent networks in learning long-term dependencies. The development of efficient cooperative coevolution methods can enhance neuro-evolution in general.

2. **Efficient algorithms in the sub-populations:** Cooperative coevolution frameworks have used fairly old evolutionary algorithms in

their sub-populations in the past. It has been argued by Yang et. al [233], that there is a need to use more recent and efficient evolutionary algorithms in the sub-populations. The evolutionary algorithm in sub-populations for neuro-evolution needs adaptive properties and has to be applicable in training neural networks. The G3-PCX evolutionary algorithm [42] has shown good performance for real parameter optimization and for training feedforward networks [23]. Therefore, the G3-PCX evolutionary algorithm is used in the sub-populations of cooperative coevolution in this thesis.

3. **Problem decomposition:** Encodings at the synapse [77] and neuron level [76, 78] are the major problem decomposition methods in cooperative neuro-evolution of recurrent networks. The use of synapse level problem decomposition for training neural networks for pattern classification has not been explored. It is important to understand the nature of the neural network training problem in terms of the interacting variables in order to implement efficient problem decomposition methods. The problem decomposition method must take into account the architecture of the neural network in order to group interacting variables into separate subcomponents that can allow better performance in cooperative neuro-evolution.

4. **Local refinement in cooperative neuro-evolution:** Local search in memetic algorithms has shown good performance in function optimization [149, 147], however, local search has not been well studied and used for cooperative coevolution. Local search can be beneficial during learning. A study on the balance of diversification using cooperative coevolution and intensification using local search is needed for cooperative neuro-evolution. The success of local search in memetic algorithms gives the motivation for incorporating local search in cooperative coevolution. The main concerns in memetic algorithms are in employing efficient local search methods and in

balancing diversification and intensification during evolution. It is important to investigate how often to apply local search *(local search frequency)* and for how long to apply them *(local search intensity)*.

5. **Modularity adaptation in cooperative neuro-evolution:** Much work has been done in the use of cooperative coevolution in large scale function optimization and the concentration has been on non-separable problems for large-scale parameter optimization [221, 233]. In order to take advantage of cooperative coevolution, it is important to group interacting variables into separate subcomponents [221]. Although cooperative coevolution has been used for training feedforward [168, 64] and recurrent neural networks [76, 187], the attention has not been on the issue of separability and interacting variables. Each problem decomposition method groups interacting variables and in effect makes assumptions about the degree of non-separability *regardless of the problem*. Instead, it is reasonable to adapt the problem decomposition method (modularity) to different levels *as needed* at different stages of evolution. There has not been any previous study on the adaptation of modularity in cooperative neuro-evolution.

6. **Applications:** In the literature, there has not been any study on the evaluation of the different problems decomposition methods for training feedforward networks on pattern classification and recurrent networks grammatical inference problems. This gives the motivation of using the existing problem deposition methods in the literature to evaluate their strengths and weaknesses. Cooperative neuro-evolution of recurrent networks has not been tested for grammatical inference problems. The application of cooperative neuro-evolution in learning grammars generated by *finite state machines* can establish a foundation for their applicability in training recurrent networks to a wider range of real-world application problems. Chaotic time series prediction is a challenging field and the use of neural networks for mod-

elling them has shown promising results in the past.

1.3 Research Goals

The major goal of this thesis is to explore and improve problem decomposition and adaptation in cooperative neuro-evolution of feedforward and recurrent neural networks. The contribution is explored through the validation of the following research objectives.

1. Evaluate the performance of the existing problem decomposition methods and develop a new method based on the architectural properties of the neural network.
    - Apply new and existing problem decomposition methods for training feedforward networks for pattern classification problems, and recurrent neural networks on grammatical inference problems.
    - Evaluate the performance of the new method on different neural network topologies, this reflect on scalability and robustness.
2. Develop a memetic cooperative coevolution framework that incorporates local search for the neuro-evolution of feedforward and recurrent networks.
    - Employ crossover-based local search in the memetic cooperative coevolution framework, for both feedforward and recurrent neural networks.
    - Study the effect of diversification (from cooperation coevolution) and intensification (from local search) in the memetic cooperative coevolution framework.
3. Develop an algorithm for adapting the modularity (problem decomposition) during evolution. Compare the performance of the adap-

tive modularity problem decomposition method against standard decomposition methods:

- Apply the adaptive modularity method to learning classification problems using feedforward networks.
- Apply the adaptive modularity method to training recurrent networks on grammatical inference problems.

4. Apply adaptive modularity cooperative coevolution and other problem decomposition methods to training recurrent neural networks on chaotic time series problems.
   - Train recurrent neural networks on Lorenz, Mackey Glass and Sunspot time series.
   - Compare the performance of the proposed cooperative coevolution methods with computational intelligence methods from the literature.

1.4 Major Contributions

The major contributions of the thesis can be summarised as follows.

1. The development of a new problem decomposition method that efficiently groups interacting variables into separate subcomponents for training feedforward and recurrent networks. The new method performs better than existing problem decomposition methods on feedforward networks for pattern classification problems and recurrent networks for grammatical inference problems.
2. The understanding of the nature of the neural network training problem in terms of the degree of non-separability. This thesis has empirically shown that neural network training is a partially separable problem and the degree of non-separability (interaction between synapses) changes as the problem is being learnt.

3. The development of a new memetic cooperative coevolution method that efficiently incorporates local search during evolution. The relationship between diversification and intensification has been evaluated, and better performance has been achieved by incorporating local search into cooperative coevolution.
4. The development of the cooperative coevolution method that adapts modularity during evolution. The adaptation of modularity also helps in understanding the nature of the neural network problem in terms of the degree of non-separability. The proposed method achieves better performance than existing methods in terms of training time, scalability and robustness.
5. The application of cooperative neuro-evolution methods for training recurrent neural networks on chaotic time series is a new development in this field. The results show improved performance over several existing computational intelligence techniques.

1.5 Thesis Outline

The thesis is presented as follows.

- **Chapter 1** outlines the motivations, research questions, research goals, contribution and organisation of the thesis.

- **Chapter 2** introduces the fundamentals of feedforward and recurrent neural networks, training algorithms such as backpropagation, backpropagation-through-time and neuro-evolution. Evolutionary computation methods such as genetic algorithms, evolutionary operators, memetic algorithms and cooperative coevolution are also discussed with an emphasis on optimization for non-separable problems. The application of cooperative coevolution to neuro-evolution and existing problem decomposition methods are given in detail.

- **Chapter 3** investigates on the performance of the existing problem decomposition methods. A new problem decomposition method is proposed for feedforward and recurrent networks. The method is tested on pattern classification and grammatical inference problems and compared with existing approaches.

- **Chapter 4** presents the incorporation of local search in cooperative coevolution. The relationship between the diversification provided by cooperative coevolution and intensification provided by local search is explored with applications in training feedforward and recurrent networks.

- **Chapter 5** presents the details of the modularity adaptation method with applications for training feedforward and recurrent networks.

- **Chapter 6** presents an application of cooperative coevolution methods to chaotic time series prediction. The proposed problem decomposition with adaptive modularity in cooperative coevolution is used for training recurrent neural networks on chaotic time series problems. The comparison with the performance of computational intelligence methods from literature is also done.

- **Chapter 7** presents the conclusions from the results and analyses in Chapters 3 - 6 with discussion of future work.

Chapter 2

Background and Literature Review

The chapter begins with a background of feedforward and recurrent neural networks, learning algorithms and grammatical inference problems. The background on major evolutionary computation methods with a review of literature is then given with emphasis on memetic algorithms, cooperative coevolution and neuro-evolution.

2.1 Machine Learning and Optimisation

Machine learning is the processes of learning from a data set using a computer program [145]. Machine learning methods employ models such as neural networks. The process of learning involves adjusting some variables so that the data can be explained by the model. The process of learning can also be seen an optimisation problem [14].

The data set consists of a collection of examples or *instances*. The data set is usually divided into a *training set* and a *test set*. The training data is used for the learning process and the test data is used to evaluate how well the algorithm has learned. In some cases, the data set is split into a third set called the *validation set*. This is used to monitor the training process in

order to avoid *over-fitting* which affects the generalisation performance of the model [122].

Machine learning algorithms are used to learn from the data for problem which involve *classification*, *prediction*, *clustering* and *control*. *Supervised learning*, *unsupervised learning* and *hybrid learning* are the three main learning paradigms [102]. *Learning* is the ability to recognise complex patterns in data to make intellient decisions when presented with unseen data that is not present in the learning process. In supervised learning, the training data consists of input vectors and corresponding outputs. In unsupervised learning, there is no output in the training data for the corresponding input vector. The goal of unspervised learning is to find hidden structures in unlabelled data. Unsupervised learning mostly deals with problems such as clustering. Hybrid learning combines elements of both supervised and unsupervised learning.

This thesis deals with supervised learning where neural networks are used as the model and evolutionary computation methods, such as cooperative coevolution are used for learning. The problems used in this thesis are drawn from classification and prediction domains.

2.2 Neural Networks

Neural networks (NN) are nature inspired computational methods that are loosely modelled after biological neural systems [140]. A neural network consists of a group of interconnected units called *neurons* that have adaptive properties that provide learning. Neural networks are used to model the relationship between input and output values in data and also to find patterns in data [91]. They are capable of performing tasks that include pattern classification, function approximation, prediction or forecasting, clustering or categorization, optimisation, and control [166].

A *neuron* is a single processing unit that computes the weighted sum of its inputs. The interconnections between neurons are called *synapses* or

weights. Neural networks learn by training on data using an algorithm that modifies the interconnection weights as directed by a learning objective for a particular application. The knowledge learnt is distributed over a set of trained networks weights. Neural networks are characterized into *feedforward* and *recurrent* architectures and others [86].

Neural networks are typically arranged into an input, an output and a number of hidden layers. Feedforward networks are also known as *multilayer perceptrons*. Figure 2.1 shows examples of (a) a feedforward neural network and a (b) recurrent neural network that contains feedback connections.

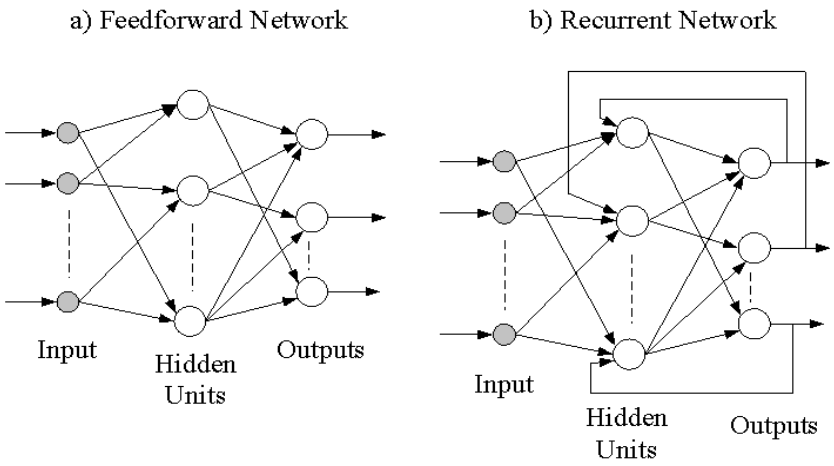


Figure 2.1: An outline of the architectural difference between (a) feedforward and (b) recurrent neural networks

2.2.1 Feedforward Neural Networks

Feedforward networks contain an input layer, one or more hidden layers and an output layer [181]. Each layer contains one or more neurons which propagate activation from one layer to the next by computing a transfer function of their weighted sum of inputs. Figure 2.1 (a) shows the archi-

tecture of a feedforward network.

The standard model of the neuron comprises a set of input connections, a linear combiner and a transfer or activation function. The weight w_{ij} is defined from the input signal x_j to neuron i. The linear combiner computes the *weighted sum* of input signals and adds a *bias*. The dynamics of a feedforward network is described by Equation (2.1), where the total net input activation value y_i of neuron i is given for N input connections.

$$y_i = \sum_{j=1}^{N} (w_{ij} x_j) + \theta_i \tag{2.1}$$

where θ_i represents the bias. The transfer function $f(y_i)$ computes the output o_i of the unit. Some common transfer functions are threshold, sigmoid, hyperbolic tangent and piece-wise linear functions [91]. The sigmoid and hyperbolic tangent transfer function are given in Equation (2.2) and Equation (2.3), which will be used in this thesis.

$$f(y_i) = \frac{1}{1 + e^{-y_i}} \tag{2.2}$$

$$f(y_i) = \frac{e^{2y_i} - 1}{e^{2y_i} + 1} \tag{2.3}$$

The inputs x_j and weights w_{ij} with $j = 1, \ldots, N$ can be represented as the vectors $\mathbf{x} = [x_0, x_1, \ldots, x_N]^t$ and $\mathbf{w}_i = [w_{i0}, w_{i1}, \ldots, w_{iN}]^t$, respectively[1]. The activation value for the ith neuron can be written as $y_i = \mathbf{w}_i \cdot \mathbf{x}$ and thus its output is $o_i = f(\mathbf{w}_i \cdot \mathbf{x})$.

2.2.2 Learning in Feedforward Networks

The goal of learning is to find the set of weights of the neural network on the given training data in order to achieve maximum performance on unseen data. This is done by adjusting the weights in the network according

[1] Vectors and matrices will be indicated in bold script.

to a learning rule until a certain criterion is met which is usually expressed in terms of the *network output error* or *cost function*.

Rumulhart *et al.* [181] introduced the *backpropagation* algorithm which employs gradient descent for training feedforward networks. Gradient descent in its purest form is a technique for function optimisation, neural network training can be seen as an optimisation problem. Backpropagation requires the transfer function that governs the neurons to be differentiable. The backpropagation algorithm adjusts the connection weights in the neural network in a two-phase process which consists of a *forward* and a *backward pass*. In the forward pass, information is propagated from the input to the hidden and output layer, and the error of the network is calculated using a cost function. The cost function is usually the *sum-squared-error* which is calculated with the actual y_k and desired or target output t_k of the respective neurons in the output layer as given in Equation 2.4.

$$E = \frac{1}{2} \sum_{k=1}^{N} (y_k - t_k)^2 \tag{2.4}$$

where N is the number of neurons in the output layer and E is the sum-squared-error. The gradient $\frac{\delta E}{\delta w}$ of each weight w can be computed by propagating an error backwards through the network in a manner analogues to the way activations are propagated forward. This is known as the generalised *delta rule* [181]. This is done for all instances in the training data and an *epoch* of training is completed. The process is repeated until a specified number of epochs is reached or the error E reaches some desired value.

Computational Capabilities

The computational capabilities of neural networks are determined by their topology and the number of parameters. Hornik *et al.* [96] have shown that all bounded continuous functions can be approximated to a desired error by a network with one hidden layer using neurons with continuously differentiable activation functions.

The learning complexity is the rate the network takes in converging to a solution. Blum and Rivest [17] have shown that the learning problem in neural networks is NP-complete using a neural network with n input neurons, 2 hidden neurons and 1 output neuron. Sima [197] demonstrated the inefficiency of the backpropagation algorithm for training neural network using the sigmoid transfer function, and used the approach by Blum and Rivest [17] to show how the problem is NP hard. Sima concluded that the backpropagation algorithm is inefficient for large problems given a fixed network architecture. Engel [50] presented a discussion on the complexity of learning in neural networks and concluded that heuristic methods for hard optimisation methods should be used in training neural networks.

The generalization ability of neural networks is an important measure as it demonstrates the performance of a trained network when presented with data not present in the training set. A poor choice of the network architecture can result in poor generalization even with optimal weight selection [10]. The number of neurons in the hidden layer affects the generalization performance as too many neurons may result in *overfitting* while few neurons will result in *underfitting*. The generalization performance in the case of overfitting may be improved by increasing the number of instances in the training set. Another technique is by using *weight decay* during training [1]. A successful method in improving generalisation is to provide a validation data set in addition to the training data. In this method the training algorithm monitors the generalization error on the validation set and terminates the training before this increases.

2.2.3 Overview of Recurrent Neural Networks

In contrast to feed forward networks, recurrent neural networks are dynamical systems whose next state and output depend on the present network state and input; this makes them capable of modelling dynamical systems. This feature has made them successful in problems that include

time series prediction, classification, language learning and control.

Recurrent Neural Network Architectures

A survey on recurrent neural network architectures is given in [214, 215] and some popular architectures are as follows.

1. **First-Order Recurrent Networks:** The first-order recurrent network was proposed by Elman and Zipser [49, 48] and has since been known as the Elman recurrent network. Variants in the first-order recurrent network architecture were also proposed by Williams and Zipser, Robinson and Fallside, and Jordan [177, 230, 104]. The thesis will use the Elman recurrent network architecture and further details will be provided in Section 2.2.4.

2. **Second-Order Recurrent Networks:** Second order recurrent neural networks are more suited for modelling finite-state behaviour than first-order context layer networks [69]. It has been shown that deterministic finite-state automata can be directly encoded in them [69]. Due to their architecture, second-order networks have more weight connections than first-order recurrent networks with the same number of hidden neurons [80]. Second order recurrent networks take more computational resources for training [80].

3. **NARX Networks:** NARX recurrent networks [126] are based upon non-linear autoregressive models with exogenous inputs [196] which have limited feedback that comes only from the output neuron rather than from hidden states. It has been shown that NARX networks are as computationally powerful as fully connected recurrent networks although they contain limited feedback [196]. They can retain information up to two or three times longer than conventional recurrent neural network architectures and hence can alleviate the problem of

long-term dependencies [2].

4. **Long Short Term Memory Networks:** Long Short Term Memory Networks (LSTM) networks [93] have been proposed to overcome the problem of learning long-term dependencies. LTSM networks can be trained using multi-grid random search, time-weighted pseudo Newton, discrete error backpropagation, and expectation maximization. LTSM solves complex long time lag tasks that have never been solved by previous recurrent network algorithms of that time. It also works with local, distributed, real-valued, and noisy pattern representations. Evolutionary computation methods have also been used for training LSTM networks [187].

5. **Reservoir Computing:** *Liquid State Machines* (LSM) introduced by Wolfgang Maass [137] and *Echo State Networks* (ESN) by Herbet Jaegar [101] are the main approaches in *reservoir computing* [134]. In reservoir computing, a recurrent neural network called a reservoir is randomly created and remains unchanged during training. Only the weights from the reservoir to the output are adapted, which are linear and do not have any recurrent connections. ESNs use simple sigmoid transfer functions in the untrained recurrent network part (which is called a *dynamical reservoir*) and the resulting states are called *echos* of its input history [101]. LSMs use more sophisticated and biologically realistic models of spiking integrate-and-fire neurons, and employ dynamic synaptic connections in the reservoir [137]. The reservoir is referred to as the *liquid* which follows an intuitive metaphor of the excited states as ripples in a pool of water. LSMs are more difficult to implement and tune, and are more ex-

[2]Long term dependency problem arises when there are long time lags between two points in a sequence of a time series [37]. In the cases of recurrent networks, the desired output depends on inputs presented at times far in the past. It has been identified that recurrent neural networks trained with backpropagation-through-time have difficulty in learning sequences with long time lags [13].

pensive to simulate when compared to ESNs. In some instances of ESNs, evolutionary algorithms have been used to optimize the reservoirs [100] rather than randomly generating reservoirs. Reservoir computing approaches have been used in many engineering applications [134].

2.2.4 First-Order Recurrent Networks

First-order recurrent neural networks use context units to store the output of the state neurons from computation of the previous time steps. The context layer is used for computation of present states as they contain information about the previous states. Manolios and Fanelli have shown that first-order recurrent networks can learn and represent deterministic finite-state automata [139]. The Elman and Jordan network architecture have been trained using evolutionary algorithms [167]. The computational power of Elman recurrent networks has been studied and it has been shown that their dynamical properties can represent any finite-state machine [120].

The Elman architecture [48] uses the context layer which makes a copy of the hidden layer outputs in the previous time steps as shown in Figure 2.2. The dynamics of the change of hidden state neuron activations in Elman style recurrent networks is given by Equation (2.5).

$$y_i(t) = f\left(\sum_{k=1}^{K} v_{ik}\, y_k(t-1) + \sum_{j=1}^{J} w_{ij}\, x_j(t-1)\right) \tag{2.5}$$

where $y_k(t)$ and $x_j(t)$ represent the output of the context state neuron and input neurons respectively. v_{ik} and w_{ij} represent their corresponding weights. $f(.)$ is a sigmoid transfer function. This architecture can be expanded to include additional hidden layers [49].

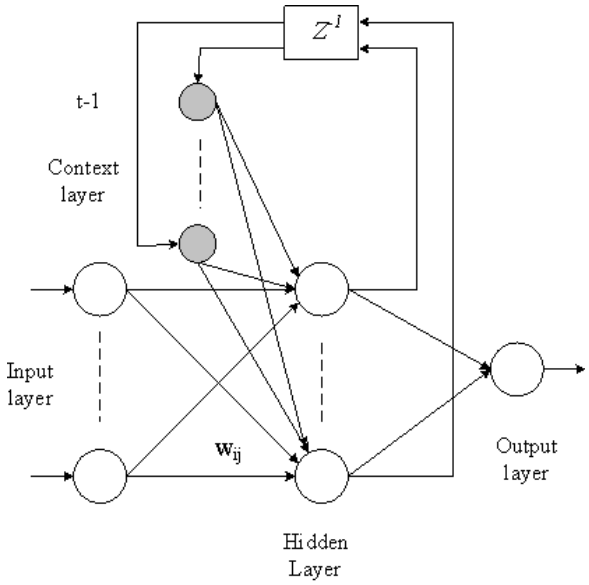


Figure 2.2: Elman RNN architecture [48]

2.2.5 Learning in Recurrent Neural Networks

Gradient descent is most widely used for recurrent network training in two variations: backpropagation-through-time [226], and real-time-recurrent learning [229]. Evolutionary computation methods have also been used for training recurrent networks [2, 75, 77].

Recurrent neural networks can be trained with the principle of the delta learning rule as discussed earlier. The general idea behind the delta learning rule is to use gradient descent to search the hypothesis space of the weight vectors and find the weights that best fit the set of training examples.

Initially, backpropagation-through-time (BPTT) was used as the standard algorithm for training recurrent neural networks [226]. Although the BPTT algorithm has been successful in training recurrent neural networks for a number of real world problems, there are limitations in learning sequences with long-term dependencies. The error gradient approaches zero

for longer term dependencies, which in turn reduces the weight update [13, 92]. To overcome this shortcoming, the long short-term memory networks were developed [93]. The long-term dependency problem have also been partially solved by formulating RNN as state space models which have shown to learn time lags of at least 100 time steps as discussed in [185].

BPTT is the extension of backpropagation algorithm. The BPTT algorithm unfolds a recurrent neural network in time into a deep multilayer feedforward network and employs the error backpropagation for weight update. This can be done by adding a layer for each time step. When unfolded in time, the network has the same behaviour as a recurrent neural network for a finite number of time steps. BPTT has time complexity of $O(n^3)$.

BPTT uses the backward propagation of error information to compute the error gradient used in the weight update. An alternative approach for computing the gradient is to propagate the error gradient information forward. Real-time recurrent learning is a real time learning algorithm which updates the weights at the end of each sample string presentation with the delta rule. The algorithm computes the derivatives of states and outputs with respect to all weights as the network processes the sequence during the forward step [229].

2.2.6 Learning Finite-State Machines with RNNs

Finite-state machines have been used to demonstrate knowledge representation and learning in recurrent networks [139, 68]. Symbolic knowledge in the form of a finite automaton can be encoded into recurrent neural networks to enhance training [158]. Knowledge extraction from recurrent neural networks aims at extracting the underlying models of the learnt knowledge in the form of finite state machines [224, 159, 30, 115]. This thesis will use finite state machines as test beds for training recurrent neural

networks.

Finite-state automata and their corresponding languages can be viewed in terms of a general paradigm of temporal and symbolic language. There is no feature extraction necessary for recurrent neural networks to learn these languages. The knowledge acquired in recurrent neural networks through learning corresponds with the dynamics of finite-state automata [68]. The representation of automata is a prerequisite for learning its corresponding languages; i.e. if the architecture cannot represent a particular automaton then it would not be able to learn it either. It has been shown that recurrent networks can represent certain finite-state machines that can be mapped directly into them [158, 160].

In particular, the dynamics of recurrent neural networks corresponds well with some instances of finite-state machines (FSM), such and deterministic finite-state automata (DFA) and fuzzy finite-state automata (FFA) [20]. A deterministic finite automaton is a finite automaton where for each pair of state and input signal, there is one successor state. The output of a deterministic finite automaton is either an accepting or rejecting state. A fuzzy finite automaton is a finite-state automaton where for each pair of state and input signal, there is a set of possible successor states.

Grammatical inference refers to the process of learning formal languages form a set of observations. The observations can be a set of data which are sequential or structured in specific ways such as strings, words, trees, terms or limited forms of graphs [40]. The task of an algorithm for grammatical inference is to return a grammar that should in some way explain these data. The rest of this discussion will restrict to the use of strings for data as grammatical inference and recurrent neural networks as algorithms for learning them. The data consists of a set of examples which are positive, negative or with some fuzzy value.

A formal language is a set of strings of symbols over some alphabet. An alphabet Σ is a finite set of symbols. Simple alphabets, e.g. $\Sigma = 0, 1$, are typically considered in the study of formal languages since results can

easily be extended to larger alphabets. The set of all strings of odd parity $L = \varepsilon, 1, 01, 001, 111, 0101, ...$ is an example of a simple language. The symbol ε is used to denote a null string. The language contains an infinite number of strings. A formal definition on deterministic and fuzzy finite-state automata is given in Definition 1 and 2, respectively.

Definition 1. *A deterministic finite-state automata is defined as a 5-tuple* $M = (Q, \Sigma, \delta, q_1, F)$*, where* Q *is a finite number of states,* Σ *is the input alphabet,* δ *is the next state function* $\delta : Q \times \Sigma \rightarrow Q$ *which defines which state* $q' = \delta(q, \sigma)$ *is reached by an automaton after reading symbol* σ *when in state* q*,* $q_1 \in Q$ *is the initial state of the automaton (before reading any string) and* $F \subseteq Q$ *is the set of accepting states of the automaton [20].*

The language $L(M)$ accepted by the automaton contains all the strings that bring the automaton to an accepting state. The languages accepted by DFAs are called regular languages. Figure 2.3 shows the DFAs selected from the Tomita grammar which will be used for training the recurrent network in this thesis.

Definition 2. *A fuzzy finite-state automaton (FFA)* M *is a 6-tuple,* $M = (\Sigma, Q, R, Z, \delta, \omega)$*, where* Σ *and* Q *are the input alphabet and the set of finite states, respectively,* $R \in Q$ *is the automaton's fuzzy start state,* Z *is a finite output alphabet,* $\delta : \Sigma \times Q \times [0, 1] \rightarrow Q$ *is the fuzzy transition map, and* $\omega : Q \rightarrow Z$ *is the output map [20].*

A restricted type of fuzzy automaton is considered whose initial state is not fuzzy, and ω is a function from F to Z, where F is a non-fuzzy set of states, called finite states. The transformation of a fuzzy automaton to its corresponding deterministic acceptor is discussed in [70]. Figure 2.4 shows an example of a FFA with its corresponding deterministic acceptor, which is used for training recurrent neural networks. This FFA has been used to show that RNNs can be trained by evolutionary algorithms [15].

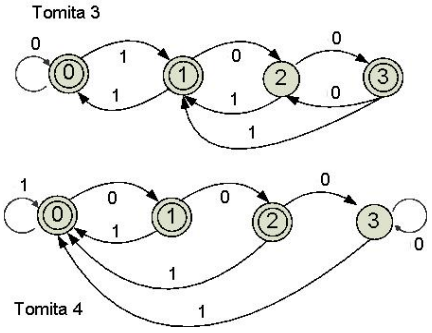


Figure 2.3: Deterministic Finite-State Automata from the Tomita grammar. Double circles in the figure show accepting states while rejecting states are shown by single circles. State 1 is the automaton's start state.

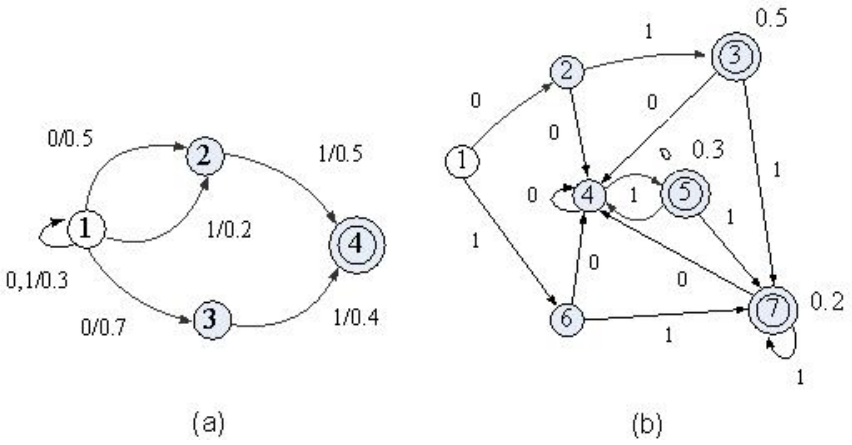


Figure 2.4: The fuzzy finite-state automata (a) and its equivalent deterministic acceptor (b). The accepting states are labelled with a degree of membership. State 1 is the automaton's start state. Accepting states are drawn with double circles.

The Tomita Grammar [213] has been used as a benchmark in order to evaluate RNN training algorithms and architecture [26]. They have been used to show the performance of the real-time recurrent-learning algorithm (RTRL) in training a generalised recurrent network architecture [58] and for the optimisation of the RNN architecture during training by an evolutionary algorithm [45, 2]. There are seven languages in the Tomita grammar as summarised in Table 2.1 and two of them (Tomita 3 and 4) are shown in Figure 2.3.

Language	Description
1	1*
2	(1 0)*
3	Accepts any binary string without an odd number of consecutive 0's after an odd number of consecutive 1's
4	Accept any binary string without more than two consecutive 0's
5	Accepts any binary string of even length which has an odd number of (01)'s or (10)'s while making pairs
6	Accepts any binary string such that the absolute difference between the numbers of 1's and 0's is a multiple of 3, i.e (Number of 1's - Number of 0's) mod 3 = 0
7	0 * 1* 0 * 1 *

Table 2.1: The Tomita Grammar. * indicates any number of repetitions. For instance, (10)* means any sequence of '10', such as '10', '1010' and '101010' etc.

The training and test datasets are generated by presenting strings of different lengths to a finite-state automata and labelling them with the output given by the automata. For instance, the training data set can be generated by presenting 500 different random strings of length 10 to 25 to a DFA as shown in Figure 2.3. The DFA labels the output on each string as

it takes input depending on the state where the final symbol of the string was presented. Similarly, the testing data set can be generated for different string lengths. Figure 2.3 (Tomita 3) shows the 4 state DFA where double circles show accepting states. Rejecting states are shown by single circles; state 1 is the automaton's start state. The training and testing sets are obtained upon the presentation of strings to this automaton which gives an output i.e. a rejecting or accepting state depending on the state where the last sequence of the string was presented. For example, the output of a string of length 6, i.e. 010010 is state 2. It is an accepting state; therefore the output label is 1.

2.3 Evolutionary Computation

Evolutionary computation aims to solve difficult optimisation problems that are unsolvable by conventional mathematical approaches. These problems are usually non-deterministic polynomial-time hard (NP hard). Evolutionary computation is loosely inspired from biological evolution and employs a population of candidate solutions which are evolved over time for a specific goal. The goal is usually minimization of maximization of a single or several objective functions.

Simulation of evolutionary algorithms began with the work of Nils Aall Barricelli in the 1950's at the Institute for Advanced Study in Princeton, New Jersey [9]. Following this, Alex Frazer published papers on artificial selection and evolution [56]. Fogel *et al.* introduced evolutionary programming in 1960's [54]. In the 1970's, Ingo Rachenberg used evolution strategies to solve complex engineering problems [172]. Genetic algorithms became popular as introduced by John Holland [95].

The common ground between most evolutionary algorithms is that they use a population of solutions, and the differences lie in the way they represent the problem and the way they form new solutions. The main strengths of evolutionary algorithms are that they are considered to be

global search methods and can be used for problems that are noisy and change over time. Evolutionary algorithms are problem independent, they can be applied to problems irrespective of their nature and hence they can be used for problems that are non-differentiable. The term *generation* is mainly used for an iteration in which new individuals for a population is created in the evolutionary process. The major areas of evolutionary computation are *evolutionary algorithms*, *swarm intelligence* and other techniques given as follows.

2.3.1 Real Coded Genetic Algorithms

The original genetic algorithm (GA) introduced by Holland [95] has evolved significantly in order to suit the real-world optimisation challenges faced by engineers and scientists. Traditionally, candidate solutions used binary encoding with 1's and 0's. The development of real coded genetic algorithms (RCGA) use real numbers in their candidate solutions rather than binary encoding. Real-valued encoding is also a more natural presentation to use for many real world applications. The computational and optimisation power of genetic algorithms has been demonstrated in several theoretical studies [73, 173]. They have been successfully deployed in several real-world problems [232, 142]. This thesis will use RCGA in the sub-populations of cooperative coevolution for evolving neural networks.

In RCGA, the population of candidate solutions are called *individuals*. New solutions are called *offspring* which are built by combining candidate solutions which are known as *parents*. RCGA optimisation is the same as in Holland's original scheme where a number of individuals make up a population. Each individual is evaluated according to its *fitness* and employs *genetic operators* such as *selection*, *crossover* and *mutation* for producing offspring. The offspring are added into the population while sometimes, less fit individuals are removed and the process is repeated. In the standard procedure, two or more parents are selected using the selection operator

from a population to make a predefined number of offspring.

The fitness function measures the quality of the solution which is problem dependent. The choice of the appropriate genetic operator is important as it directly influences the convergence of the genetic algorithm. Different forms of the main genetic operators are needed according to the type of the genetic algorithm and the nature of the optimisation problem. An overview of the main components of a genetic algorithm is discussed below.

1. **Initialisation:** In the initialisation stage, candidate solutions or individuals are typically randomly generated. The number of individuals in the population is determined according to the problem, and in many cases, empirically evaluated in trial experiments. In some cases, the individuals are seeded in the area of search space where the desired solution is likely to be found.

2. **Selection:** The selection operator plays an important role as it encourages qualities from fitter individuals to survive into future generations. Some common selection strategies are rank selection [227], fitness proportionate or roulette wheel selection [8], tournament selection [8] and the elitist strategy [44]. In rank selection, the selection is done according to the fitness rank of the individual; the most-fit individuals have priority over lesser ones. In roulette wheel selection, the selection gives priority to those individuals with greater fitness; however, less fit chromosomes are chosen occasionally as they may contain useful genetic material for the future. In the elitist strategy, some of the fittest individuals are always retained in the new population to make sure that the best performing chromosomes always survive. Tournament selection involves running several *tournaments* with few individuals chosen randomly from the population. The winner from the tournament is chosen for crossover or mutation. The selection pressure is easily adjusted by changing the tournament

size as with a larger tournament size, weaker individuals will have a lower chance of selection.

3. **Reproduction using Crossover:** The main reproduction operators are *crossover* and *mutation*. The crossover operator exchanges genetic material from selected parents and forms either a single or multiple offspring. Some common crossover operators for real-coded genetic algorithms are flat crossover [173], simple crossover [73, 143], arithmetic crossover [143], linear breeder crossover [186], and Wright's heuristic crossover which has also shown superior performance compared to binary GA for a set of optimisation problems [232]. Eshelman and Schaffer introduced the blend crossover for two parents [51]. Muhlebein and Voosen introduced extended line crossover and extended intermediate crossover which is a special case of the blend crossover [152]. Deb and Agarwal introduced the simulated binary crossover (SBX) [41], which simulates the principle of the single-point crossover operator on binary strings for continuous domain. SBX works on a pair of parents and produces a pair of offspring. Simplex crossover showed good performance in multi-modal optimisation problems [218]. Deb *et al.* [42] introduced the parent-centric crossover (PCX) operator which reported the best performance than any other optimisation technique in literature for the given test functions. However, the PCX operator was used for only three test functions which were unimodal. The modified version proposed by Sinha *et. al* [198] showed significant improvement but was heavily dependent on prior knowledge of the problem. Laplace crossover which is a self adaptive parent centric operator showed good performance when compared to the genetic algorithms studied in [43]. Section 2.3.2 will give further details on the PCX as it will be used in this thesis.

4. **Reproduction using Mutation:** The mutation operator provides ran-

dom diversity in the population. This is important when the algorithm gets trapped in a local minimum [142]. Some common mutation operators include random uniform mutation and Michalewicz non-uniform mutation [142]. In uniform mutation, a random number in the range of $[a, b]$ is added to a selected gene where a and b are the highest and lowest values in the chromosome, respectively. In non-uniform mutation, the strength of mutation is decreased as the number of generation increases. The effect of the mutation rate, the strength of mutation and its impact in building better solutions have been studied in [199].

5. **Termination:** The evolutionary process in the genetic algorithm continues until the termination condition is satisfied. The search is terminated when typically one of the following is true. 1) Fixed number of generations or function evaluations is reached, 2) a solution with a desired fitness is found, or 3) the highest ranking solutions come to a point where there is no change of fitness indicating no improvement in candidate solutions.

In addition, several techniques have been proposed to improve the performance of genetic algorithms. Fan and Xu introduced three improved genetic algorithms which implement: 1) a dual fitness function to adapt the mutation probability, 2) a new evolutionary directional operator, 3) and a probabilistic binary search respectively to reveal a new offspring [52]. Other modifications include a stopping rule based on asymptotic considerations and mutation scheme based on particle swarm optimisation [216]. Further improvements are by incorporating local refinement using memetic approaches [149, 163] and hybrid evolutionary methods [130]. These will be discussed in detail later in Section 2.4.

2.3.2 G3-PCX: Generalised Generation Gap with PCX

The generalised generation gap differs from a standard genetic algorithm in terms of selection. In G3-PCX, the whole population is randomly initialised and evaluated as done in the canonical genetic algorithm. The difference lies in the selection method where a small sub-population is made of few chosen parents and children. At each generation, only the sub-population is evaluated rather than evaluating the whole population as in a standard genetic algorithm. The children with their new fitness become part of the bigger population. The best individual of the population is retained at each generation.

The parent centric crossover operator is used in creating an offspring based on orthogonal distance between the parents. The parents are made of female and male component. The *female* parent points to search areas and the *male* parent is used to determine the extent of search of the areas pointed by the female. The genes of the offspring extract values from intervals associated in the neighbourhood of the female and male using probability distribution. The range of this probability distribution depends on the distance among the genes of the male and the female parent. The parent-centric crossover operator assigns more probability to create and offspring near the female than anywhere else in the space. The general procedure used in the G3-PCX is given in Algorithm 1.

Algorithm 1 begins by initialising and evaluating all the individuals in the population. The sub-population is then created which has the size of number of parents and the children. The number of parents and children must be defined beforehand. The best parent is selected from the population and the rest of the parents are selected randomly. The selection of the best parent ensures elitism in the procedure. The children are created from the parents using the PCX crossover operator given in [42]. The parents and the children are combined in the sub-population. Afterwards, n strong individuals are chosen from the combined sub-population which are further replaced in the population.

Alg. 1 G3-PCX Evolutionary Algorithm [42]

Set the number of *parents* (α) and *children* (β)
initialise and evaluate all individuals in the *population*
Setup the *sub-population* which would contain parents and children
while not optimal solution **do**
1) Select the best *parent* and α - 1 *parents* randomly from *population*
2) Create β *children* from α *parents* using PCX
3) Choose two parents at random from the *population*
4) From the combined *sub-population* of two chosen parents and β created children, choose the two best individuals and replace the chosen two parents (in Step 3) with these solutions..
end while

The parent-centric crossover operator was compared with simplex crossover and simulated-binary crossover using the generalized generation gap model. The results showed that the parent-centric crossover operator achieved improved performance in terms of lower optimisation time and also scaled up better as the problem size was increased. These simulations were limited to 3 unimodal functions and further comparisons of G3-PCX with Differential Evolution [206] and Evolution Strategies [84] showed improved performance in terms of optimisation time and scalability [42].

The advantage of the parent-centric crossover operator is that it behaves like a mutation operator and at the same time retains diversity and is self-adaptive. It has been used as a hill-climbing local search procedure in a memetic algorithm [132]. There is no mutation operator in the original G3-PCX algorithm. Amendments to the original algorithm have been done by proposing a mutation operator [209] which has shown good performance in multi-modal problems. Further amendments have been done in the parent-centric crossover operator by introducing a female and male differentiation process which determines the male and female individuals chosen from the population and by further using parent selection mechanisms shown in [66]. A roulette wheel based parent selection scheme has also shown to perform better than the original G3-PCX on highly non-

linear multi-dimensional problems [176].

The G3-PCX needs fairly large population for small problems [171]. In a study, several short experiments revealed that even for 2-dimensional problems, a population size of 90 was needed to find the solution reliably. The population size of 300 was needed in order to solve a 40-dimensional sphere function [171].

A major limitation of G3-PCX is in multi-modal optimisation problems as shown in [171] where a restart schemes also showed to be inappropriate. This problem can be handled if more diversity is given to G3-PCX as parent-centric crossover has more emphasis for local search.

2.4 Hybrid Meta-heuristics and Memetic Algorithms

Meta-heuristics (MH) refer to the family of search algorithms that have extended basic heuristic methods by extending exploration capabilities [72]. The term *metaheuristic* was first introduced by Glover in 1986 [71] which is derived from the composition of two Greek words. The word heuristic is derived from the verb *heuriskein* which means "to find" while *meta* means "beyond, in an upper level". Metaheuristic methods have been useful for approximately solving difficult optimisation problems. They do not guarantee a global optimal solution, however, near global optimal solutions can be found faster in comparison to conventional approaches for combinatorial optimisation problems [19]. Examples of classic meta-heuristic approaches include simulated annealing, scatter search, Tabu search, evolutionary algorithms, ant colony optimisation, path re-linking, multi-start and iterated local search, greedy randomised adaptive search procedures and artificial immune systems [72, 19].

Hybrid metaheuristics (Hybrid MHs) [130] refer to the group of hybrid algorithms where two or more metaheuristic search techniques are com-

bined to solve difficult problems. They provide *diversification* and *intensification* during the search process for obtaining a stronger solution. Diversification refers to the ability to visit many different regions in the search space while intensification refers to the ability to obtain high quality solutions in a particular search space [130] . Diversification is often referred as *global search* or *exploration*. Intensification is also referred to as *local search, local refinement* or *exploitation*. A major goal of hybrid meta-heuristic is to balance diversification and intensification in the search process. The goal is to obtain global optimum and quality solution through proper diversification and intensification. Examples of intensification and diversification components in meta-heuristics are genetic or evolutionary operators such as selection, crossover and mutation, local search and collaboration with other metaheuristic methods [19].

Gunther [174] grouped hybrid MHs into two major categories that are collaborative hybrid MHs and Integrative hybrid MHs. Lozano and Garcia-Marninez [130] have grouped hybrid MHs into a third category that is MHs with evolutionary intensification and diversification components.

2.4.1 Collaborative Hybrid MHs

Collaborative hybrid HMs are based on the exchange of information between different metaheuristics that run sequentially or in parallel [174]. Collaborative hybrid MHs are further devided into subcategories that include *teamwork* and *relay* collaborative MHs.

Teamwork collaborative MHs

Teamwork collaborative MHs use several metaheuristics that include evolutionary algorithms that work in parallel [212, 88]. They partition the population into several sub-populations where each sub-population is independently processed by the evolutionary algorithm. There is information exchange between sub-populations, and intensification and diversi-

fication are carried out by means of different parameter values, population sizes and genetic operators. Teamwork collaborative MHs were initially known as distributed genetic algorithms [88]. Potts *et. al* [170] presented distributed binary-coded genetic algorithms that used four sub-populations with different mutation probabilities. After certain generations, the best individuals from three sub-populations are added into one sub-population. Tsutsui *et. al* [217] used an explorer population with an exploiting population that used different mutation schemes and population sizes for diversification and intensification.

Herrera *et. al* [89] presented gradual distributed real-coded genetic algorithms that used several different sub-populations for diversification and intensification and evolved in parallel with the migration of information. Schlierkamp-Voosen *et. al* [186] presented heterogeneous distributed genetic algorithms that integrates different EAs whose population sizes were dynamically adjusted during evolution. The *COSEARCH* method proposed by Talbi and Bachelet [208] uses in parallel, the combination of an EA, Tabu search and a local search procedure that communicates via an adaptive memory that contains the history of the search already done. The method was tested on the *quadratic assignment problem* and has shown to give the best known solutions.

Relay collaborative MHs

Relay collaborative MHs use a pipelined fashion to execute meta-heuristic search methods where the output of each method is used as the input of the other. They follow the heuristic where exploration is done in the initial stages and exploitation in later stages. Chelouah *et. al* [32] presented a hybrid MH that used two main stages where a specified genetic algorithm is used for diversification and intensification is followed by a local search procedure from the best solution found in the first stage. Chandra and Omlin [31] trained recurrent neural networks using genetic algorithms in the first stage and used the best solution for refinement using

back-propagation-through time in the second stage. Garcia-Martinez *et. al* [66] proposed a procedure that determines the male and female components in the population of real-coded genetic algorithms, which then apply parent-centric crossover operator [42]. They proposed a male and a female differentiation process through which they designed *global real-coded GAs* and *local real-coded GAs*. These methods were combined where the global real-coded GA was initially used for a predefined number of generations and then the best individuals were passed to the local real-coded GA. This approach gave better performance than its counterparts.

2.4.2 Integrative Hybrid MHs: Memetic Algorithms

Integrative hybrid MHs use embedded methods such as local improvement of candidate solutions by an inner optimisation algorithm and exact techniques for searching very large neighbourhoods [174]. Memetic algorithms are well known methods for this category where a master MH is used for diversification and a subordinate MH is used for intensification. Memetic algorithms address the shortcomings of EAs in balancing diversification and intensification.

Memetic algorithms (MAs) [149] typically combine population-based evolutionary algorithms with individual learning, local refinement or local search (LS) in order to provide an improved global solution. Memetic algorithms also include the combination of EAs with problem dependent heuristics and approximate methods and special recombination operators [151]. Applications of memetic algorithms include NP hard combinatorial optimisation problems, machine learning and robotics such as pattern classification and training neural networks, molecular optimisation problems, electronics and engineering and other optimisation problems as discussed in [151]. In literature, MAs are often referred to as Baldwinian EAs, Lamarckian EAs, cultural algorithms or genetic local search.

The term *meme* was introduced by Dawkins [39] in evolutionary biol-

ogy which denotes the *basic unit of cultural transmission* that undergo social evolution and learning. Examples in nature include learning tunes, ideas, skills such as making pottery, art, and fashion. The term *memetic algorithm* (MA) was introduced by Moscato in his report [150] where he viewed memetic algorithms as a population based hybrid genetic algorithm with local search. A review on memetic algorithms appears in [164] and a progress report indicates that they are an emerging field in evolutionary computation [162].

Evolutionary algorithms have been used as master MH in memetic algorithms for diversification and also as subordinate MH for intensification. These will be discussed in the subsections to follow.

Memetic Algorithms with EAs for Diversification

Initially, memetic algorithms have used EAs for diversification combined and local search methods such as *hill-climbing* for intensification. Initial work was done by Moscato who used a genetic algorithm for diversification with local search for intensification [150]. Lozano *et. al* [133] presented memetic algorithm with crossover hill-climbing as a local search. The crossover operator repeatedly produces a fixed number of offspring from which the best is selected. The method showed better performance than G3-PCX and other memetic algorithms.

Seront *et. al* [188] presented a memetic algorithm that used clustering methods which allows the local search to avoid multiple rediscoveries of local optima. In this way, the computational cost of the local search is saved. The clustering method also supplies information that can be used to maintain population diversity. Kemenade [222] presented a memetic algorithm based on evolution strategies that uses a similar idea of using clustering to avoid premature convergence.

Krasnogor and Smith [119] presented a memetic algorithm where a genetic algorithm is used with the *Monte Carlo method*. The Monte Carlo method is used for as a local search when the population is diverse and

later the goal of the Monte Carlo method is to provide diversification when the population converges. The experiments were performed on the travelling salesman and protein folding problems. The method performed better than standard GA and two other MAs.

Ong and Keane [161] presented a meta-Larmarckian memetic framework where several different types of local searches are employed during evolution. Initially, all local searches are given a chance and hence their fitness is measured which is kept in future so that roulette wheel selection can be used to select the local search. The method showed high quality and efficient performance on classic benchmark functions for continuous optimisation and a real world aerodynamic design problem. Krasnoger and Gustafson [118] presented self generating memetic algorithms that create local searches and co-evolve the behaviours it needs to successfully solve the problem. Smith [200] presented a review on co-evolving memetic algorithms in which a rule-based representation of local search is coadapted alongside candidate solutions within a hybrid evolutionary algorithm. Nguyen *et. al* [154] presented a probabilistic memetic framework that analyses the probability of the process of individual learning in locating global optimum.

Memetic Algorithms with EAs for Intensification

EAs have also been used for local search in memetic algorithms. The EA employed for local search has a small population size, which is then evolved over a short duration [109, 133, 201, 146].

Kazarlis *et. al* [109] introduced the concept of micro genetic algorithm for local search where a population of few individuals was employed as a generalised hill-climber intended for intensification and a genetic algorithm with a larger population was used for diversification. The major advantage of micro genetic algorithm over other local search methods was to identify and follow narrow ridges of arbitrary directions that lead to the global optimum. The proposed method was tested with 13 different

methods which include a simple GA with different hill-climbers. The micro genetic algorithm based local search performed better than all other methods in terms of accuracy, feasibility rate and robustness for the same number of fitness evaluations.

Lozano *et. al* [133] presented a real-coded memetic algorithm with crossover hill-climbing (XHC). XHC maintains a pair of parents which consists of the solution being refined and the best solution obtained so far. XHC performs crossover on the pair until some number of offspring has been reached. The best offspring is selected and replaces the worst parent only if the best solution is better. The method performed better than the memetic algorithms presented in literature. Noman and Iba [155] incorporated an adaptive XHC for differential evolution where the depth or intensity of local search is adjusted adaptively. They proposed fixed length and an adaptive method for the intensity of local search. In the adaptive method, the XHC is evolved while the offspring performs better than the first parent. If the performance of the offspring is worse, then the search returns to the differential evolution algorithm. The method showed better performance than other memetic based differential evolution algorithms from literature.

Soak *et. al* [201] presented a memetic algorithm that used ideas from particle swarm optimisation used for diversification and recombination operators from a genetic algorithm was used for intensification. The method obtained promising results to several instances of the degree constrained minimum spanning tree problem. Mutoh *et. al* [153] presented the flexible-step crossover operator that performed local search and results showed improved performance for continuous optimisation problems. Gang *et. al* [59] used a global genetic algorithm as the master and a local genetic algorithm as the subordinate for the travelling salesman problem.

Molina *et. al* [146] used the CMA-ES (covariant matrix adaptation evolution strategies) [84] with a small population as a subordinate EA for local search. They used a steady-state genetic algorithm [90] as master EA

which has the property of high population diversity by means of BLX- α crossover operator and negative assorted mating strategy. The BGA mutation [152] operator was also used to favour the diversity of the main population. The method showed good results for continuous problems with high dimensions when compared to its counterparts from literature.

2.4.3 MHs with evolutionary intensification and diversification

Lozano and Garcia-Marninez [130] have identified a third group of hybrid MHs that incorporate evolutionary intensification and diversification components as evolutionary operators in a customised EA. They identified two examples in literature that incorporate recent EAs for intensification as local search procedures, which include binary local GA (BLGA) [60] and the covariance matrix adaptation evolution strategy (CMA-ES) [84, 83].

CMA-ES was introduced to improve the local search performance of evolution strategies. It has global search features and is very good in exploiting local structures in search spaced for continuous optimisation problems. In CMA-ES, the step size and direction are adjusted at each generation in a multidimensional problem. There is a mutation strength per dimension and their combined update is controlled by a covariance matrix whose elements are updated with the evolutionary process. CMA-ES is an instance of multi-start L-CMA-ES [7] which was one of the winners of real parameter optimisation competition at 2005 congress on evolutionary computation.

BLGA uses a steady state binary-coded genetic algorithm that uses crowding replacement method for preservation or diversity. The individuals from the population are selected by positive assorted mating which ensures that they are very similar to the leader chromosome. This orientates the search in the nearest regions to the leader chromosome. The method outperformed its counterparts from literature which used other

local search approaches.

2.5 Cooperative Coevolution

In the evolutionary process of nature, different species compete in order to survive with the given resources. The individuals of a particular group of species mate among themselves in order to produce stronger individuals. Cooperative coevolution (CC) is an evolutionary computation method inspired from nature which divides a large problem into subcomponents and solves them collectively in-order to solve the large problem [169]. The subcomponents are implemented as sub-populations that are evolved in isolation. Diversity is an important feature of cooperative coevolution. The combination of the individuals in the sub-populations will lead to more diverse solutions when compared to a single population evolutionary algorithm [168]. Cooperative coevolution has mainly been used for tackling large-scale optimisation problems [127, 233] and neuro-evolution of feedforward [168, 61] and recurrent networks [76, 77].
The original cooperative coevolution [169] can be summarised as follows.

1. **Problem decomposition:** Decompose a high dimensional problem into subcomponents that can be solved by conventional evolutionary algorithms. The subcomponents can vary in sizes and are often expressed as sub-populations.

2. **Subcomponent optimisation:** Evolve each subcomponent separately by an evolutionary algorithm where evolutionary operators such as crossover and mutation are restricted to a subcomponent and do not affect other subcomponents.

3. **Fitness evaluation:** Fitness of individuals in each of the subcomponents are evaluated cooperatively with representative examples from the other subcomponents.

Problem decomposition is considered to be an important step in cooperative coevolution. The problem decomposition method solely relies on the nature of the optimisation problem. The original cooperative coevolution framework (CCEA) decomposed the problem in a way where a single sub-population is used for each variable [169]. The sub-populations in the cooperative coevolution framework are evolved separately and the cooperation only takes place for fitness evaluation for the respective individuals in each sub-population. The general framework for function optimisation is given in Algorithm 2 which outlines how the large problem is decomposed. The algorithm begins by initialising and cooperatively evaluating each of individuals of the respective sub-populations. After the initialisation and evaluation phase, the evolution proceeds. All the sub-populations are evolved in a round-robin fashion for the *depth* of n generations . A *cycle* is complete when all the sub-populations have been evolved for n generations. The algorithm terminates until the maximum number of cycles are reached or the minimum error is satisfied. The size of a subcomponent and the way a subcomponent is encoded is dependent on the problem. The way the algorithm cooperatively evaluates each subcomponent and the way the problem is decomposed have been a major focus in the study of cooperative coevolution.

Existing cooperative coevolutionary algorithms employ different methods for problem decomposition as follows.

- **One-dimensional decomposition:** A separate subcomponent is used for each dimension or decision variable and is very effective for separable problems [169].

- **Decomposition by splitting in halves:** The splitting in half problem decomposition method begins the evolution by dividing the n-dimensional vector into 2 m-dimensional vectors [193]. The division can be repeated further after certain stages in the evolutionary process.

Alg. 2 The General Cooperative Coevolution Framework

1) Decompose the problem into k subcomponents
2) Initialise and cooperatively evaluate each subcomponent represented as a sub-population
while until termination **do**
  for each Subpopulation **do**
    for n Generations **do**
      i) Select and build new individuals
      ii) Cooperatively evaluate the new individuals
      iii) Update sub-population
    end for
  end for
end while

- **Adaptive and heuristic subcomponents:** Adaptive and heuristic methods build subcomponents according to the nature of the problem in terms of non-separability [233, 156]. In multi-level cooperative coevolution, the size of a set of subcomponents are predefined. The set is used to adapt the size of subcomponents in the evolutionary process [234]. Some techniques use statistical methods such as correlation to determine the subcomponent sizes in the initial stages of the evolutionary process [175].

- **Problem-based subcomponents** In using cooperative coevolution for neural network training, the subcomponent sizes are determined by the network architecture and connectivity of the neurons [168, 61, 77].

Potter and Jong [168] proposed a *collaborative* method for fitness evaluation. The method obtains the fitness of each individual in a sub-population by combining it with the best individuals from the rest of the sub-populations. This method has been used to train cascade networks on the two-spirals problem and has shown to learn the task with smaller networks when compared to the cascade correlation learning architecture [168]. The fitness assignment problem in cooperative coevolution has also been viewed

as a multi-objective optimisation problem for training neural networks (MOBNET) [61]. roulette wheel section and mutation is done using simulated annealing in each sub-population [61].

2.5.1 Diversity in Cooperative Coevolution

Population diversity is a key issue in the performance of evolutionary algorithms. The diversity of a population affects the convergence of a evolutionary algorithm. A population which consists of similar candidate solutions in the initial stages of the search is prone to convergence in a local minimum. The selection pressure and recombination operations mainly affect the diversity of the population. Evolutionary operators such as crossover and mutation must ensure that the population is diverse enough in order to avoid local convergence. Diverse candidate solutions can ensure the algorithm to escape a local minimum. In evolutionary algorithms, diversity has been improved by using techniques such as 1) complex population structures [219, 211], 2) the use of specialized operators to control and assist the selection pressure [74] , 3) reintroduction of genetic materials in the population [34, 81] and 4) diversity measures such as the hamming distance [194], gene frequencies [105] and diversity measures to explore and exploit search [220].

Cooperative coevolution naturally retains diversity through the use of sub-populations, where mating is restricted to the sub-populations and cooperation is mainly by collaborative fitness evaluation [169, 168]. Since selection and recombination is restricted to a sub-population, the new solution will not have features from the rest of the subpopulations; therefore cooperative coevolution produces more diverse population when compared to a standard evolutionary algorithm with a single population.

2.5.2 Cooperative Coevolution for Non-Separable Problems

This thesis explores the issue of separability in terms of the interaction among the synapses in neural networks. Much work has been done in using cooperative coevolution for non-separable global optimisation problems; hence, they are reviewed here.

Typically, large-scale complex problems require complex algorithms for their solutions. Cooperative coevolution employs divide and conquer strategy to decompose complex problems into several simpler sub-problems and employs greedy approaches on the sub-problems. Cooperative coevolution naturally appeals to problems that can be broken down and those that do not have any interdependencies within the decision variables. These types of problems are often expressed as *separable problems*, as given in Definition 3.

Definition 3. *A function of n variables is separable if it can be written as a sum of n functions with just one variable as given in Equation 2.6. The parameters of a separable function are called independent variables [165].*

$$\underset{(x_1,x_2,...,x_n)}{\arg\min}\ f(x_1,x_2,...,x_n) = \left(\underset{x_1}{\arg\min}\, f(x_1,...), ... \underset{x_n}{\arg\min}\, f(...x_n)\right) \tag{2.6}$$

An example of a separable problem is the Sphere function as given in Equation 2.7.

$$\min_x f(x) = \sum_{i=1}^{n} {x_i}^2 \tag{2.7}$$

$$-5.12 \leq x \leq 5.12, x^* = (0,0,..,0)\ and\ f(x^*) = 0$$

In real world problems, interdependencies exist among decision variables. Non-separable problems have interdependencies between decision variables. Tang *et. al* [106] have further divided the level of non-separability

into two classes as given in Definition 4. These are *m-non-separable* and *fully-non-separable*.

Definition 4. *A non-separable function $f(x)$ is called m-non-separable function if at most m of its parameters x_i are not independent. A non-separable function $f(x)$ is called fully-non-separable function if any two of its parameters x_i are not independent [106] .*

An example of a non-separable problem is the extended Rosenbrock function as given in Equation 2.8. Examples of separable and non-separable functions with up to 1000 dimensions for testing large-scale function optimisation can be found in [106].

$$\min_x f(x) = \sum_{i=1}^{n-1}[(1 - {x_i}^2) + 100(x_{i+1} - x_i^2)] \tag{2.8}$$

The extended Rosenbrock function has been shown to have exactly 1 minimum for n=3 at (1,1,1) and exactly 2 minima for $4 \leq n \leq 7$. This result has been obtained by setting the gradient of the function equal to zero [114].

In summary, separable functions are those where the minimization of a variable independently contributes to the global minimization of the whole function. This is not applicable to non-separable problems as the minimization of a variable independently may hinder the global minimisation due to interdependencies between variables. In general, totally separable problems are the easiest to solve and the fully non-separable are the most difficult to solve. Most problems fall between the two extremes and are also known as *partially separable* [82, 35].

The *curse of dimensionality* is a term introduced by Richard Bellman [12] that signifies that the performance of an optimisation algorithm will typically deteriorate significantly as the dimensionality of the problem increases. A reason for the curse of dimensionality is due to the characteristics of the problem which may change due to the increase in dimension. For instance, the Rosenbrock function which is uni-modal for two dimensions becomes multi-modal when the dimension increases [192]. For this

reason, the search algorithm that has been successful for smaller dimensions may not be able to scale to higher dimension instances of the same problem.

In order to solve large-scale optimisation problems that fall between total separable and fully non-separable, it is important to group interacting variables within a subcomponent. Fast evolutionary programming in the cooperative coevolutionary framework (FEPCC) has been the first attempt to tackle large-scale function optimisation of up to 1000 dimensions [127]. FEPCC used the original CCEA framework by Potter and Jong [169]. The major drawback of CCEA is that it does not have the mechanism to provide interaction between subcomponents which is needed for non-separable problems. Due to this, FEPCC performed poorly on non-separable problems.

Cooperative Coevolution based Particle Swarm optimisation (CPSO) [221], unlike the original cooperative coevolution framework [169], decomposes the problem into m s-dimensional subcomponents where s is the number of variables in a subcomponent. CPSO used static grouping where the arrangement of the variables do not change during evolution. The variables of each individual in a particular sub-population are concatenated with the best individuals from the rest of the subcomponents and a *context vector* is formed in order to evaluate the individuals in each subcomponent. Shi *et. al* presented a Differential Evolution based cooperative coevolution framework which divides the problem into halves and each half is optimized using Differential Evolution [193]. The method was applied for large-scale problems of only 100 dimensions. This strategy does not work for problems of high dimension as the halved subcomponents cannot cope with higher dimensions.

Yang *et. al* [233] have presented the cooperative coevolution framework that uses a *random grouping* and *adaptive weighting* strategy with differential evolution (DECC-G) for its subcomponents. The method groups interacting and non-interacting variables into separate subcomponents heuris-

tically. During each cycle of the cooperative coevolution framework, different variables are arbitrarily assigned to each subcomponent and a separate adaptive weight vector is used to assess the subcomponents. The adaptive weight vector is also optimized using a separate optimisation process. Note that the size of the adaptive weight vector equals the number of subcomponents which is significantly small in comparison to the problem size. The adaptive weight vector is used to further fine tune the solutions obtained using cooperative coevolution. The framework was used for non-separable problems of up to 1000 dimensions. Omidvar et. al. [156] made amendments to the adapting weighting approach in DECC-G. They argued that the *random grouping* approach in DECC-G proved to capture two interacting variables in a subcomponent with a probability of 0.9662 which gets significantly lower when more than two interacting variables are present in the problem. They presented a *more frequent random grouping* approach which turned to outperform its counterpart in several non-separable problems of up to 1000 dimensions.

Yang *et. al* presented a multi-level cooperative coevolution framework (MLCC) [234] which adapts the size of the subcomponents in DECC-G in order to group interacting and non-interacting variables. The framework starts with small sized subcomponents and adapts to bigger subcomponents sizes from a predefined set. MLCC showed better performance than DECC-G for non-separable problems of up to 1000 dimensions.

An recent extension to the CPSO has been the cooperative coevolution framework for particle swarm optimisation (CCPSO) [124] which has been applied to large-scale non-separable problems using random grouping and adapting weighting present in DECC-G [233]. CCPSO showed to perform significantly better for large scale problems of up to 1000 dimensions than its differential evolution counterpart in DECC-G. The authors concluded that cooperative coevolution can be a good method for tackling the limitations of particle swarm optimisation in handling problems of high dimensions.

Ray and Yao presented a cooperative coevolution framework using correlation based adaptive partitioning technique (CCEA-AVP) [175]. In this method, the problem is evolved for a few generations using a single population and then the correlation coefficients of the top 50 % of the individuals of the population are calculated. The variables with correlation coefficients greater than a predefined threshold are grouped into one subcomponent and the rest into another subcomponent. The correlation based partitioning is then repeated after every generation. It has been shown that this technique performs better than the original cooperative coevolution framework on non-separable problems of up to 100 dimensions.

Chen *et. al* presented the cooperative coevolution with variable interaction learning (CCVIL) [33] which divides the optimisation problem into two stages. These are the learning stage and the optimisation stage that execute once in sequence. In the learning stage, all decision variables are treated independently and cooperatively evolved from which the interacting variables are identified using variable interaction learning and variables are merged into groups. In the optimisation stage, optimisation is done using conventional cooperative coevolution. The results showed that the method performs better than MLCC and DECC-G for non-separable problems of up to 1000 dimensions.

Omidvar *et. al* presented another approach to large scale non-separable problems using *Delta Grouping* [157]. This method computes the averaged difference of a certain variable across the entire population which is used for identifying interacting variables. The method is based on the idea presented by Salomon [183] who showed that coordination rotation is one way of turning a separable problem into an non-separable one. A vector is used which calculates the amount of change or delta values in each of the dimensions between two consecutive cycles for all the individuals of the population. The size of this vector equals the number of dimensions which gives a rough estimation of the improvement interval in every dimension which is used for capturing the interacting variables.

The magnitude of the delta values of with the corresponding variables are then sorted in descending order and grouped according to predetermined equally sized subcomponents. In this way, interacting variables are grouped in the same subcomponent. This algorithm is called Differential Evolution Cooperative Co-evolution using Delta-Grouping (DECC-D). A variant is also presented using the MLCC approach [234] which is called DECC-DML. Both methods showed better performance in comparison with MLCC and DECC-G for large scale non-separable problems of up to 1000 dimensions.

2.6 Neuro-Evolution

The ability of neural networks to approximate complex functions and model any open dynamical system has well been praised [184, 97, 70]. However, the search for its optimal training algorithm is still an open problem. Gradient descent based training methods are unable to guarantee acceptable solutions in difficult problems and those involving long-term dependencies [13, 55]. Therefore, evolutionary algorithms (EAs) have been used in neural network training and design in order to achieve a better solution when compared to traditional gradient descent based approaches [2]. The paradigm of using evolutionary computation for evolving neural networks is known as *neuro-evolution* .

Neuro-evolution uses evolutionary algorithms to search for the optimal weights, parameters, suitable topology and possible transfer functions in training neural networks [235]. Unlike gradient based approaches such as backpropagation, neuro-evolution uses a population of solutions in building stronger solutions for future generations. One of the main strengths of neuro-evolution is that they are not dependent upon any particular neural network architecture and can work with a set of different activation functions without being limited to differentiability. They are easily applicable without major alteration in different network architec-

tures as they search the solution space only using feedback from the overall fitness of the neural network. Another reason for its popularity is due to the use of evolutionary algorithms, which are known as global search methods.

Neuro-evolution has been praised for solving control problems that were difficult to solve by conventional reinforcement learning and other machine learning approaches [204, 77]. They have been seen to be a reinforcement learning technique as they adjust their solutions according to the overall fitness or reward of the system [228]. The optimisation strategy in neuro-evolution does not depend on the back-propagation of gradient information. This is useful in the case of learning long-term dependencies in recurrent neural networks since BPTT has problems as outlined in [13].

Neuro-evolution is divided into two different streams which are *direct* and *indirect* encodings. In direct encoding, every connection and neuron is specified directly and explicitly in the genotype whereas in indirect encoding, the genotype specifies rules or some other structure for evolving the network. The major streams are discussed in the following subsections.

2.6.1 Direct Encoding in Neuro-evolution

In direct encoding, the genotype is the same as the phenotype, every connection and neuron is specified directly and explicitly in the genotype. Angeline et. al proposed Generalised Acquisition of Recurrent Links (GNARL) which uses evolutionary programming and genetic algorithms to simultaneously search for the structure and weights in a recurrent neural network [2]. GNARL adds or deletes several nodes and links through structural mutation. The authors argued that conventional binary encoded genetic algorithms are inappropriate for neuro-evolution due to the crossover operator. The crossover operator can be harmful in training neural network weights as they do not have the feature to retain the weights for particular neurons or layers. The weights can be exchanged between layers which

can cause loss of infomation associated with different neurons. GNARL was mainly used for learning languages from grammatical inference problems. In Symbiotic Adaptation Neuroevolution (SANE), a population of neurons and a population of networks are coevolved separately. The neural network is randomly constructed from the population neurons and evolved [148].

Direct encoding has also been used in the evolution of feedforward networks for pattern recognition problems using conventional evolutionary algorithms [189], memetic based approaches [23] and cooperative coevolution [62]. Blanco *et. al*, [16] used Wright's heuristic crossover operator in an evolutionary algorithm and compared its performance with the real-time-recurrent learning algorithm for training recurrent networks for grammatical inference problems. The experiments in their work were done using second-order recurrent neural networks. Their work showed that the evolutionary algorithm was better than gradient based methods in terms of training time and guarantee of convergence. Similar work was done in [46] which used multi-objective evolutionary algorithms for evolving the structure and weights of a recurrent network for grammatical inference problems.

In *enforced sub-populations* (ESP), each neuron in the hidden layer corresponds to a separate sub-population that is cooperatively evaluated and coevolved [75]. ESP showed promising results when compared to previous techniques for inverted pole balancing problems with and without velocity information. Hierarchical ESP simultaneously evolves a separate population of entire network with the sub-populations of neurons and keeps a cache of the best combinations [79]. They showed better performance than conventional ESP. *Cooperative synapse neuro-evolution* (CoSyNE) used cooperative coevolution on synapse level encoding where every connection in the network is encoded into a separate sub-population [77]. CoSyNE was shown to outperform neuro-evolution techniques such as ESP, SANE and NEAT [204] for pole balancing problems. However, co-

variant matrix adaptation evolution strategies (CMA-ES) [85] for neuro-evolution [99] outperformed CoSyNE only for the Markovian double-pole balancing problem. *Covariance matrix adaptation neuro-evolution strategy* (CMA-NeuroES) [87] with improved parameter setting has shown to outperform CoSyNE for both Markovian and non-Markovian double pole balancing problems. In summary, CMA-NeuroES has performed fairly well in comparison to CoSyNE for pole balancing problems. These methods were only applied to pole balancing problems, therefore, their performance in other problems such as pattern recognition is still and open issue. A sophisticated version of ESP known as *Evolino* has been used to evolve the long short-term memory networks (LSTM) where it was shown that the framework outperformed gradient based LSTM and learned tasks that were unlearn-able by Echo State Networks [187].

In feedforward networks, *cooperative coevolutionary neural networks* (COVNET) has been proposed for pattern recognition problems [62]. MOBNET uses *multi-objective cooperative coevolution* for assigning fitness for subcomponents in evolving feedforward networks. The roulette wheel section is used and mutation is done using simulated annealing in each of the sub populations. Structural mutation is done by adding and deleting a node or its connections. In this way, the network architecture is also adapted during evolution [61]. The COVNET [62] differs from MOBNET as it does not use multi-objective optimisation for assigning fitness to subcomponents. COVNET and MOBNET have shown promising results in terms of accuracy in classification problems and have shown to learn the problem in simpler or smaller network structures. Cooperative coevolution has also been used in the construction of Bayesian networks for data mining [231], designing neural networks ensembles [63] and cooperative constructive method for designing neural network for pattern classification [65].

2.6.2 Indirect Encoding in Neuro-evolution

In indirect encoding, the genotype specifies rules or some other structure for generating the network. This approach allows large networks to be efficiently represented without using large chromosomes as it grows the network nodes and connections.

Neuro-Evolution of Augmenting Topologies (NEAT) begins evolution with the simplest network topology and adapts nodes and weights together during evolution [204]. Compositional Pattern Producing Networks (CPPN) [202] use several activation functions in the network to build different level of abstraction than conventional methods. CPPN used NEAT in their implementation. Hypercube-based Neuro-Evolution of Augmenting Topologies (HyperNEAT) [203] employed CPPN to learn the lower dimensional space of a substrate or hypercube connection. CPPN determined the weights in the substrate which can further be generalised into very large networks. In [203], CPPN learnt the spatial pattern of a smaller substrate which later generalised and formed a hypercube network of 8 million connections. HyperNEAT is the state-of-art in neuro-evolution and well resembles the structure of biological neural systems. Buk *et. al* substituted the NEAT in CPPN with genetic programming which has shown to achieve faster convergence [21]. Group Adaptive Models Evolution (GAME) also use several optimisation methods in evolving weights, topology and transfer functions [116].

2.6.3 Hybrid HMs for Neuro-Evolution

Hybrid evolutionary approaches have been proposed to simultaneously exploit the strengths and alleviate the weaknesses of evolutionary algorithms with respective gradient descent methods. The most common paradigm is a two-phase combination of evolution with gradient descent optimisation where the evolutionary search algorithm is used in the initial training phase and gradient descent training performs refinement to arrive at the

final solution. Therefore, the evolutionary algorithm performs a global search in hypothesis space and delivers promising regions (solutions) and the gradient descent algorithm in turn performs a local search which improves upon these promising regions. This approach has been shown to reduce the training time and improve the classification performance in a number of applications in feedforward networks [112, 98].

Memetic algorithms have been used for training feedforward networks and a survey of their contribution has been presented in [23].

Hybrid evolutionary methods have been used for training recurrent neural networks. One example is the combination of particle swarm optimisation (PSO) with evolutionary algorithms [22] for training recurrent neural networks on time series problems, where the hybrid method performed better than PSO and EA alone. The hybrid parallel Tabu search with the crossover operation from genetic algorithm was proposed for training recurrent neural networks for the non-linear plant problem and showed better performance than standard Tabu search and genetic algorithm as discussed in [107]. The cellular genetic algorithm (CGA) has been combined with local search in order to train RNN for long-term dependency and the inverted pendulum problem [121]. The combination was efficient only when the local search was implemented using the delta-rule that only adjusted the output layer weights. The CGA and real-time-recurrent learning (RTRL) was combined as CGA-RTRL which took more computational time to converge when compared to CGA alone.

2.6.4 Modularity and Problem Decomposition

The major advantage of the cooperative coevolution in neuro-evolution is that it provides a mechanism for dividing the neural network problem in smaller problems using subcomponents. This gives the flexibility to assign or encode different network weights into different subcomponents or modules according to the network architecture and the type of the learn-

ing problem. In this way, information in the subcomponents are preserved during evolution which is lost in neuro-evolution with conventional evolutionary algorithms due to harmful effects of some reproduction operators such as the crossover. The crossover operator in conventional evolutionary algorithms is designed to make adjustments to all the variables in the individual. In the case of neural networks, the crossover operator will make changes to all or most network weights. The weights associated with neurons in the first layer can be easily interchanged with that of the second layer. Cooperative coevolution ensures that operators such as crossover do not harm the weights of the entire network, and provides the ability to restrict evolution to certain group of weights using the subcomponents. Note that the level of modularity is given by the problem decomposition method in cooperative coevolution.

Problem decomposition is a major concern in neuro-evolution using cooperative coevolution. Some efficient problem decomposition strategies or encoding schemes are discussed as follows. In training *radial-basis* networks, *k-means clustering* has been used to group hidden neurons with similar properties into subcomponents in order to obtain a compact network structure. The method has shown to outperform conventional methods in terms of accuracy [123]. COVNET [62] and MOBNET [61] build subcomponents by encoding input and output connections to the respective hidden neuron. These have been used for training feedforward network architectures. This encoding scheme is similar to that used in ESP which is used for training recurrent networks. In the ESP, recurrent connections also form part of the subcomponent, which is not the case in either of COVNET or MOBNET.

Problem Decomposition in Feedforward Neural Networks

COVNET and MOBNET have been used for training feedforward neworks. These cooperative co-evolutionary methods have used neural level encoding scheme where the neurons in the hidden layer act as the main reference

point for a subcomponent. ESP [76, 78] uses a similar encoding scheme; however, recurrent connections are also present as it has been used mainly for training recurrent networks.

Henceforth, the encoding scheme used in COVNET, MOBNET and ESP are referred to as "CME" due to their similarities. This is done by taking the first letters from each abbreviation. Figure 2.5 shows the schematic of the interconnected input and output links to a hidden neuron. It is assumed that the network has one hidden layer only. The number of hidden neurons is equal to the number of subcomponents. In this mapping, all sub-populations have the same size for the entire framework.

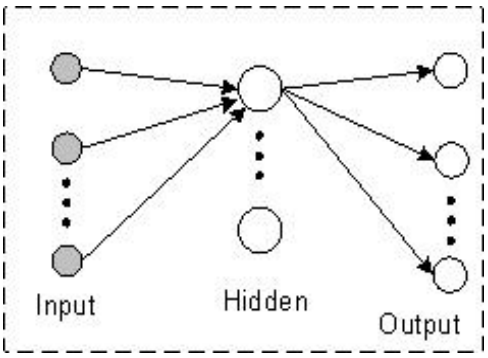


Figure 2.5: The CME encoding scheme summarised from [78, 62, 61] is used for comparison in the experiments.

Problem Decomposition in Recurrent Neural Networks

Problem decomposition in cooperation coevolution for recurrent networks is as follows. There have been two major problem decomposition methods for training recurrent neural networks. The first method proposes a *neuron level* encoding where each neuron in the hidden layer is used as a major reference point for each sub-population. Therefore, the number of hidden neurons is equal to the number of sub-populations [78]. This encoding is used in ESP where a particular neuron h_i in the hidden layer would

encode the following weight links in its sub-population:

1. The weights links connecting from the input layer to h_i
2. The weight links connecting from h_i to each context neurons
3. The weight links connected from h_i to each output layer
4. The bias associated with h_i

In this method, all individual sub-populations have the same size for the entire framework. This encoding is shown in Figure 2.6.

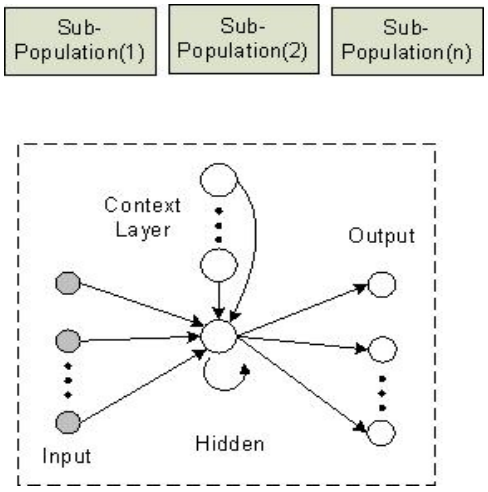


Figure 2.6: The ESP encoding scheme taken from [78].

The second problem decomposition method is based on *synapse level* encoding as given in CoSyNE [77]. This encoding scheme decomposes the network into its lowest level; where each weight link (synapse) in the network is part of a single sub-population. CoSyNE has shown to outperform ESP for training recurrent networks. The difference in performance in the ESP and CoSyNE for the same problem shows that problem decomposition is an important feature of cooperative co-evolutionary recurrent networks.

2.6.5 Fitness Evaluation of the Sub-populations

A major concern in cooperative coevolution is the cooperative evaluation of each individual in every sub-population. There are two main phases of evolution in the cooperative coevolution framework. The first is the *initialisation phase* and second is the *evolution phase*. In the initialisation phase, the subcomponents are seeded with random values and evaluated. In the evolution phase, the subcomponents are evolved in a round-robin fashion until a cycle is completed. The initialisation phase is shown in Step 2 of Algorithm 2.

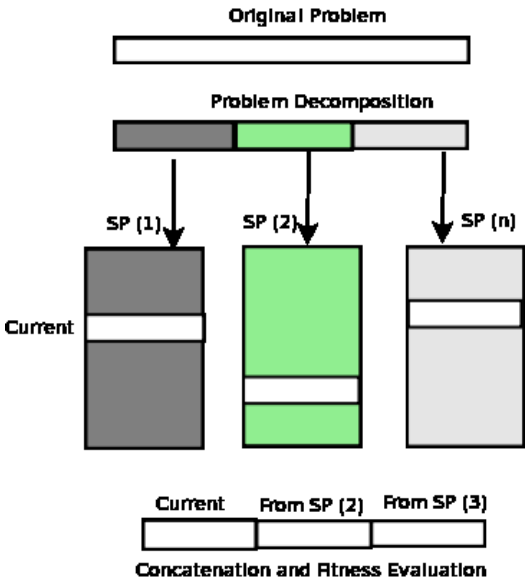


Figure 2.7: The current individual whose fitness has to be evaluated is joined with arbitrary individuals from the rest of the sub-populations in the initialization phase. In the evolution phase, the best individuals from the rest of the sub-populations are chosen. The current individual is then concatenated with the chosen individuals. The concatenated individual is encoded into the neural network by the given cooperative coevolution encoding scheme. The fitness is then evaluated and assigned to the current individual.

In order to evaluate the ith individual of the kth sub-population, arbitrary individuals from the rest of the sub-populations are selected and

combined with the chosen individual and cooperatively evaluated. Arbitrary individuals are selected because in the initialization stage the individuals from the rest of the sub-populations do not have a known fitness.

In the evolution phase, cooperative evaluation shown in Step 2 (ii), is done by concatenating the chosen individual from a sub-population k with the fittest individuals from the remaining sub-populations. After the initialization phase, an estimate of the best individuals from the rest of sub-populations can easily be found through ranking according to fitness. The concatenated individual is encoded into the neural network and the fitness is calculated through the training error. The fitness evaluation of individuals in each sub-population is further shown in Figure 2.7. In this way, the fitness of each subcomponent in the network is evaluated. Note that the fitness of the whole network is assigned to each individual, even though it represents a part of the solution. The contribution of the individual will affect the functionality of the entire neural network as all weights and neurons contribute to its output. A similar approach is taken in [168].

Other problem decomposition methods such as CME, ESP and CoSyNE evaluate the fitness of each individual differently. For instance, CoSyNE uses n trial runs and takes the average fitness over trials only in the initialisation phase. In this thesis, the same method for sub-population initialisation and evaluation during evolution will be used in different encoding schemes order to show a fair comparison.

2.6.6 Evaluation of Cooperative Neuro-evolution

Training neural networks through any evolution computation method is computationally expensive in terms of the optimisation time given by number of function evaluations. The computational cost of a single function evaluation depends on the type and size of the network architecture and the size of the training data. Moreover, the number of features in the training data also adds to the computational cost. In any optimisation problem,

there are two main attributes, which are the accuracy of the solution and the elapsed time. In neural network training, the accuracy of the solution is either the overall error of the neural network or the performance of a validation set. The overall error of the network is usually given by mean-squared-error or sum-squared-error.

A good measure of time is the number of functions evaluation it takes for the learning algorithm to converge to a desired solution. It is easy to compare the performance in terms of number of function evaluations with other algorithms, which have been executed on different machines.

The ability of an algorithm to give a solution within a given time is also an important measure. The guarantee for a solution can be measured with a success rate over a specified number of experimental runs. A successful run is when the algorithm converges to the desired solution before reaching the maximum time in terms of function evaluations. The average function evaluations with the number of successful run can be used in comparison to other methods. The least number of mean function evaluation with maximum success is desired. Therefore, this thesis will use function evaluation and success rate as the main measure to evaluate the performance of the proposed algorithms. The success rate reflects on robustness of evolutionary algorithms.

Evolutionary algorithms like other methods tend to deteriorate in performance when the size of the problem increases. Scalability is one of the current challenges for evolutionary algorithms. Scalability ensures that the algorithm can perform when the problem size is sufficiently increased; this is usually when the training data or the number of neurons of the neural network increases. In this thesis, we will also check the performance of the algorithm when the problem becomes larger. The number of variables significantly increases when the size of the hidden neuron increases. The connections associated with recurrent state neurons also increases the overall problem size. Therefore, scalability will be tested by evaluating the proposed methods on different number of hidden neurons on feedforward

and recurrent neural networks.

2.6.7 Chapter Summary

This chapter has given the background on feedforward and recurrent neural networks, evolutionary algorithms, Memetic algorithms and neuro-evolution. It has also given background of cooperative coevolution, non-separability, modularity and cooperative neuro-evolution. It has provided a review of the recent developments in these areas and outlined their strengths and limitations. These lead to motivations for the work carried out in the rest of the thesis.

The G3-PCX evolutionary algorithm has limitations in multi-modal optimisation problems and this limitation is mainly due to the intensive local search feature of the parent-centric crossover operator. The G3-PCX needs a fairly large population even on for small problems. This shows a high level of diversity is needed for parent-centric crossover. This can become a problem for training neural networks as they can be seen as multi-modal optimisation problems. Cooperative coevolution maintains high diversity through the use of sub-populations where recombination is restricted within a sub-population. Cooperative coevolution can help the G3-PCX evolutionary algorithm by providing more diversity and this can be beneficial in neural network training.

The issue of separability has been explored in the literature for global optimisation problems; however, there has not been any work about in issue of separability in the case of training neural networks. Problem decomposition is central to cooperative coevolution and problem decomposition should consider the nature of the problem in terms of separability and interacting variables. It is important to analysis how the variables interact with each other in a neural network training process and this can help is building efficient problem decomposition methods for cooperative coevolution.

Furthermore, the established synapse level problem decomposition (CoSyNE) and neuron level problem decomposition (ESP) have not considered the interacting variables in the neural network. CoSyNE views the neural network as a separable problem while ESP groups the variables with reference to a hidden neuron. The ESP approach makes it difficult to be generalized to a neural network with more than one hidden layer. A good problem decomposition method has to consider the interacting variables in the neural network and at the same time its architectural properties so that it can be generalized to neural network with several hidden layers.

In the cooperative coevolution literature, the use of local search has not been fully explored. There are several sub-populations involved in cooperative coevolution and local search can further add to the computational costs, however, local search can also improve the quality of the solution. Therefore, the incorporation of local search in cooperative coevolution can further improve its performance.

In global optimization methods, several techniques for non-separable problems have been used that mainly include 1) DECC-G that employs random grouping of interacting variable and its variant that used delta grouping, 2) MLCC that adapts the size of the subpopulation at different levels of evolution and 3) DECC-DML that employs delta grouping with MLCC. These methods improved performance for global optimization problems; however, the application of these ideas has not been explored for cooperative coevolution of neural networks. A major problem of the MLCC approach is that it merges the sub-populations based on a predefined set which does not cater for the interacting variables between the sub-populations. In order to make adaption of problem decomposition in neural networks, the interacting variables must be considered and the sub-populations should be merged using some of the established problem decomposition methods.

Based on the above motivations, new techniques in cooperative coevolution can be developed for training feedforward and recurrent networks

to solve interesting problems that include pattern classification, grammatical inference and time series prediction.

Training neural networks which large datasets is a computational expensive task, therefore, it is important to measure the optimisation time in terms of the number of function evaluations. The optimisation time, scalability and robustness are the important attributes of a neural network training algorithm. Scalability will be tested by evaluating the algorithms performance with different number of hidden neurons in the network. A robust algorithm will ensure that it delivers a solution within a specified time-frame given different initial positions in weight space. These attributes will be tested in the rest of the thesis in order to demonstrate the efficiency of the training algorithms.

The remaining chapters deal with problem decomposition, local search and adaptation of problem decomposition methods in cooperative neuroevolution. These methods are then applied for solving difficult chaotic time series problems.

Chapter 3

Problem Decomposition in Cooperative Neuro-evolution

The literature in the previous chapter has presented cooperative neuro-evolution and how it decomposes a neural network problem into subcomponents and evolves them. This chapter proposes a problem decomposition method based on the functional properties of a neuron. The proposed encoding scheme is used for training feedforward networks on pattern classification problems, and recurrent neural networks on grammatical inference problems. The chapter begins with a summary of the problem decomposition methods presented in the literature. Then it presents the motivations for the new method. The details of the proposed method are then given for feedforward and recurrent networks and the simulation is presented with an analysis of the results and discussion.

3.1 Introduction

Problem decomposition has been a major area of study in using cooperative coevolution for neuro-evolution. The major problem decomposition methodologies are those based on the *neuron level* and *synapse level*. The neuron level encoding scheme uses each neuron in the hidden layer as

the main reference point for the respective subcomponents [62, 61, 76, 78]. In synapse level problem decomposition, each weight or synapse in the network forms a subcomponent [77]. Therefore, the number of subcomponents depends on the number of weights and biases. The use of CoSyNE in training neural networks for pattern classification has not been explored.

In the literature, there has not been much study on how to encode interacting variables into separable subcomponents for the specific case of neuro-evolution. Most of the study has been on large-scale function optimization problems, which is not directly applicable to neuro-evolution. In the case of neuro-evolution, the problem decomposition method must take into account the architecture of the neural network in order to group interacting variables into the same subcomponents. It is important to analyse how the different weight links interact with each other. This forms the basis of the proposed problem decomposition method in this chapter.

In this chapter, the problem decomposition methods from the literature are used for training feedforward and recurrent neural networks for solving pattern classification and grammatical inference problems, respectively. Their performance is compared on benchmark datasets, and an encoding scheme is proposed that falls in the group of neuron-level encodings. The new encoding scheme is motivated by the analysis of the *degree of non-separability*, which reflects on the interacting variables in the neural network. The proposed scheme is called neuron-based sub-population (NSP). The goal of NSP is to reduce the optimization time and achieve better guarantee for convergence. The investigation proceeds with the optimal *depth of search* required for the respective problem decomposition methods. The depth of search can determine whether the encoding schemes have been able to group the interacting variables into the same subcomponents. If the interacting variables have been grouped efficiently, a deep greedy search for the sub-populations would be possible, implying that the problem has been efficiently broken down into subcomponents. The optimal depth of search is used to further evaluate the performance

of the respective methods with different numbers of hidden neurons. The difference in the number of hidden neurons affects the total number of variables (the problem dimensionality). The problem dimensionality is an important measure that reflects on the issue of scalability in evolutionary algorithms.

In order to demonstrate the effectiveness of NSP in training feedforward networks, benchmark problems from the UCI machine learning repository [6] are used and a comparison is done with the methods from the literature. In the case of recurrent neural networks, a specific grammatical inference problem from [15] is used with the Tomita grammar [213, 26].

The neural network optimization time in terms of number of function evaluations, and the success rate, are considered to be the main performance measures for the method presented in this study. The success rate determines how well the particular algorithm can guarantee a solution within a specified time. It reflects on robustness of the evolutionary algorithm. A run is considered successful if a desired solution is found before the maximum time is reached in n independent experimental runs with different random initialisations. The desired solution for neural network training is specified by a predefined maximum network error or minimum classification performance, depending on the type of the problem.

3.1.1 Preliminaries and Motivation

Degree of Non-Separability

This section examines the interaction between the variables (synapses) in a feedforward neural network. We revisit the definition of separability from Definition 3 in Section 2.5.2 of the literature review. Separable functions are those where the minimization of a variable independently contribute to the global minimization of the whole function. This does not hold for non-separable function where the variables interact with each other and the independent minimisation of a variable may hinder the optimisation

process. In cooperative coevolution, it is important to group interacting variables within a subcomponent.

A measure for the *degree of non-separability* is established by monitoring the gradient of the rest of the variables when a single variable is perturbed several times. In a fully separable problem, there will be no change in the sign of the gradient of the rest of the variables when a single variable is perturbed.

The degree of non-separability is used to understand the interactions among variables in the case of neural networks. The neural network in Figure 3.1 contains two sigmoid neurons in the hidden layer (h_1 and h_2) and output layer (y). The set of synapses W, is given by $W = [a, b, c, d, e, g]$. i and j are the bias of h_1 and h_2, respectively. x_1 and x_2 are inputs and E is the error given for a single instance. y is the actual output of the network. t will be used as the target output. Equation 3.1 gives further details of Figure 3.1.

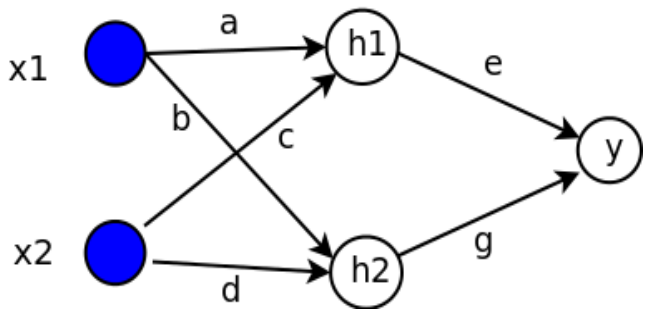


Figure 3.1: The simple feedforward neural network used for analysis.

$$
\begin{aligned}
\phi_1 &= x_1 a + x_2 c + i \\
\phi_2 &= x_1 b + x_2 d + j \\
h_1 &= F(\phi_1) \\
h_2 &= F(\phi_2) \\
y &= F(F(\phi_1)e + F(\phi_2)g) \\
E &= \frac{1}{2}(t-y)^2 \\
\text{where } \; F(x) &= \frac{1}{1+e^{-x}}
\end{aligned}
\tag{3.1}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial a} &= e\ x_1\ (t-y)\ F'(\phi_1)\ F'[F(e\ F(\phi_1) + g\ F(\phi_2))] \\
\frac{\partial E}{\partial b} &= g\ x_1\ (t-y)\ F'(\phi_2)\ F'[F(e\ F(\phi_1) + g\ F(\phi_2))] \\
\frac{\partial E}{\partial c} &= e\ x_2\ (t-y)\ F'(\phi_1)\ F'[F(e\ F(\phi_1) + g\ F(\phi_2))] \\
\frac{\partial E}{\partial d} &= g\ x_2\ (t-y)\ F'(\phi_2)\ F'[F(e\ F(\phi_1) + g - k\ F(\phi_2))] \\
\frac{\partial E}{\partial e} &= F(\phi_1)\ (t-y)\ F'[F(e\ F(\phi_1) + g\ F(\phi_2))] \\
\frac{\partial E}{\partial g} &= F(\phi_2)\ F'(t-y)\ [F(e\ F(\phi_1) + g\ F(\phi_2))]
\end{aligned}
\tag{3.2}
$$

Algorithm 3 shows how the degree of non-separability is computed for a feedforward neural network. A total of m Monte Carlo trials are done where vector W is intialised in the range of [-5,5]. In each trial, a single synapse in the neural network is perturbed by multiplying the variable by -1 and the rest of the synapses are frozen. The gradient $\frac{\partial E}{\partial W}$, as shown in Equation 3.2, for the frozen weights are computed. For instance, when synapse a is perturbed, the behaviour on the rest of the synapses, which include weights and biases $W = [i, j, b, c, d, e, g]$, are observed. The change in the sign of the gradient for a particular frozen weight indicates that there is interaction. Therefore, whenever there is a sign change in the gradient, the count of the number of interactions of the particular synapse is

Alg. 3 The Degree of Non-Separability in Feedforward Neural Networks

1. Create vector W of size n where n is total number of weights and biases
2. Create vector of gradients P and Q of size n
3. Create a matrix C of size $n \times n$.

for each Monte-Carlo trial **do**
  Initialise the vector W with random values in a range
  for each i until n is reached **do**
    i. Compute all the *previous gradients* ($P = \frac{\partial E}{\partial W}$), as shown in Equation 3.2
    ii. Perturb $W[i]$, flip the sign ($W[i] = -1 * W[i]$)
    iii. Compute all the *current gradients* ($Q = \frac{\partial E}{\partial W}$), as shown in Equation 3.2
    iv. Compare the *previous and current gradients*
    for each j until n is reached **do**
      Compare $P[j]$ and $Q[j]$ and increment $C[i][j]$ if the sign of the two gradients change
    end for
  end for
end for

incremented. This is the measure for the degree of non-separability.

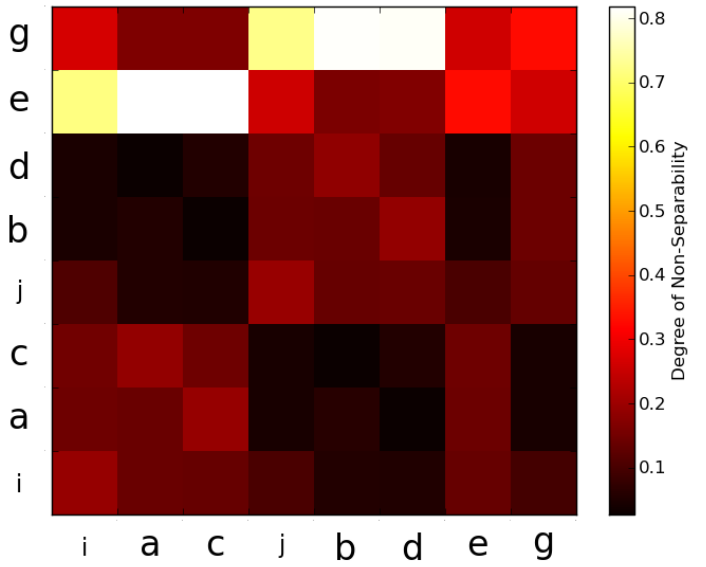
A total of 10000 experiments with different values of weights and biases are done and results are averaged and shown as heat-maps in Figures 3.2 - 3.3.

The results in Figures 3.2 and 3.3 show independent Monte-Carlo trials of 4 different combinations of input $x1$ and $x2$ ([0,0], [0,1], [1,0] and [1,1]). They show how the rest of the synapses in the neural network interact with each other when a synapse is perturbed independently in the Monte-Carlo trials.

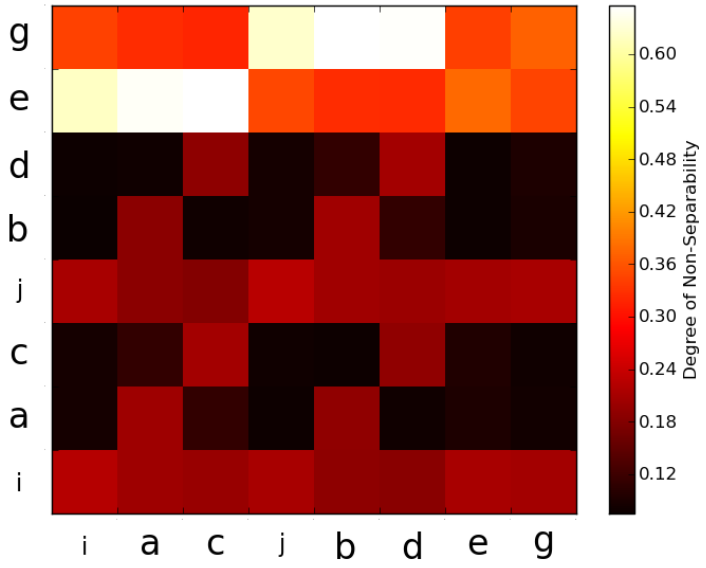
The target output for each input instance is heuristically generated to depict different stages in learning. In the beginning stages of learning, this is done by generating a target output which has a higher difference when compared to the actual output given by the neural network. Towards the ending stages of the learning process, the target is generated such that the difference with the actual network output is small and will produce a low root-mean-squared-error (RMSE), depicting that the neural network weights have learnt the problem. The RMSE will indicate the learning process of the 4 different stages of evolution. Figure 3.2 shows that in the beginning of the learning process, there is little interaction between the weights e and g (Stage 1). The weights in the input-hidden layer interact with the weights in the hidden-output layer. In Stage 2, the interaction between weights e and g become higher. Figure 3.3 shows that the interaction between e and g remain high during the later stages of learning. In Stage 3, the interactions between $W = a, b, c, d$ grow and more interaction is seen among weights e and g. Stage 4 shows the end of the learning process and there is more interaction with e and g and the rest of the weights.

In most of the learning process, the interaction between e and g grows and hence it is reasonable to group these weights together into a separate subcomponent.

Note that the heat-maps in both of these figures do not have symmetry. This is due to the way the gradients are calculated and the sigmoid neuron

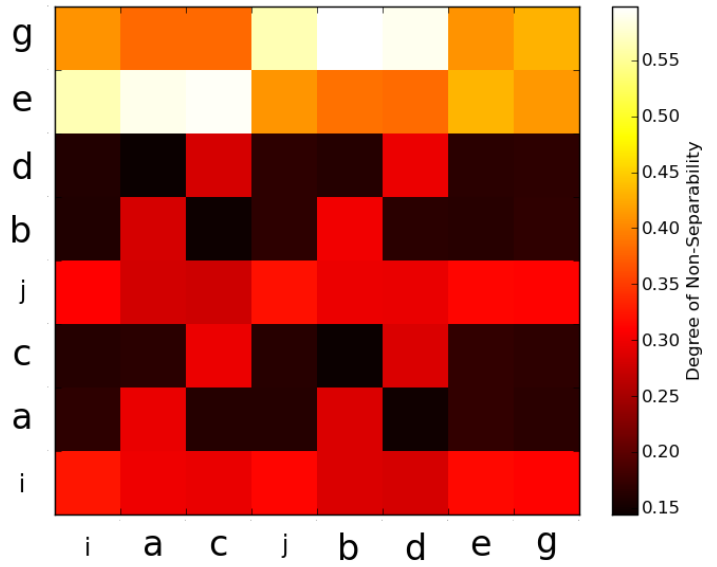


(a) Stage 1: RMSE is 0.381

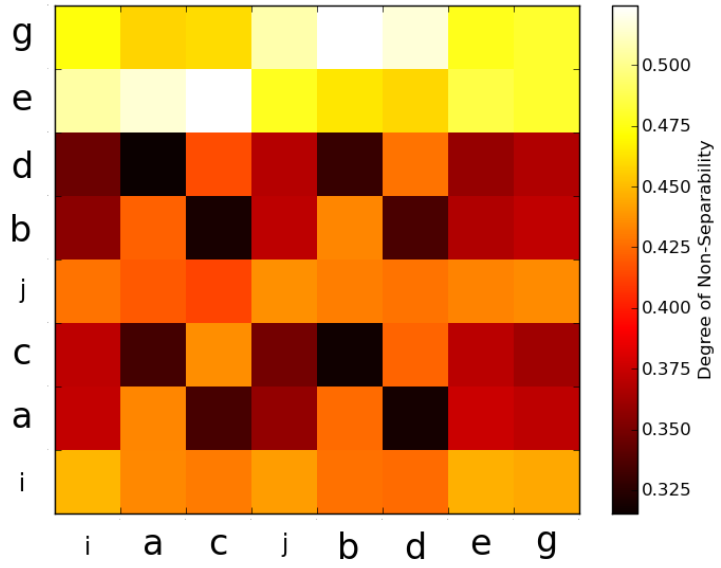


(b) Stage 2: RMSE is 0.095

Figure 3.2: The level of interaction between the synapses at the beginning of the evolutionary process



(a) Stage 3: RMSE is 0.047



(b) Stage 4: RMSE is 0.009

Figure 3.3: The level of interaction between the synapses towards the end of the evolutionary process

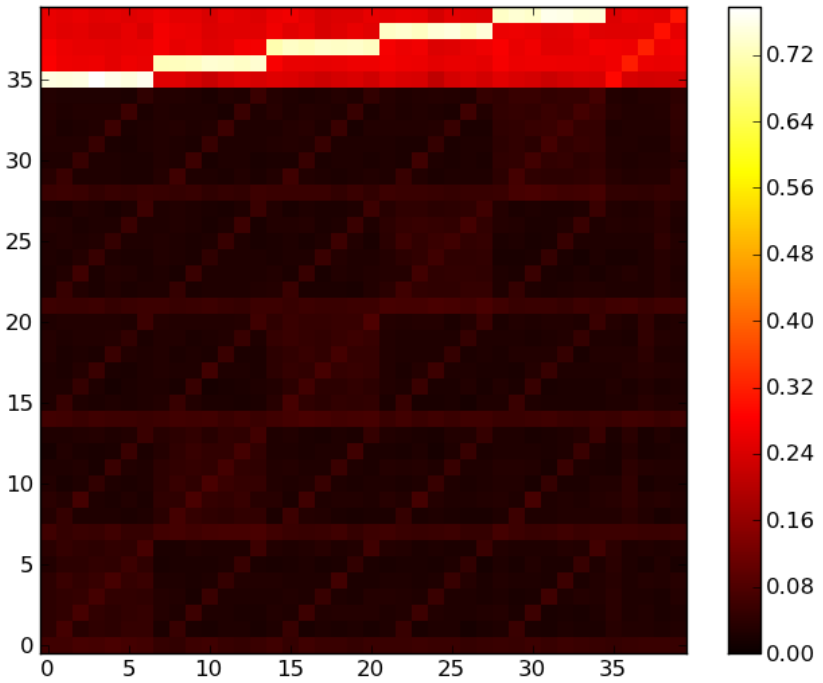


Figure 3.4: The level of interaction between the synapses at the beginning of the evolutionary process on a 6 dimension problem. There are 6 inputs (plus a bias) on a neural network with 5 hidden units and one output.

in the hidden layer makes a difference in the gradients. We can see that a interacts with e, but it is not the other way around i.e e does not interact with a; therefore, they are not required to be grouped. Due to this asymmetry, it is reasonable to separate the groups of the hidden-output layer weights from the input-hidden layer weights.

Note that Figures 3.2 - 3.3 actually indicates that the degree of non-separability for a variable/weight can be measured with respect to itself. This does not have any meaning in real context; however, these values were obtained by the way experiments have been set up. The diagonal values vary in the different layers as indicated by the colour of the heat-maps in the input-hidden and hidden-output layers. This is due to the level of interactions involved between the weights among these layers.

We observe that there are two options for grouping the weights. *Option 1* groups the weights as follows; $[a, c, i]$ are grouped together, $[b, d, j]$ are grouped together, and finally $[e, g]$ are grouped together.

Another possibility is *Option 2* where the weights are grouped as follows; $[e, g]$ are be grouped together as in Option 1. In the input-hidden layer weights, $[a, b, i]$ are grouped together and $[c, d, j]$ are grouped together. This option seems to be more suitable according to the interaction in the input-hidden layer weights during the later stages of learning (Stages 2-4).

Both of the options will have more subcomponents than CME and can be generalised to a neural network with more than one hidden layer.

However, Option 1 will be used in this thesis. With Option 1, the number of subcomponents depends mainly on the number of hidden neurons while in Option 2, it depends on the number of input units. In the experiments, different number of hidden neurons will be used and the effect of diversity can be better evaluated with Option 1. This will be done using different number of hidden neurons in the neural network in Section 3.2. All the problems in this thesis will have a fixed number of input units and different number of hidden units will be used to measure the performance

of the proposed methods.

Figure 3.4 shows the degree of non-separability on a 6 dimensional problem in the early stage of evolution. There are 6 inputs (plus a bias) on a neural network with 5 hidden units and one output. Indices 0 to 35 show the interaction among the input-hidden later weights. The results in this heat-map indicates that the degree of non-separability is similar for a larger network architecture.

The heat-maps in general have shown that the degree of non-separability increases during the later stages in evolution as the problem is being learnt. It is also reasonable to merge or group the subcomponents together during the later stages of evolution. Synapse level problem decomposition can be used in the beginning of evolution as lesser interactions are present in Stage 1 and 2. The subcomponents can be merged in the later stages of evolution as shown in Stage 3 and 4.

Diversity in Cooperative Coevolution

An important feature of cooperative coevolution is that it provides higher level of diversity when compared to an evolutionary algorithm with the same number of individuals. Potter *et. al* [168] have shown that a large number subcomponents impact the performance of cooperative coevolution as more diversity is achieved when more sub-populations are used. Diversity directly relates to global search and local convergence; if the individuals in a population are not sufficiently diverse, local convergence is possible. The sub-populations of cooperative coevolution are evolved in isolation and are more diverse than a single population based evolutionary algorithm. Consider that there are P sub-populations of N individuals each, there will be total $\binom{P}{N}$ diverse solutions when compared to only N diverse solutions in a conventional evolutionary algorithm.

It is important to balance diversity and interacting variables during problem decomposition. More diversity is needed during the initial search where global search is needed and local search in needed during the final

stages of the search. The degree of interaction as shown in Section 3.1.1 changes during the later stages of learning and efficient problem decomposition has to take these into consideration. These will be further discussed in Chapter 5.

3.2 Neuron Based Sub-population (NSP)

A major limitation of the encoding scheme given in CME given in Section 2.6.4 is that it cannot be extended to a neural network with more than one hidden layer. The same problem also exists for ESP shown in Section 2.6.4 for training RNNs. This is because the hidden neuron of a single layer acts as a major reference point and contains outgoing neurons in the subcomponents. We need a problem decomposition method that can be easily extended to a network with more hidden layers and at the same time it should efficiently group the interacting variables in a separate subcomponent.

3.2.1 Feedforward Neural Networks

Synapse level encoding in neural networks provides the most diversity and ensures global search. However, it will work well if there are less interacting variables in the neural network. CME encoding has less diversity and gives more emphasis on interacting variables. The problem in cooperative neuro-evolution is to balance diversity and interacting variables. A problem of CME encoding is that it cannot be generalised to a neural network with more than one hidden layer. If the CME encoding is broken down, it can be generalised to more than one hidden layer and also achieve more diversity. The new encoding scheme should also group interacting variables better than Synapse level encoding which does not have any grouping but works only with diversity. The results in Section 3.1.1 have shown that it is reasonable to break the CME encoding.

The new problem decomposition method is called neuron based sub-population (NSP). NSP breaks down the CME encoding scheme into a lower level. Each subcomponent in NSP consists of incoming connections associated with neurons in the hidden and output layers. NSP employs a single subcomponent for each neuron that groups interacting variables (synapses) that are connected to the neuron. Therefore, each subcomponent for a layer is composed as follows:

1. For a given neuron i in the $hidden$ layer, the *hidden layer subcomponents* consists of all synapses connected from $input$ layer to neuron i. The bias of i is also included.

2. For a given neuron j in the $output$ layer, the *output layer subcomponents* consists of all synapses connected from $hidden$ layer to neuron j. The bias of j is also included.

Figure 3.5 shows a detailed diagram of the NSP encoding. Note that this encoding schemes can be easily extended for a network with more than a single hidden layer. The NSP encoding used for training feedforward networks is summarised in Algorithm 4. In effect, each neuron in the hidden and output layer acts as a reference point to its subcomponents given as sub-populations.

In Algorithm 4, the problem is decomposed into k subcomponents, where k is equal to the number of hidden neurons, plus the number of output neurons. Each sub-population contains all the weight links from the previous layer connecting to a particular neuron. A *cycle* is completed when all the sub-populations are evolved in a round-robin fashion for a fixed number of generations. The algorithm halts if the termination condition is satisfied. The termination condition is when the network correctly classifies a given percentage of the training data or when the maximum training time has been reached.

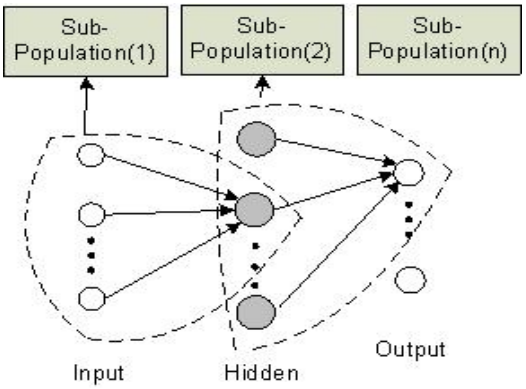


Figure 3.5: The NSP encoding scheme for feedforward networks [27]. Each neuron in the hidden and output layer acts as a reference point to its sub-components. The same encoding is used in the rest of the neurons in the hidden and output layer. Note that only one hidden layer is used. NSP can also be used for more than one hidden layer.

Alg. 4 The NSP for Training Feedforward Networks

Step 1. Decompose the problem into subcomponents according to the number of Hidden and Output neurons. The total number of subcomponents is the number of hidden plus output neurons.
Step 2. Encode each subcomponent in a sub-population
Step 3. Initialise and cooperatively evaluate each sub-population
for each Cycle until termination **do**
  for each Subpopulation **do**
    for n Generations **do**
      i) Select and create new offspring
      ii) Cooperatively evaluate the new offspring
      iii) Add new offspring's to the sub-population
    end for
  end for
end for

3.2.2 Recurrent Neural Networks

In this section, the NSP given in Section 3.2.1 for feeedforward networks is adapted for recurrent neural networks. Each subcomponent in the NSP consists of incoming weight links associated with a neuron in the hidden, state (recurrent), and output layer.

In the NSP problem decomposition method for recurrent networks, each neuron in the hidden and output layer is a reference point for a subcomponent. Each hidden neuron also acts as a reference point for the context weight links connected to it. Therefore, each subcomponent for a layer is composed as follows:

1. For a given neuron i in the $hidden(t)$ layer, the *hidden layer subcomponents* consists of all synapses connected from $input(t)$ layer to neuron i. The bias of i is also included.

2. For a given neuron j in the $hidden(t)$ layer, the *state (recurrent) layer subcomponents* consists of all synapses connected from the $hidden(t-1)$ layer to neuron j.

3. For a given neuron k in the $output(t)$ layer, the *output layer subcomponents* consists of all synapses connected from the $hidden(t)$ layer to neuron k. The bias of k is also included.

where t is time and $hidden(t-1)$ is the layer representing the state or recurrent neurons.

The NSP encoding for training RNN is summarised in Algorithm 5. Figure 3.6 shows a detailed diagram of the NSP encoding.

In Algorithm 5, the recurrent network is decomposed in k subcomponents where k consists of the number of hidden neurons, the number of state neurons, and the number of output neurons. Each subcomponent contains all the weight links from the previous layer connecting to a particular neuron. The algorithm is identical to Algorithm 4, with the addition of subcomponents for the state neurons.

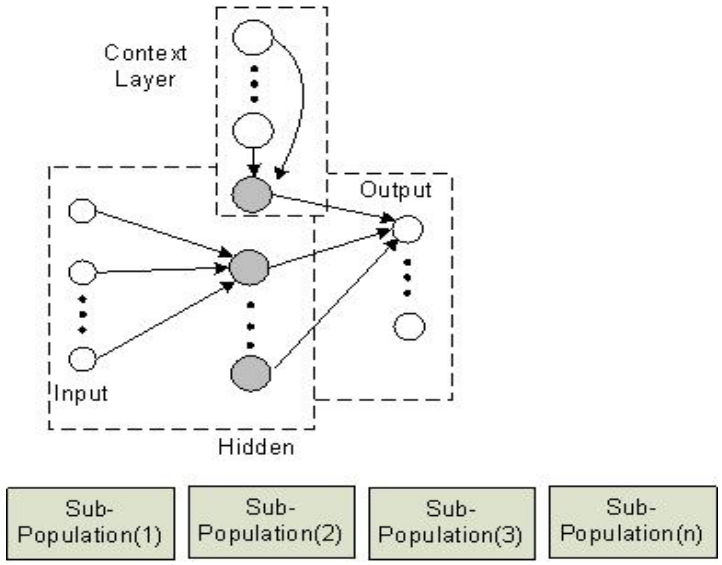


Figure 3.6: The NSP encoding scheme for recurrent networks. Each neuron in the hidden and output layer acts as a reference point to its subcomponents. The subcomponents for the state neurons are also shown. The same method is used in the rest of the neurons in the hidden and output layer. Note that only one hidden layer is used in this case; however, additional hidden layers can also be used.

Alg. 5 The NSP for Training Recurrent Neural Networks

Step 1. Decompose the problem into subcomponents according to the number of Hidden, State, and Output neurons
Step 2. Encode each subcomponent into its corresponding sub-populations
Step 3. Initialise and cooperatively evaluate each sub-population
for each Cycle until termination **do**
  for each Subpopulation **do**
    for n Generations **do**
      i) Select and create new offspring
      ii) Cooperatively evaluate the new offspring
      iii) Add new offspring to the sub-population
    end for
  end for
end for

3.3 Simulations

This section presents an experimental study of NSP and compares it with CoSyNE and CME as discussed in the previous sections. In this study, the G3-PCX [42] evolutionary algorithm was used in each method. This evolutionary algorithm has also been used for training feedforward neural networks in the past [23]. NSP is used for training feedforward and recurrent neural networks.

The G3-PCX algorithm uses a population size of 100, a pool size of 2 offspring and a family size of 2 parents in all cases. This set-up has been used in [42] and has shown good results for general optimisation problems. The synaptic weights within the sub-populations were initialised with random real numbers in the range of [-5, 5] in all the experiments.

3.3.1 Feedforward Neural Networks

Real-World Problems and Neural Network Configuration

This section gives an overview of the datasets obtained from the UCI machine learning repository which include Iris, Wine, Breast-Cancer and Heart-Disease [6]. Table A.1 of the Appendix shows the data information which leads to the neural network configuration for all the experiments. The maximum training time given by the number of function evaluations in all the problems is fixed 100000. The number of hidden neurons is fixed for evaluating the depth of search. 4 neurons are used in the Iris and Wine problems, Breast-Cancer and Heart-Disease problems use 5 and 7 neurons, respectively.

Evaluation During Initialisation

Note that the results in this section do not include the number of function evaluations done in the initialisation stage for each method. The relationship between the number of subcomponents and function evaluations for

each method can be evaluated as follows.
1) CME = $hidden \times P$

2) NSP = $(hidden + output) \times P$

3) CoSyNE = $((input \times hidden) + (hidden \times output) + hidden + output) \times P$

where P is the size of the population, *hidden* = number of hidden neurons, and *output* = number of output neurons. The number of function evaluations in the initialisation stage for each problem decomposition method is given by:

FuncEval = Number of Subcomponents $\times P$

The number of hidden neurons in each method directly influences the number of subcomponents. This further influences the number of function evaluations. Therefore, the number of function evaluations used in the initialisation stage differs for each method and is an important measure for their evaluation. The NSP uses fewer function evaluations during initialisation when compared to CoSyNE, but greater than CME. In the following subsections, the performance of NSP during evolution is evaluated.

Depth of Search in the Sub-populations

Each sub-population is evolved in a round-robin fashion for a fixed number of generations in NSP as shown in Algorithm 4. The study begins by determining the optimal number of generations needed for the sub-population in each method. Note that all the sub-populations are meant to evolve for the same depth of search.

The results shown in Figures 3.7 to 3.10 report the performance of the

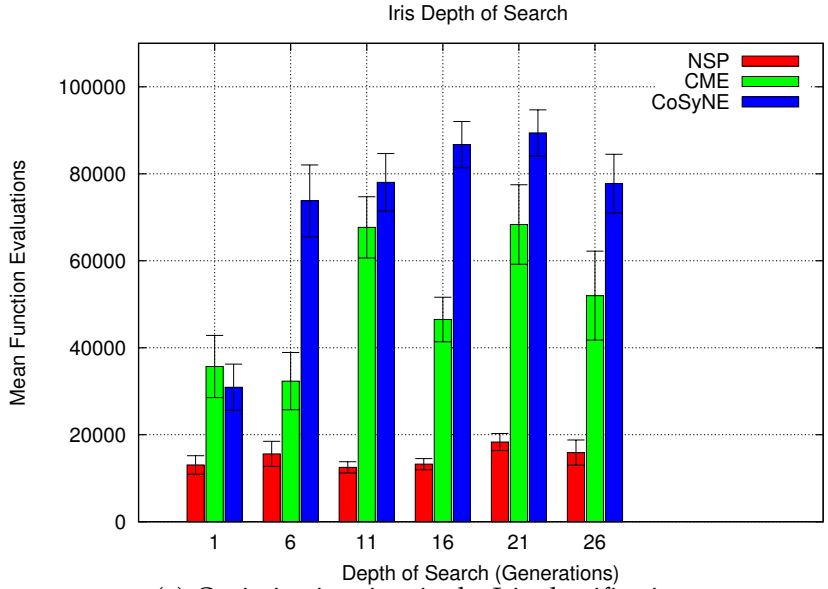
respective methods in terms of number of function evaluations and the success rate out of 30 independent runs. The depth of search 'Depth' given by number of generations for all sub-populations is shown. The mean of the number of function evaluations is shown with 95 % confidence interval as error bars in the histograms.

The results for the Iris classification problem shown in Figure 3.7 shows that NSP outperforms the other methods in terms of the number of function evaluations and the success rate. CoSyNE takes the most training time and gives a poor success rate. CME performs slightly better than CoSyNE but tends to be weaker than NSP. NSP is the best choice for this problem under the given conditions. The depth of search in NSP does not make a major difference in its performance.

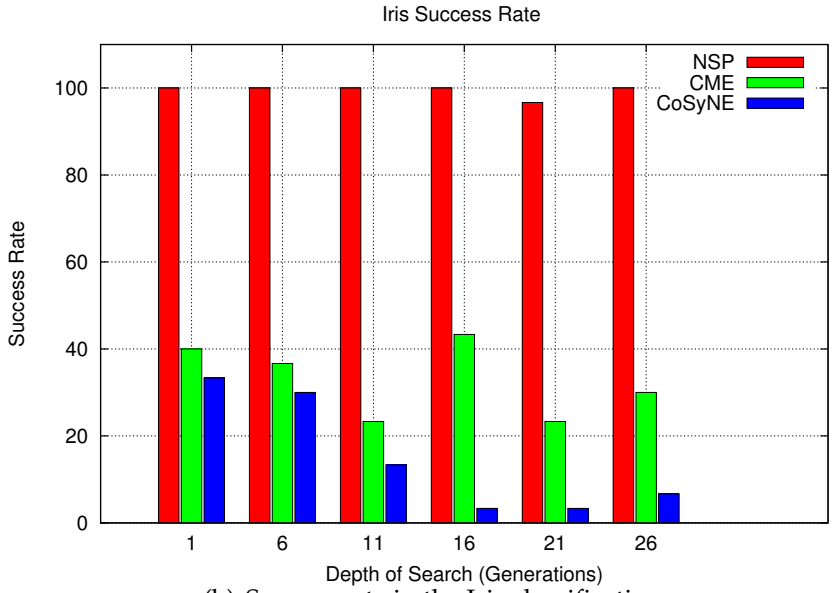
In the results for the Wine classification problem shown in Figure 3.8, NSP gives the best performance. The depth of search in NSP does not play a major role. The depth of search impacts the performance of CME and mostly, CoSyNE. CoSyNE shows good performance for the depth of 1 generation only. The best results are given by NSP in terms of least training time and the best success rate.

The results from the Heart-Disease classification problem shown in Figure 3.9 shows that NSP outperforms CME and CoSyNE. The CoSyNE method delivered a solution only with the depth of 1 generation. It performs poorly in comparison to CME and NSP. The depth of search in NSP does not make a significant difference. Similar performance is shown for the Breast-Cancer classification problem as shown in Figure 3.10, where NSP outperforms other methods. The depth of search for NSP and CME does not play a significant role in their performance. This suggests that NSP has been able to group the interacting variables successfully in the different sub-populations.

The results in general show that a lower depth of search used for a subcomponent is efficient, especially for CoSyNE. The depth of search for NSP does not show a significant difference for the interval of [1, 26]. This

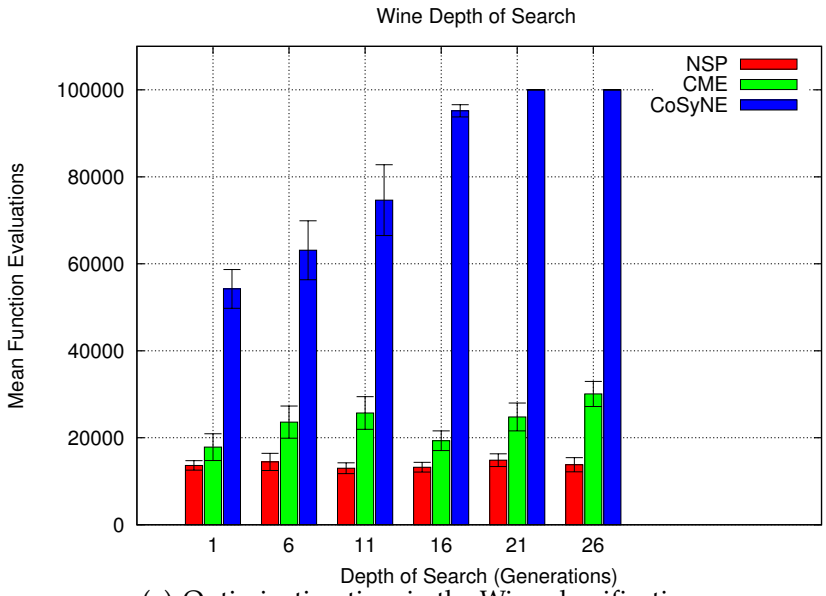


(a) Optimization time in the Iris classification

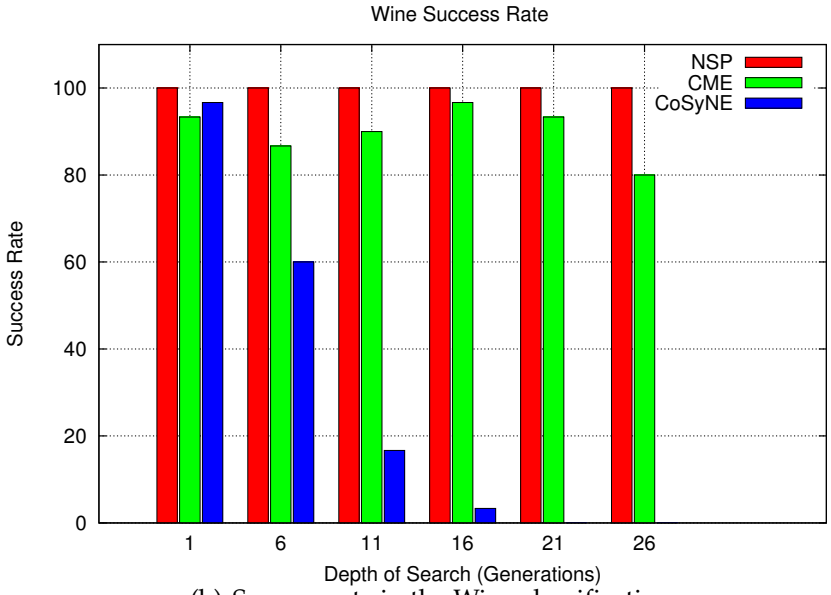


(b) Success rate in the Iris classification

Figure 3.7: The evaluation of the depth of search in the different problem decomposition methods for the Iris classification problem.

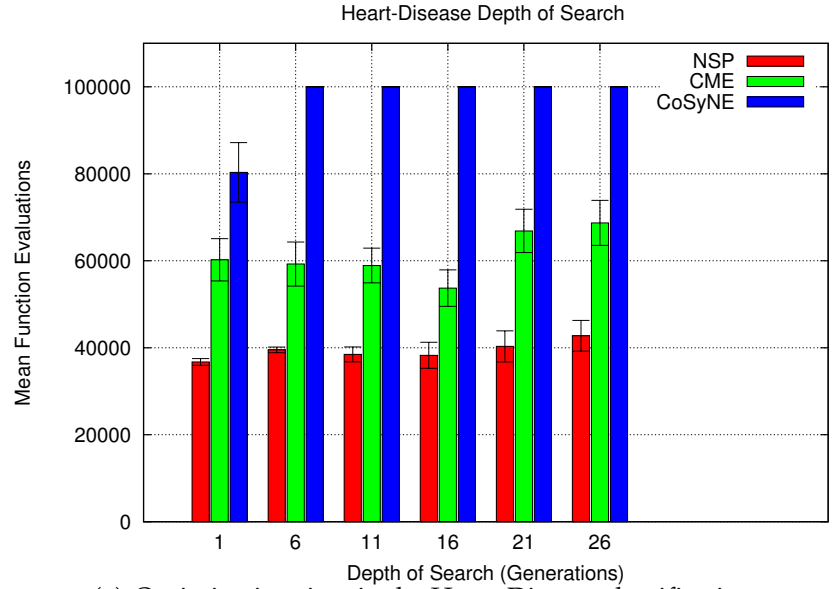


(a) Optimization time in the Wine classification

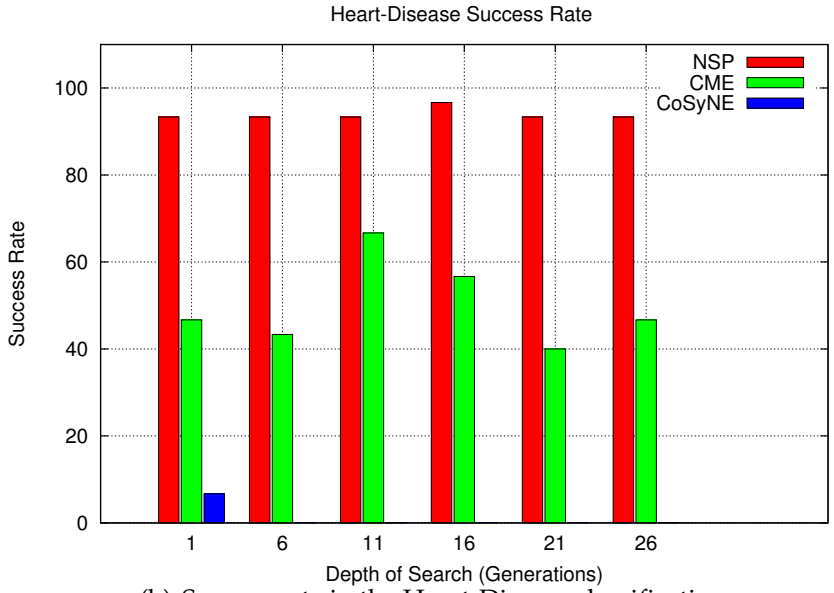


(b) Success rate in the Wine classification

Figure 3.8: The evaluation of the depth of search in the different problem decomposition methods for the Wine classification problem.

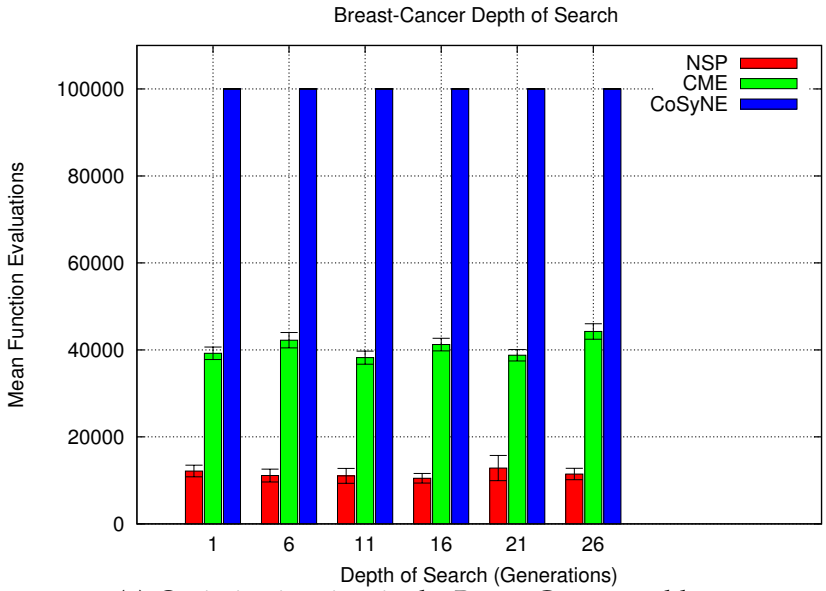


(a) Optimization time in the Heart-Disease classification

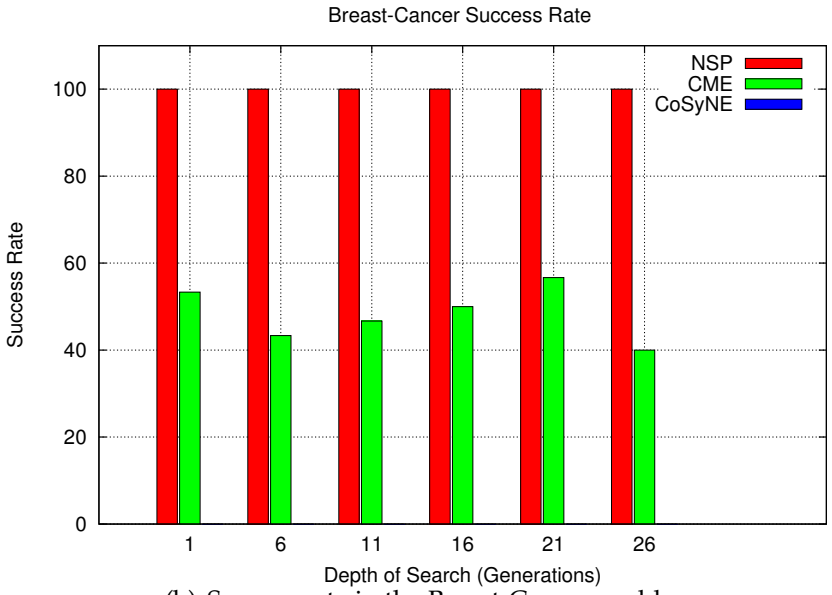


(b) Success rate in the Heart-Disease classification

Figure 3.9: The evaluation of the depth of search in the different problem decomposition methods for the Heart-Disease classification problem.



(a) Optimization time in the Breast-Cancer problem



(b) Success rate in the Breast-Cancer problem

Figure 3.10: The evaluation of the depth of search in the different problem decomposition methods for the Breast-Cancer classification problem.

is due to the difference in the encoding schemes for the respective problem decomposition methods which use different number of subcomponents in the three methods.

In summary, the grouping used in the problem decomposition methods directly influences the training problem. NSP performs well regardless of the depth of search used, while CoSyNE shows good performance for the depth of 1 generation only. In general, CoSyNE is unreliable due to its low success rate in converging to a solution. This shows that the NSP encoding has been successful in grouping the interacting variables in the sub-populations. There is less interaction between sub-populations in NSP which makes a deep greedy search possible. In CoSyNE and CME, the depth of search is significant for almost all the problems which implies that the variables interact within different sub-populations. Moreover, the large depths fail to deliver good solutions in terms of training time and success rate. CoSyNE uses a higher number of subcomponents and it does not group interacting variables as the number of variables in a subcomponent is restricted to 1.

The results for the generalisation performance are given in Tables 3.1 and 3.2. The "Success" column in the tables indicates the number of successful runs out of 30 experimental runs.

Note that the mean and the 95 % confidential interval of the generalisation performance only includes the successful runs. The results show that NSP has delivered similar generalisation performance in comparison to CME and CoSyNE while achieving a faster training performance with a much better success rate.

The Scalability of NSP for Feedforward Networks

The goal of this experiment is to observe the performance of the respective encoding schemes in relation to the particular network topology; i.e. fixed number of hidden neurons. Note that the number of hidden neurons directly influences the difficulty of the learning problem. It is more

Method	Depth	Iris	Success/30	Wine	Success/30
NSP	1	$95.08_{1.08}$	30	$94.58_{1.28}$	30
	6	$94.47_{1.34}$	30	$94.67_{0.97}$	30
	11	$94.91_{1.24}$	30	$92.41_{1.70}$	30
	16	$93.51_{1.28}$	30	$93.67_{1.47}$	30
	21	$94.55_{0.97}$	29	$93.41_{1.43}$	30
	26	$94.74_{1.06}$	30	$94.41_{1.38}$	30
CME	1	$95.17_{1.19}$	12	$94.64_{1.12}$	28
	6	$94.73_{1.98}$	11	$95.00_{1.30}$	26
	11	$95.48_{1.36}$	7	$92.77_{1.65}$	27
	16	$94.33_{1.84}$	13	$94.22_{1.74}$	29
	21	$95.11_{1.62}$	7	$93.83_{0.94}$	28
	26	$94.44_{1.70}$	9	$94.47_{1.38}$	24
CoSyNE	1	$95.26_{0.65}$	10	$92.58_{0.97}$	29
	6	$92.98_{2.14}$	9	$90.00_{2.07}$	18
	11	$92.76_{1.11}$	4	$88.00_{1.63}$	5
	16	94.73_{0}	1	92.50_{0}	1
	21	94.73_{0}	1	–	0
	26	$92.10_{3.64}$	2	–	0

Table 3.1: The generalisation performance given by the different problem decomposition methods for the Iris and Wine classification problems. The 95 % confidence interval is given in the subscript with the number of successful runs (Success) out of 30 experiments.

Method	Depth	Heart-Disease	Success/30	Breast-Cancer	Success/30
NSP	1	$78.10_{1.04}$	28	$97.35_{0.35}$	30
	6	$76.89_{1.03}$	28	$97.05_{0.45}$	30
	11	$78.00_{0.99}$	28	$97.24_{0.50}$	30
	16	$78.75_{1.03}$	29	$97.16_{0.29}$	30
	21	$77.82_{0.82}$	28	$97.20_{0.45}$	30
	26	$77.78_{1.12}$	28	$97.23_{0.47}$	30
CME	1	$79.14_{1.15}$	14	$97.83_{0.45}$	16
	6	$78.84_{0.82}$	13	$97.41_{0.38}$	13
	11	$78.90_{0.92}$	20	$97.33_{0.44}$	14
	16	$80.29_{0.85}$	17	$97.63_{0.38}$	15
	21	$79.34_{0.81}$	12	$97.67_{0.51}$	17
	26	$79.29_{0.80}$	14	$97.81_{0.45}$	12
CoSyNE	1	$77.14_{1.55}$	2		
	6	–	0	–	0
	11	–	0	–	0
	21	–	0	–	0
	26	–	0	–	0

Table 3.2: The generalisation performance given by the different problem decomposition methods for the Heart-Disease and Breast-Cancer classification problems. The 95 % confidence interval is given in the subscript with the number of successful runs (Success) out of 30 experiments.

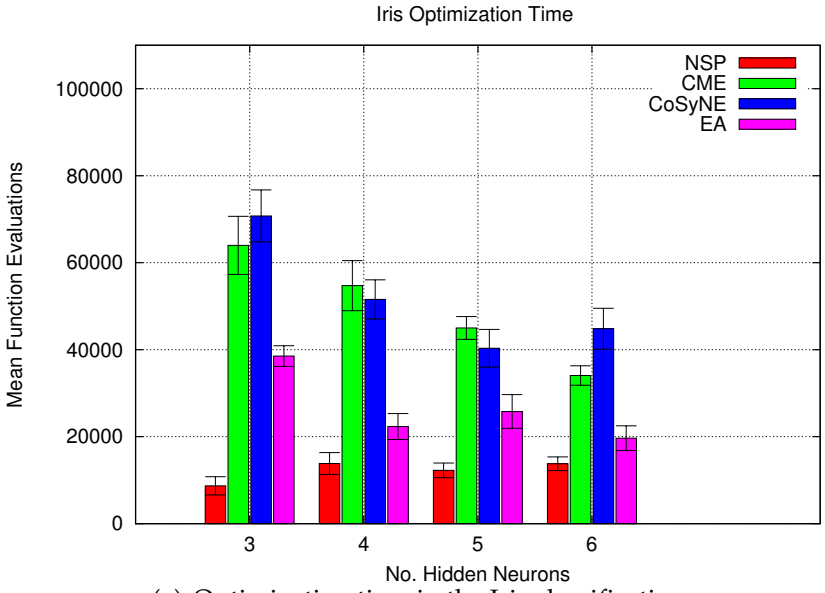
difficult to learn the problem if not enough neurons are present in the hidden layer. Moreover, increasing the number of hidden neurons also increases the number of subcomponents for the respective methods, as described earlier. It is important to observe the performance of each method when the same problem is represented with different numbers of subcomponents. The difficulty in evolutionary algorithms increases with an increase in the size of the number of variables. It would be interesting to see the behaviour of each cooperative coevolution method with the increase in the number and the size of each individual subcomponent. This reflects on scalability and robustness. The results are further compared to neuro-evolution which employs a standard evolutionary algorithm as a benchmark comparison for this behaviour. The experiments involve a fixed network architecture.

As summarised from the results in the previous section, the lowest depth of search of 1 generation is used which showed optimal or near-optimal performance for NSP, CME and CoSYNE.

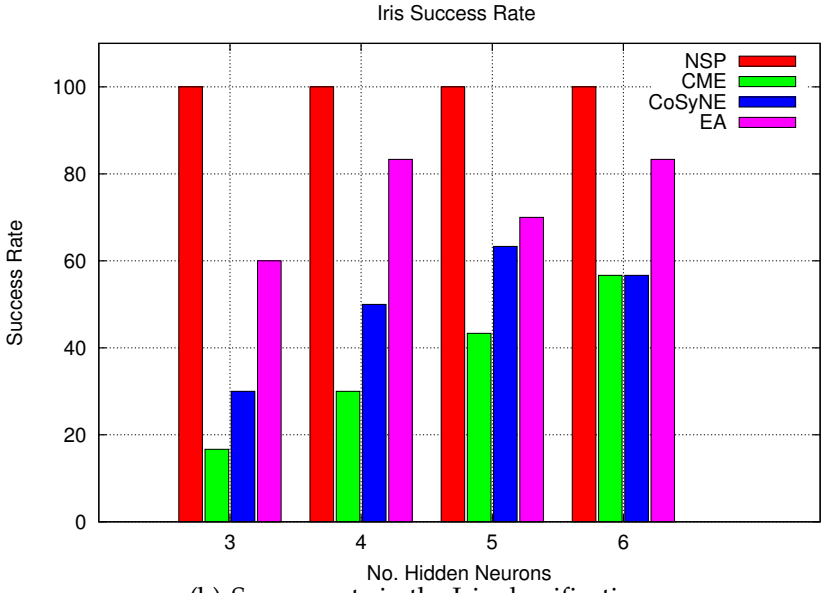
The results in Figures 3.11 to 3.14 show the comparison of the three problem decomposition methods (NSP, CME, CoSyNE) with evolutionary algorithms (EA) for different numbers of hidden neurons. The G3-PCX is used as the designated evolutionary algorithm in all cases. We evaluate the methods by measuring the number of function evaluations and the success rate in 30 independent experimental runs. The mean of the number of function evaluations is shown with 95 % confidence interval as error bars in the histograms.

Tables 3.3 and 3.4 show the generalisation performance of the four different problems with different number of hidden neurons. As before, the results from the unsuccessful runs are not included in the mean and 95 % confidence interval.

Note that the generalisation performance does not only depend on the optimisation algorithm, but also on the neural network architecture. The success rate reflects on robustness of the algorithm. The unsuccessful runs

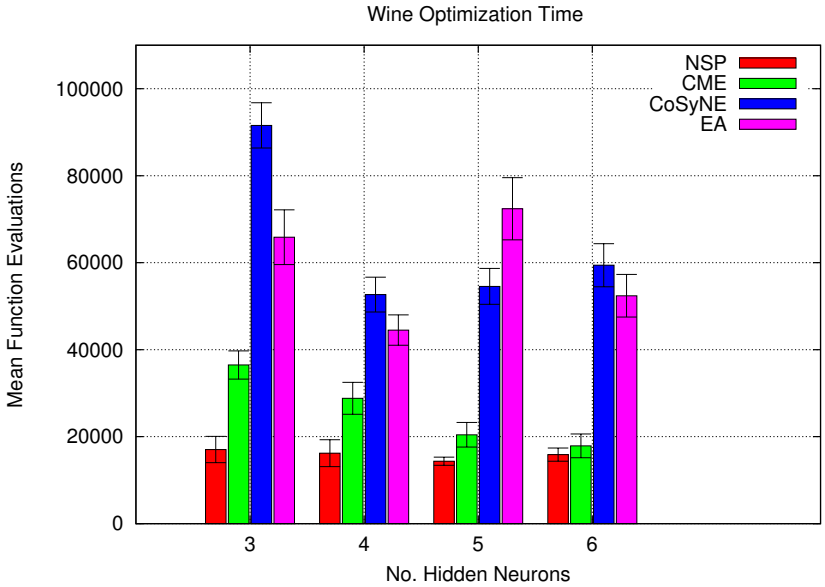


(a) Optimization time in the Iris classification

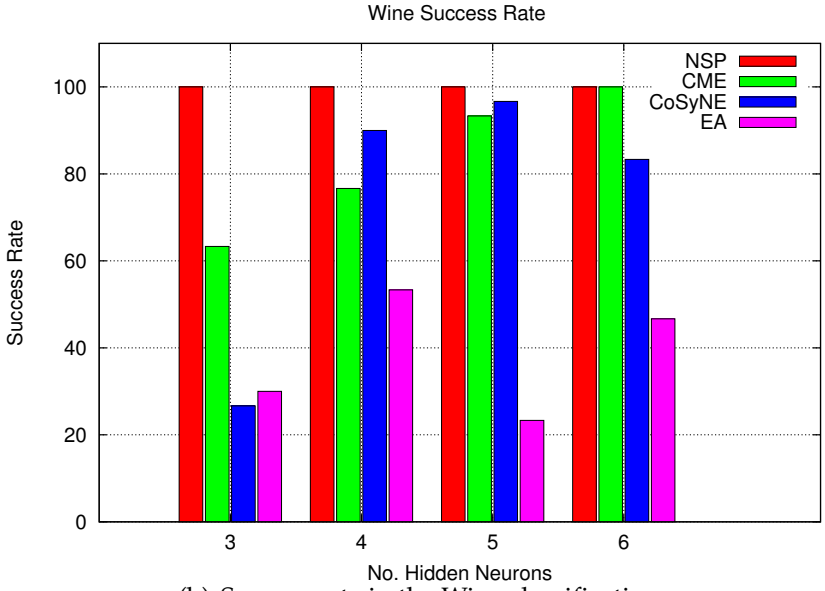


(b) Success rate in the Iris classification

Figure 3.11: The performance of the different problem decomposition methods for different number hidden neurons for the Iris classification problem.

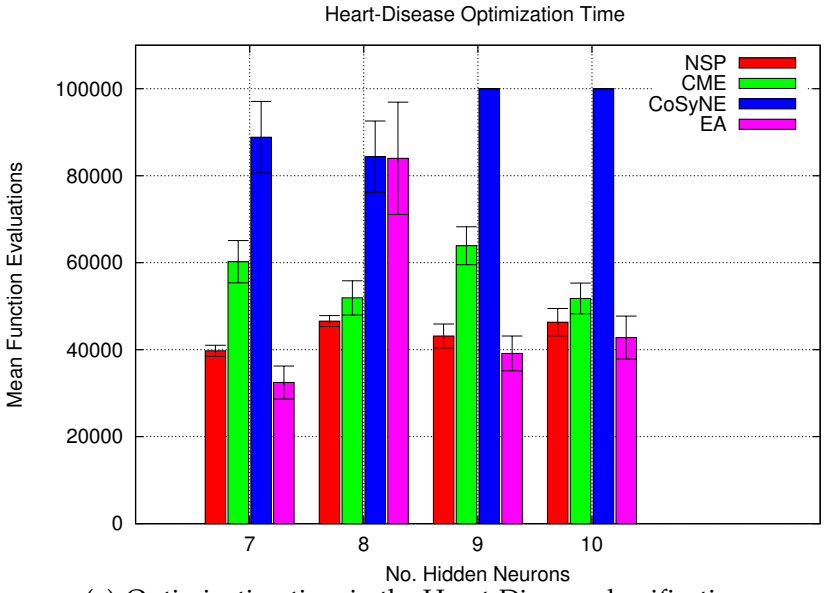


(a) Optimization time in the Wine classification

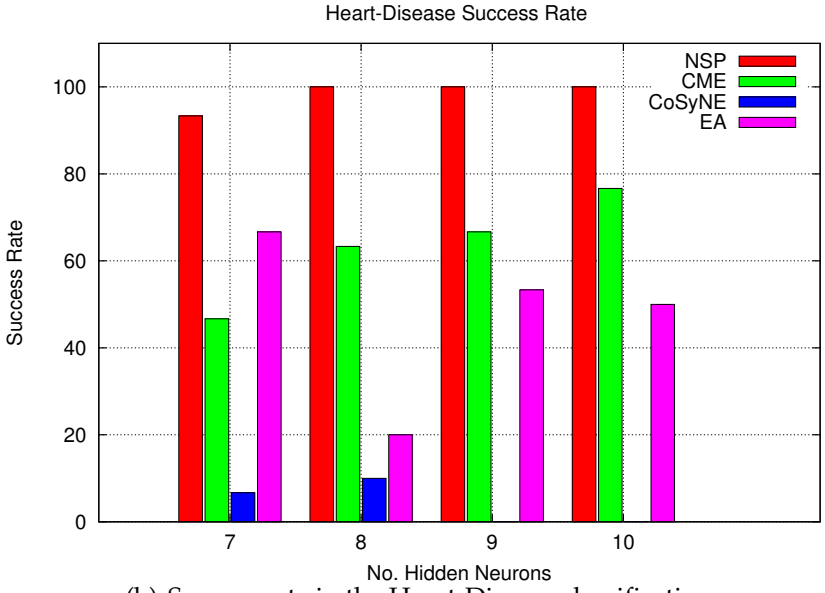


(b) Success rate in the Wine classification

Figure 3.12: The performance of the different problem decomposition methods for different number hidden neurons for the Wine classification problem.

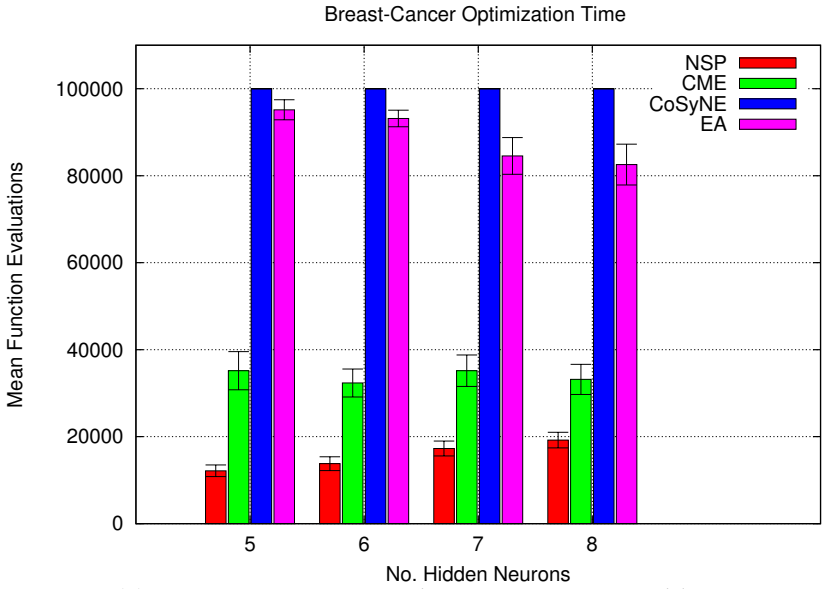


(a) Optimization time in the Heart-Disease classification

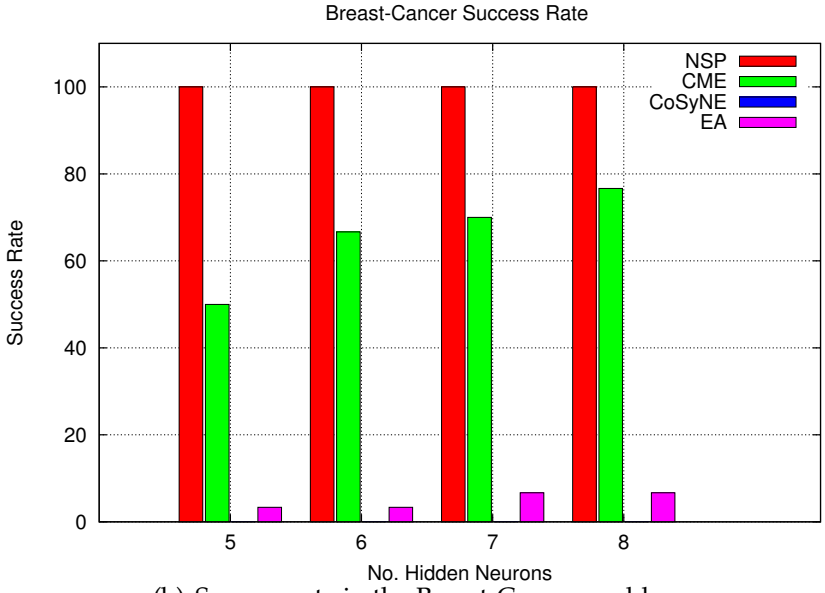


(b) Success rate in the Heart-Disease classification

Figure 3.13: The performance of the different problem decomposition methods for different number hidden neurons for the Heart-Disease classification problem.



(a) Optimization time in the Breast-Cancer problem



(b) Success rate in the Breast-Cancer problem

Figure 3.14: The performance of the different problem decomposition methods for different number hidden neurons for the Breast-Cancer classification problem.

have already been included in the mean and the confidence interval for the optimisation time given by number of function evaluations. Therefore, the results from the unsuccessful runs were not included in determining the generalisation performance. Moreover, success rate and function evaluations are inter-dependent on each other as they measure optimisation time and robustness. Generalisation is calculated after the training stage that is not dependent on the success rate.

The results in general show that the generalisation performance for NSP is similar or better than the other methods. This implies that NSP can achieve similar generalisation performance with lower optimization time (and higher success rates).

Method	Hidden	Iris	Success/30	Wine	Success/30
NSP	3	$95.08_{1.02}$	30	$93.25_{1.51}$	30
	4	$94.12_{1.51}$	30	$94.58_{1.22}$	30
	5	$95.00_{1.01}$	30	$93.91_{1.23}$	30
	6	$94.82_{1.07}$	30	$94.00_{1.25}$	30
CME	3	$91.57_{2.69}$	5	$93.55_{1.56}$	19
	4	$94.15_{1.77}$	9	$92.82_{1.54}$	23
	5	$93.52_{1.32}$	13	$95.26_{1.11}$	28
	6	$94.58_{1.63}$	17	$93.75_{1.34}$	30
CoSyNE	3	$92.95_{2.14}$	9	$92.50_{2.12}$	8
	4	$94.91_{1.02}$	15	$91.48_{1.09}$	27
	5	$94.45_{1.00}$	19	$92.41_{0.90}$	29
	6	$94.89_{1.09}$	17	$91.10_{1.11}$	25
EA	3	$91.95_{1.69}$	18	$96.67_{0.77}$	9
	4	$93.15_{1.01}$	25	$93.28_{2.10}$	16
	5	$93.23_{1.41}$	21	$92.50_{3.28}$	7
	6	$91.78_{1.28}$	25	$92.79_{2.28}$	14

Table 3.3: The generalisation performance of the problem decomposition methods given different number of hidden neurons for the Iris and Wine classification problem. The 95 % confidence interval is given in the subscript with the number of successful runs (Success) out of 30 experiments.

Method	Hidden	Heart-Disease	Success/30	Breast-Cancer	Success/30
NSP	7	$78.10_{1.04}$	28	$97.35_{0.35}$	30
	8	$78.16_{1.20}$	30	$97.17_{0.33}$	30
	9	$77.06_{0.75}$	30	$97.27_{0.36}$	30
	10	$77.56_{1.08}$	30	$97.22_{0.32}$	30
CME	7	$79.14_{1.15}$	14	$97.65_{0.65}$	15
	8	$79.63_{0.94}$	19	$97.55_{0.67}$	20
	9	$79.45_{0.87}$	20	$97.47_{0.72}$	21
	10	$79.65_{0.84}$	23	$97.63_{0.77}$	23
CoSyNE	7	$77.00_{1.38}$	2	–	0
	8	$81.00_{1.84}$	3	–	0
	9	–	0	–	0
	10	–	0	–	0
EA	7	$80.20_{0.52}$	20	97.44_{0}	1
	8	$79.83_{1.25}$	6	97.46_{0}	1
	9	$80.68_{0.59}$	16	$97.25_{0.25}$	2
	10	$79.60_{1.05}$	15	$97.44_{0.34}$	2

Table 3.4: The performance of different problem decomposition methods given different number of hidden neurons for the Heart-Disease and Breast-Cancer classification problem. The 95 % confidence interval is given in the subscript with the number of successful runs (Success) out of 30 experiments.

In the Iris classification problem given in Figure 3.11, NSP outperforms other methods by achieving a higher success rate with lower optimization time. It achieves the best results in the different network topologies given in terms of different numbers of hidden neurons. The EA method gives better results than CME and CoSyNE. In the Wine classification problem shown in Figure 3.12, NSP further outperforms the rest of the methods for the different number of hidden neurons. In this case, CME performs better than EA, CoSyNE fails to compete with EA.

In the Heart-Disease classification problem shown in Figure 3.13, NSP gives the best performance when compared to CME, CoSyNE and EA given the different number of hidden neurons. In the Breast-Cancer classification problem shown in Figure 3.14, NSP outperforms all the other methods as well.

In summary, the NSP encoding gave the best performance in all the problems given different number of hidden neurons. This shows that NSP scales up better than other methods. The comparison of the performance of the different cooperative neuro-evolution methods with EA further justifies the need for using cooperative coevolution in evolving feedforward networks for pattern classification problems. NSP has been shown to achieve similar or better generalisation performance than the other methods with a better success rate.

Evaluation of Diversity and Interdependencies in NSP

There is some evidence given in terms of the depth of search that NSP is a better problem decomposition method in terms of the degree of non-separability. An alternative interpretation of the results could be that NSP performs better due to the size of its subcomponent, rather than the actual problem decomposition which groups interacting variables.

In this section, the behavior of NSP in terms of diversity and interdependencies is tested by having the elements of its subcomponents selected from other subcomponents; however, the size of all the subcomponents re-

main the same. This is done by interchanging the positions of some chosen synapses from the input-hidden layer subcomponents with the hidden-output layer subcomponents. This approach is called Random-NSP subcomponents in this section which is compared to NSP.

Note that the problems are made more difficult by reducing the maximum optimisation time. The Iris and Wine classification problem used 15,000 function evaluations for the maximum time with 5 hidden neurons and the Heart-Disease problem used 50, 000 function evaluations with 7 hidden neurons. The Cancer classification problem used maximum of 15, 000 function evaluations with 6 hidden neurons.

In the setup for the Random-NSP for the Iris and Wine problems, 2 synapses in the first 2 subcomponents in the input-hidden layer is interchanged by 2 synapses of the 2 subcomponents from the hidden-output layer. In Random-NSP for the Cancer and Heart-Disease problems, 2 synapses in the first subcomponent in the input-hidden layer is interchanged by 2 synapses of the first subcomponent from the hidden-output layer. In this way, NSP and Random-NSP methods use same subcomponent sizes but different grouping of synapses.

The results are shown in Table 3.5. The results indicate that NSP has performed better than Random-NSP mostly in terms of function evaluations for Wine, Cancer and Heart-Disease problems. In the Iris problem, there is some difference in the performance of both methods that vary according to the depth of search.

The difference in the results given by the two methods shows that the performance of NSP is not entirely due to the size of the subcomponents it has but also due to the way the synapses are grouped in the respective subcomponents.

Problem	Depth	Random-NSP		NSP	
		FuncEval	Success (%)	FuncEval	Success (%)
Wine	1	8060	96	6326	100
	4	7395	100	5506	100
	7	7136	96	5187	100
	10	7939	96	5318	100
	13	6801	100	5378	100
Iris	1	7402	86	8581	92
	4	9159	76	7875	92
	7	9190	88	9349	80
	10	9601	84	8576	92
	13	9074	82	10253	76
Heart	1	41001	56	29950	74
	4	44040	60	26860	82
	7	46980	38	23022	80
	10	43210	62	23404	80
	13	39010	58	24635	78
	1	13187	36	12355	26
	4	12182	48	11164	38
	7	13833	32	10885	36
	10	13107	30	11262	38
	13	13107	34	10964	42

Table 3.5: Comparison of Random-NSP with NSP

3.3.2 Recurrent Neural Networks

This section presents an experimental study of NSP for recurrent neural networks and compares it with CoSyNE. CoSyNE [77] has shown better performance than ESP and standard neuro-evolution. Therefore, it is reasonable to compare the performance of NSP with CoSyNE. The G3-PCX [42] evolutionary algorithm is used in both methods. The Elman recurrent network [48] with one hidden layer is used in all the experiments. The sub-populations are seeded with random real numbers in the range of [-5, 5] in all experiments.

Grammatical inference is used as a means to study the performance of the proposed NSP encoding in recurrent neural networks. The FFA shown in Figure 2.4 from the literature Section 2.2.6 is used as a benchmark problem. The training dataset is generated by presenting strings of increasing lengths of 1 to 7 to the FFA and the corresponding output for each sample is noted. Note that for every string length, all the possible bits are generated. The training set consists of 255 samples. Similarly, the testing dataset with string lengths of 8 to 14 using the same FFA is generated. The recurrent network topology for the FFA is as follows: 1) one neuron in the input layer, 2) two output neurons in the output layer representing the 4 fuzzy output states of the FFA. The RNN is trained until 100 percent of the training sample is correctly classified or when the maximum number of function evaluation is reached. This value has been determined in trial experiments. For the FFA problem, the maximum number of function evaluations is pre-set to 20000.

Similarly, we generated the training and test datasets from the Tomita language shown in Figure 2.3 of Chapter 2. We used Tomita 3 (T3) and Tomita 4 (T4) for comparison. In this case, the training and testing data is generated by presenting random strings of length 15 to 25 for each Tomita language. The training and test datasets each contain 250 (125 positive and 125 negative) string samples. The maximum number of function evaluations in T3 and T4 is 20000. Note that the string lengths considered here

(15-25) cannot be trained using backpropagation-through-time as outlined in [226].

Depth of Search for NSP in Recurrent Networks

In the NSP encoding for recurrent networks shown in Algorithm 5 , each sub-population is evolved for a fixed number of generations in a round-robin fashion. The study begins by determining the optimal number of generations needed for the sub-population which is considered as the depth of search. Note that all sub-populations are meant to evolve for the same number of n generations which must be fixed beforehand as done in the case of evolving feedforward networks in Section 3.3.1 for a fair comparison.

The FFA used in this experiment has 7 states and in order to make the problem more difficult, only 4 neurons in the hidden layer of the RNN are used to represent 7 states. In the T1 and T2 problems, 2 neurons in the hidden layer are used. In the T3 and T4 problems, 3 neurons in the hidden layer are used.

The results given in Figures 3.15 to 3.19 report the optimization time with respect to the depth of search needed in the respective cooperative coevolution method (NSP and CoSyNE). These results are for the evolution phase only. The results do not include the time taken for the initialisation phase as the goal is to observe the convergence of the respective methods during evolution.

In the T1 problem shown in Figure 3.15, the depth of 1 to 5 generations in NSP gives similar performance, while in CoSyNE, the depth of 1 generation gives the best result. The performance of CoSyNE significantly deteriorates with depth larger than 1 generation. Similar trend is given in the T2 problem shown in Figure 3.16, the depth of 1 to 7 generations in NSP gives similar performance, while in CoSyNE, the depth of 1 generation gives the best result.

In the T3 problem shown in Figure 3.17, the depth of 1 to 5 generations

in NSP achieves similar performance, while in CoSyNE, the depth of 1 generation only gives the best result. In the T4 problem shown in Figure 3.18, the depth of 1 to 5 generations in NSP achieves similar performance considering the optimization time and the success rates. In CoSyNE, the depth of 1 generation only gives the best result. In the FFA problem shown in Figure 3.19, the depth of 1 generation in NSP and CoSyNE gives the best results. The performance of CoSyNE significantly deteriorates with depth larger than 1 generation for T3, T4 and the FFA problem. Note that the least optimization time and high success rate determines the performance evaluation. In all the problems, both methods report that the performance deteriorates at some stage as the depth increases.

In general, with NSP, the depth of 1 generation gives the best performance. The depth from 1-5 generations gives similar or acceptance performance. The performance deteriorates with a larger depth. In CoSyNE, the depth of 1 generation gives the best performance and the performance deteriorates otherwise. The comparison of the best results from NSP and CoSyNE shows that NSP has been able to solve the problem in less optimization time with a high success rate when compared to CoSyNE. This is due to the difference in the problem decomposition methods. In NSP, the interacting variables have been grouped efficiently into separate subcomponents. In CoSyNE, there is no grouping of interacting variables at the size of each subcomponent is restricted to 1, therefore, only a shallow depth of search has been able to yield acceptable performance. In NSP, the interacting variables have been grouped according to the way they interact with the respective neurons. Therefore, a deeper depth of search in the subcomponents has been possible (1 to 5 generations) in most of the problems.

The generalisation performance is given in Table 3.6 show that in all the problems, both methods have been able to achieve 100 % generalisation performance on unseen data. The results for the depth of 1 generation is shown only as both methods have been able to achieve the best perfor-

mance for this value, i.e a shallow depth with much better success rate.

Problem	NSP	Success	CoSyNE	Success
T1	100	100	100	95
T2	100	83	100	67
T3	100	45	100	37
T4	100	100	100	77
FFA	100	29	100	10

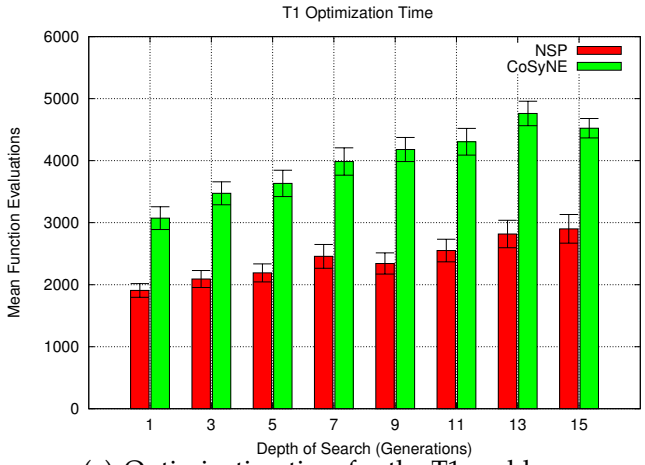
Table 3.6: The generalisation performance in percentage is given by the different problem decomposition methods for the depth of search of 1 generation. Note that the generalisation performance does not include the performance of the unsuccessful runs in the mean. The success rate (Success) out of 100 experiments is also given.

The Scalability of NSP for Recurrent Networks

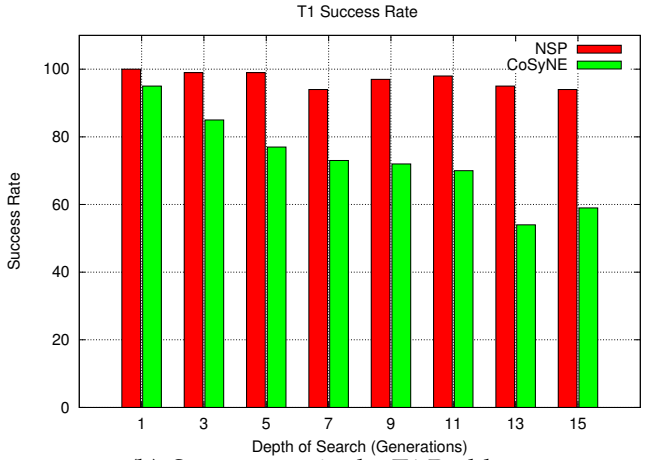
In this section, the performance of NSP is compared with CoSyNE. Note that the original CoSyNE employed a different evolutionary algorithm in their sub-population. The G3-PCX is used in both encoding schemes. The depth of 1 generation is used in NSP and CoSyNE.

Table 3.7 shows the relationship between the number of function evaluations and the number of hidden neurons used in the initialisation phase of NSP and CoSyNE. The RNN topology has 1 input neuron and 2 output neurons. The results show that the number of function evaluations given in terms of the population size P increases as the size of the networks increases in terms of "Hidden" neurons. This directly relates to the number of subcomponents represented by the sub-populations. Note that NSP uses fewer number of function evaluations shown in Table 3.7 as it requires lesser number of subcomponents when compared to CoSyNE. Therefore, the initialisation phase of evaluating different subcomponent encoding schemes for cooperative coevolution is an important measure.

In all problems, the RNN is trained until the mean-squared-error (MSE)

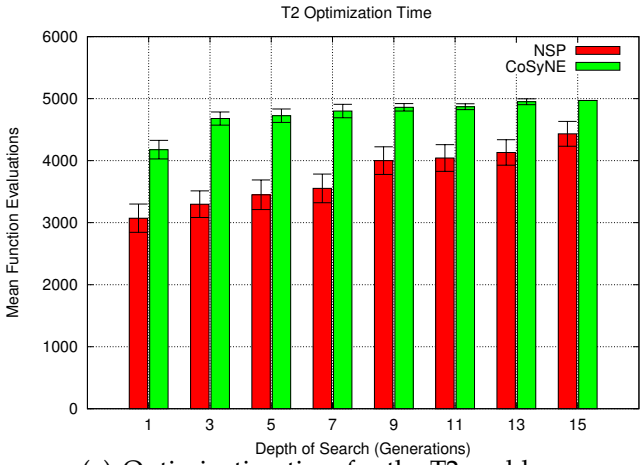


(a) Optimization time for the T1 problem

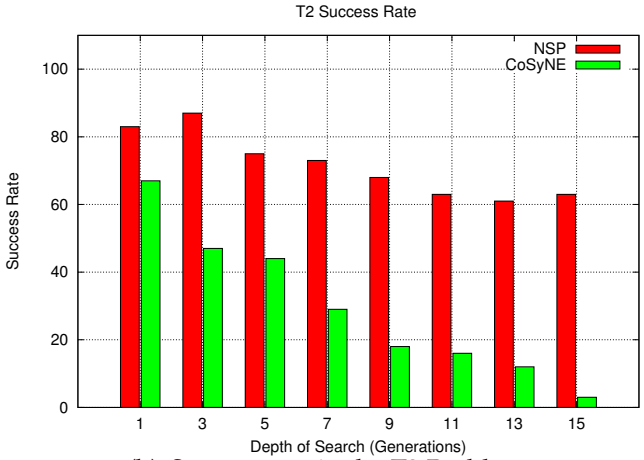


(b) Success rate in the T1 Problem

Figure 3.15: The performance of NSP and CoSyNE for the T1 problem. The optimization time in terms of the average number of function evaluations is shown in (a) and the number of success rate is shown in (b). A total of 100 independent experimental runs have been done.

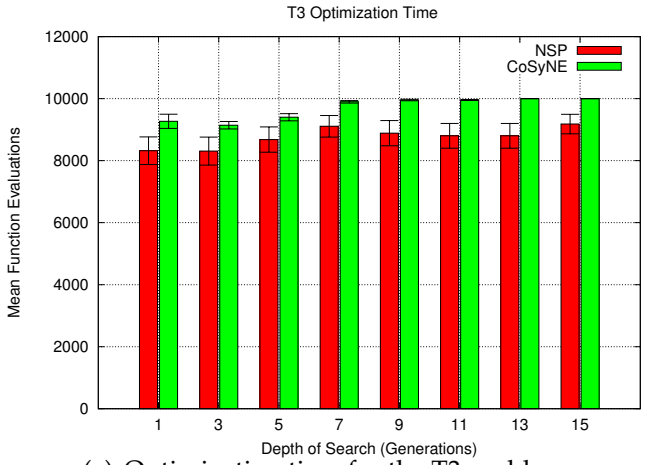


(a) Optimization time for the T2 problem



(b) Success rate in the T2 Problem

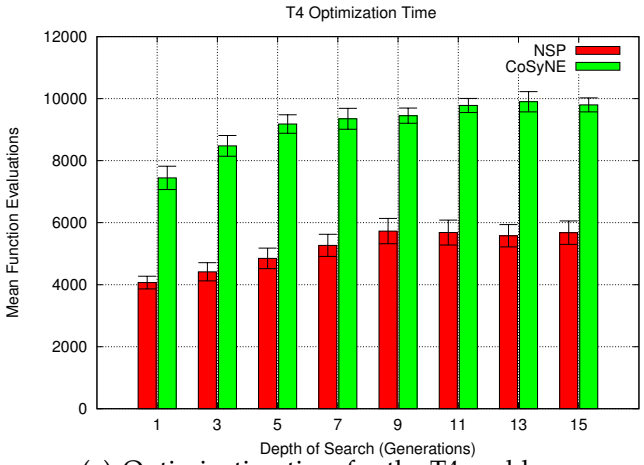
Figure 3.16: The performance of NSP and CoSyNE for the T2 problem. The optimization time in terms of the average number of function evaluations is shown in (a) and the number of success rate is shown in (b). A total of 100 independent experimental runs have been done.



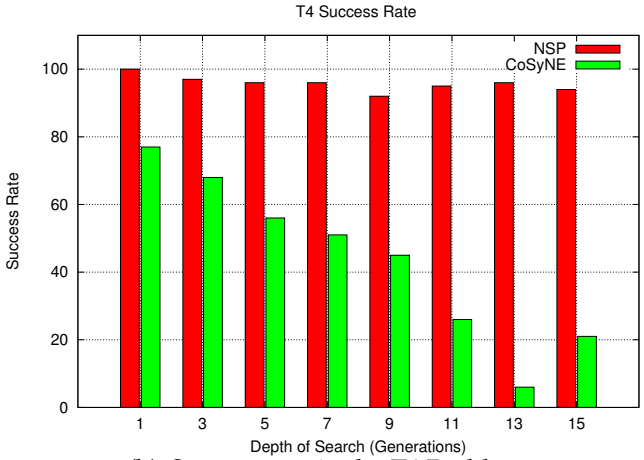
(a) Optimization time for the T3 problem

(b) Success rate in the T3 Problem

Figure 3.17: The performance of NSP and CoSyNE for the T3 problem. The optimization time in terms of the average number of function evaluations is shown in (a) and the number of success rate is shown in (b). A total of 100 independent experimental runs have been done.

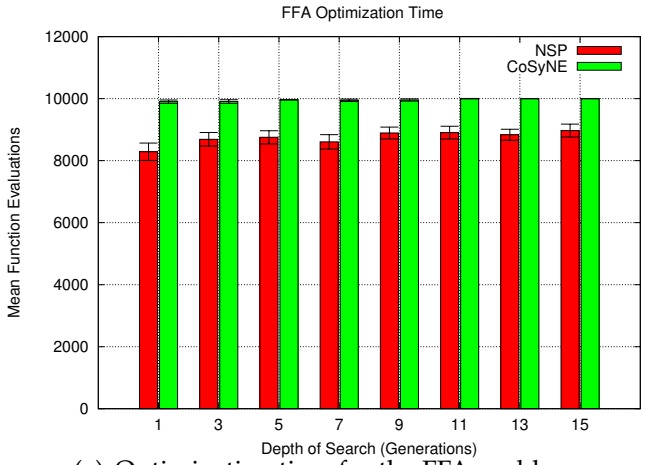


(a) Optimization time for the T4 problem

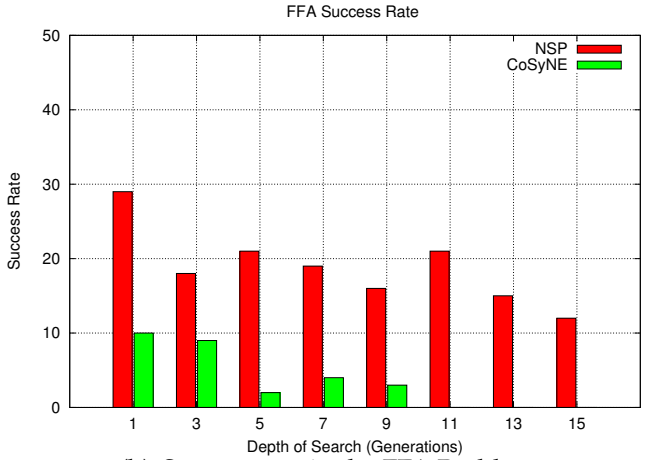


(b) Success rate in the T4 Problem

Figure 3.18: The performance of NSP and CoSyNE for the T4 problem. The optimization time in terms of the average number of function evaluations is shown in (a) and the number of success rate is shown in (b). A total of 100 independent experimental runs have been done.



(a) Optimization time for the FFA problem



(b) Success rate in the FFA Problem

Figure 3.19: The performance of NSP and CoSyNE for the FFA problem. The optimization time in terms of the average number of function evaluations is shown in (a) and the number of success rate is shown in (b). A total of 100 independent experimental runs have been done.

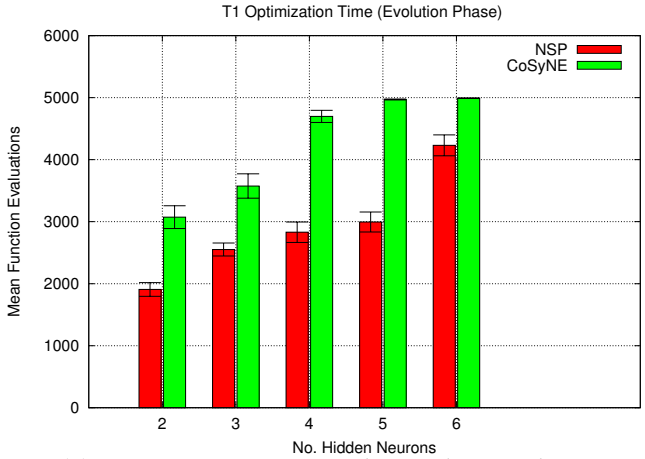
reaches below 0.0005 or when the number of function evaluation exceeds the maximum. The maximum number of function evaluations for T1 and T2 is 5000. T3, T4 and FFA problem use the maximum of 10000 function evaluations. These values are different from the previous section and have been made lower in order to make the problem more difficult; i.e the evolutionary methods need to converge before these limits.

Hidden	NSP	CoSyNE
3	8	23
4	10	34
5	12	47
6	14	62
7	16	79

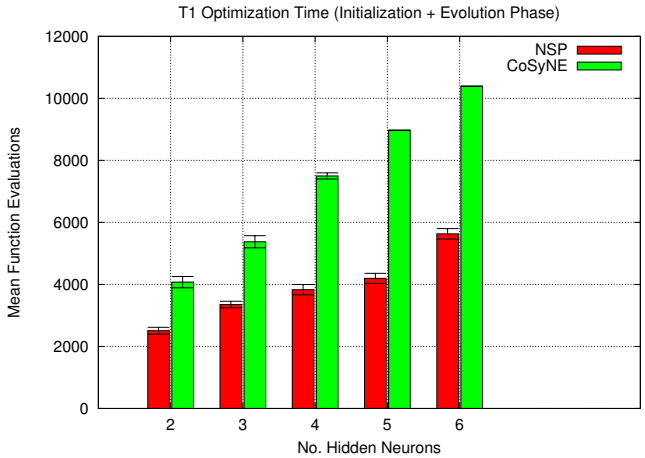
Table 3.7: A comparison of NSP and CoSyNE based on the number of function evaluation required during initialisation. This is for a RNN with one input neuron and two output neurons which is used in all our experiments. The comparison is done in terms of P which is the size of the population.

The comparative results during evolution are given in Figures 3.20 to 3.24 where the attribute "Hidden" represents the number of hidden neurons. The respective figures first show the results for the evolution phase only and then for the total optimization time with the respective success rates. The two methods are evaluated using different number of hidden neurons. A total of 100 experiments is done for each case and the mean function evaluation is given for the respective problems. The total optimization time includes the initialisation phase and the evolution phase. The results include the 95 % confidence interval given as error bars in the histograms which evaluate the optimization time.

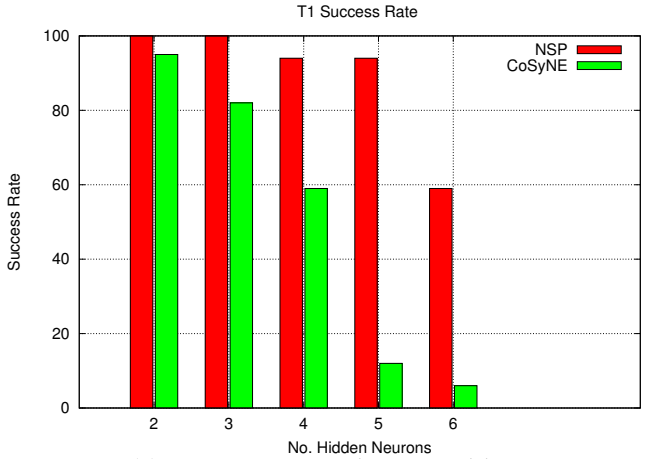
Note that the T1 and T2 problem are easier problems than the rest of the problems as they can be learned in a relatively shorter time. In Figure 3.20, NSP shows better performance than CoSyNE in (a) the evolution phase and (b) the initialisation plus the evolution phase with higher success rate



(a) Optimization time in the Evolution Phase

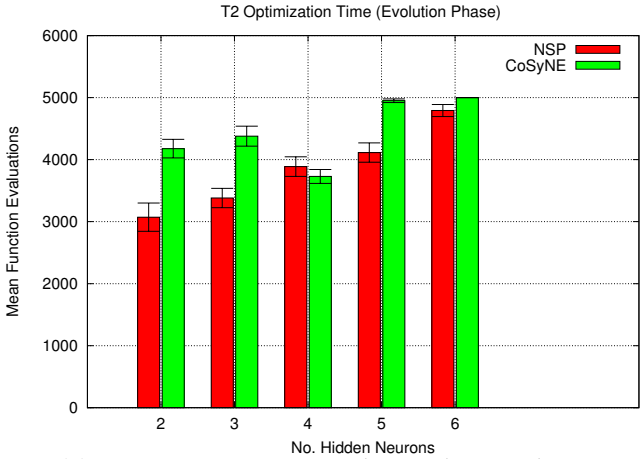


(b) Total optimization time = Initialisation + Evolution Phase

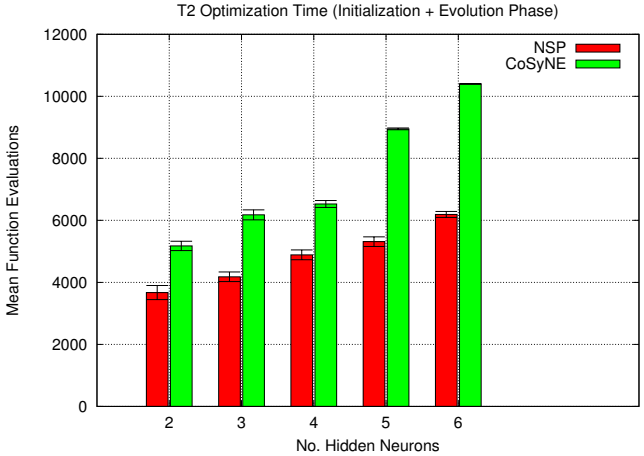


(c) Success rate in the T1 Problem

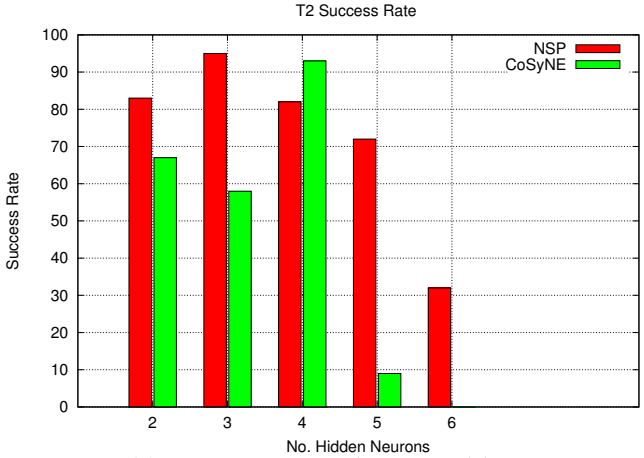
Figure 3.20: The performance of NSP and CoSyNE on different number of hidden neurons for the T1 problem. The optimization time in terms of the average number of function evaluations (Evolution Phase) is shown in (a), the total optimization time which include the initialisation and Evolution Phase is shown in (b) and the number of successful runs is shown in (c).



(a) Optimization time in the Evolution Phase

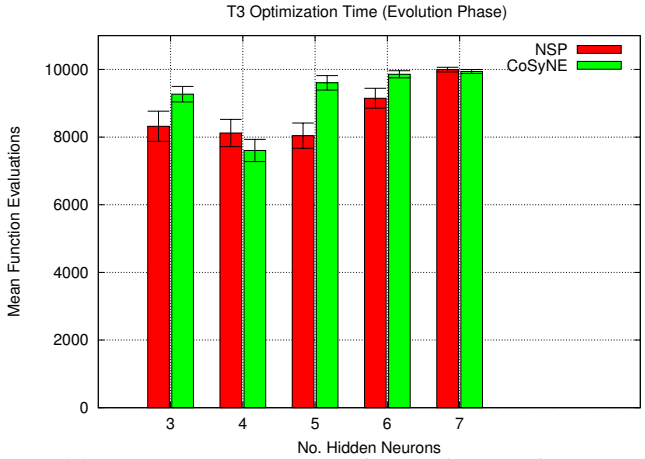


(b) Total optimization time = Initialisation + Evolution Phase

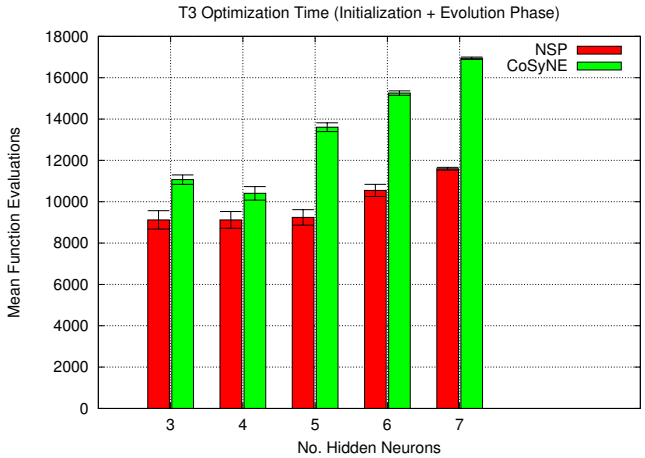


(c) Success rate in the T2 Problem

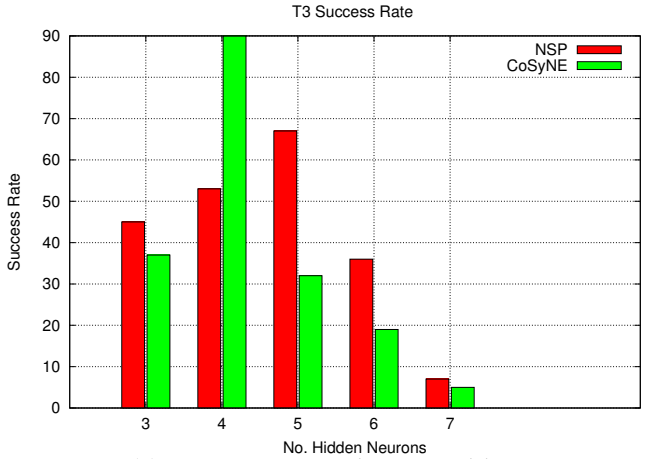
Figure 3.21: The performance of NSP and CoSyNE on different number of hidden neurons for the T2 problem. The optimization time in terms of the average number of function evaluations (Evolution Phase) is shown in (a), the total optimization time which include the initialisation and Evolution Phase is shown in (b) and the number of successful runs is shown in (c).



(a) Optimization time in the Evolution Phase

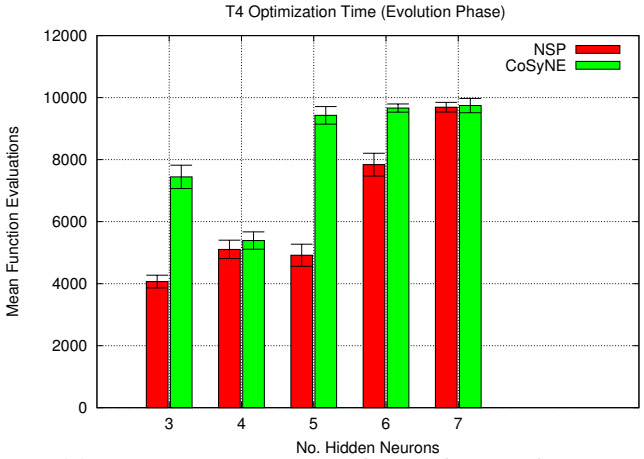


(b) Total optimization time = Initialisation + Evolution Phase

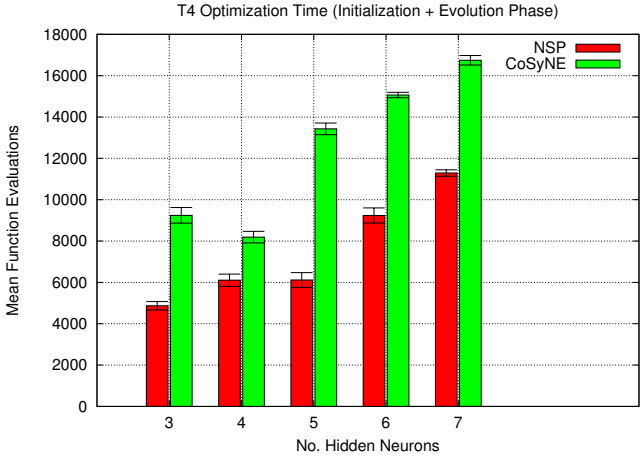


(c) Success rate in the T3 Problem

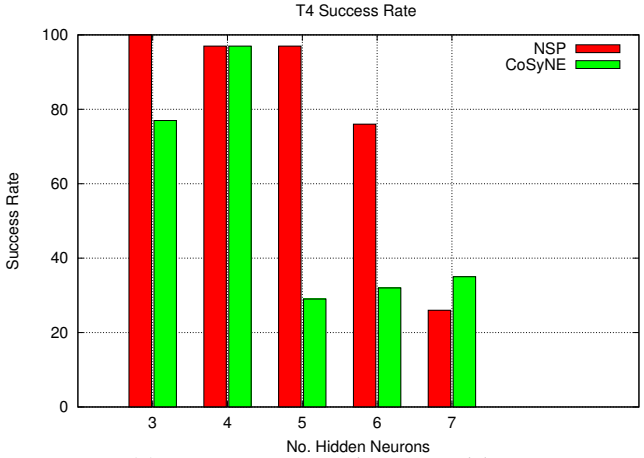
Figure 3.22: The performance of NSP and CoSyNE on different number of hidden neurons for the T3 problem. The optimization time in terms of the average number of function evaluations (Evolution Phase) is shown in (a), the total optimization time which include the initialisation and Evolution Phase is shown in (b) and the number of successful runs is shown in (c).



(a) Optimization time in the Evolution Phase

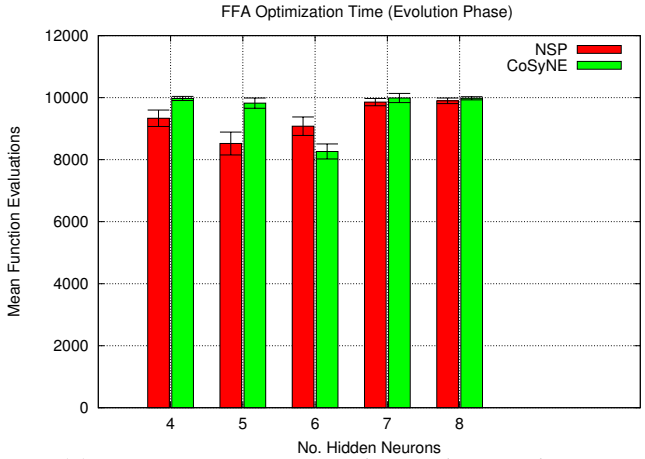


(b) Total optimization time = Initialisation + Evolution Phase

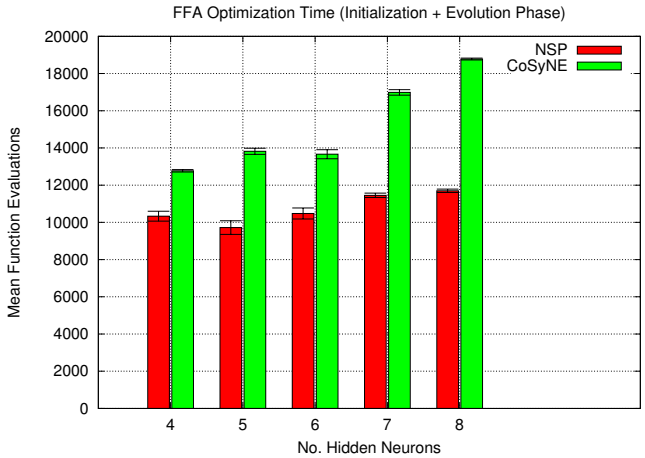


(c) Success rate in the T4 Problem

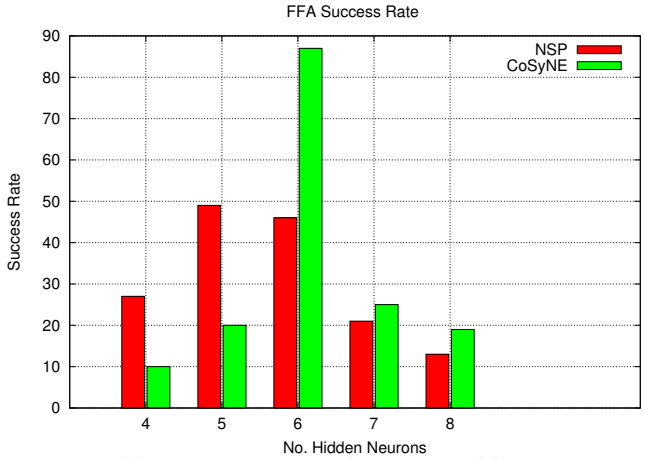
Figure 3.23: The performance of NSP and CoSyNE on different number of hidden neurons for the T4 problem. The optimization time in terms of the average number of function evaluations (Evolution Phase) is shown in (a), the total optimization time which include the initialisation and Evolution Phase is shown in (b) and the number of successful runs is shown in (c).



(a) Optimization time in the Evolution Phase



(b) Total optimization time = Initialisation + Evolution Phase



(c) Success rate in the FFA Problem

Figure 3.24: The performance of NSP and CoSyNE on different number of hidden neurons for the FFA problem. The optimization time in terms of the average number of function evaluations (Evolution Phase) is shown in (a), the total optimization time which include the initialisation and Evolution Phase is shown in (b) and the number of successful runs is shown in (c).

shown in (c). The results are similar in the case of the T2 problem shown in Figure 3.21, except that for 4 hidden neurons, the performance of NSP and CoSyNE is similar in the evolution phase (a). However, NSP shows better performance when the initialisation phase is added to the evolution phase (total time) for 4 hidden neurons in (b).

In the T3 problem shown in Figure 3.22, the evolution phase in (a) shows the performance of NSP is similar to CoSyNE for 4 and 7 hidden neurons. In (b), NSP gives better performance for the total time. In (c), NSP generally gives better performance except for 4 hidden neurons which has also affected the total optimisation time for this case in (c). This is a special case where CoSyNE has performed better due to its problem decomposition method. This shows that the nature of the T3 problem is different when 4 hidden neurons have been used in the hidden layer where CoSyNE seems more appropriate.

In the T4 problem shown in Figure 3.23, NSP gives better performance than CoSyNE in general as shown in the total optimisation time in (b). In (c), NSP has given better success rate except for 7 hidden neurons where the problem decomposition of CoSyNE was better due to the size of the network. Similar results were given for the T3 problem for 4 hidden neurons, this also reveals how the nature of the neuron network changes in terms of the degree of non-separability due to the number of hidden neurons. Due to the changing nature of the problem, adaptation of the problem decomposition method is necessary.

In the FFA problem shown in Figure 3.24, the performance of NSP is weaker than CoSyNE for 6 hidden neurons and similar for 7 and 8 hidden neurons as shown in (a). NSP outperforms CoSyNE in all the cases for the total time in (b). This is another case where the problem decomposition of CoSyNE showed better performance due to the size of the network.

Due to the success of CoSyNE in some cases, the results motivates the adaptation of problem decomposition where CoSyNE and NSP can be used together. This will be discussed and shown in Chapter 5.

3.3.3 Discussion

Cooperative coevolution has been used for the optimisation of separable and non-separable problems as discussed in Chapter 2. Naturally, cooperative coevolution performs better in separable problems, however, for non-separable problems, heuristic methods have been used to group interacting variables in separate subcomponents for large scale function optimization [233]. In the case of neuro-evolution, their architectural properties have been used to group the subcomponents. In this chapter, NSP has been introduced, which groups subcomponents according to the synapse-links that are connected to a neuron. Cooperative coevolution performs better in separable problems as they can be easily decomposed. Most problems fall in the partially-separable category and neural network are one of them.

The results reveal that good performance has been achieved when each sub-population is evolved for one generation in a round-robin fashion for the entire cycle of CoSyNE and NSP. In neural networks, interacting variables exist which determine the degree of non-separability. The degree of non-separability is dependent on the particular neural network architecture and application problem. The deep greedy search for a large number of generations for each sub-population has not shown good performance and gives an indication that recurrent networks on the given grammatical inference problems are partially-separable. This has also been the case of training feedforward networks as discussed in Section 3.3.1. In feedforward networks, NSP has been able to withstand the deep greedy search, which implies that they have been able to efficiently group the interacting variables. In the case of recurrent networks, a deep greedy search in NSP has shown similar performance for 1 to 5 generations. The performance deteriorates with deeper depth of search. This implies that the degree of non-separability is higher in recurrent networks when compared to feedforward networks. This is due to the application problem and the network architecture, i.e, in recurrent networks, feedback connections are present

which indicate that more interacting variables are present. This can be the main reason that a deeper greedy search for the case of using NSP for evolving recurrent networks has not been beneficial. been the case of CoSyNE in feedforward and recurrent networks which does not group interacting variables.

In feedforward networks, NSP provided better performance than CME due to the following reasons.

1. NSP provides more diversity than CME, hence it is less prone to local convergence. This is due to the difference in the number of subcomponents used by the two methods.

2. Synapse level has more diversity, but does not have any feature of grouping interacting variables. The neural network problem is partially separable and requires interacting variables to be grouped which is not possible through synapse level problem decomposition.

3. NSP groups the interacting variables in the output layer weights which is not done by CME. It has been shown in Section 3.1.1 that the output layer weights interact with each other and it is important to group them as done in NSP.

The performance of CoSyNE is weak when compared to NSP for training feedforward neural networks in pattern recognition problems shown in [27]. However, CoSyNE showed impressive results for pole balancing problems in [77]. The pole balancing problem is a control problem which does not use an error function such as the sum of squared error and hence the degree of non-separability would be different when compared to the analysis done in Section 3.1.1.

In T3, T4 and FFA problem, the performance of CoSyNE is better in some cases given the success rate in (c) of Figures 4.13 - 4.15. This has been mainly observed for larger network sizes where the problem decomposition of CoSyNE is more applicable, however, NSP does better in these cases given the total optimisation time (b).

The results show that the proposed NSP encoding gives better performance than CoSyNE given the total optimisation time in most cases. Note that these results are specific for grammatical inference problems. The NSP has shown to have the ability to effectively form the required states in the recurrent network during the learning process. The major advantage of NSP is that it can represent the same problem in a smaller number of subcomponents than the CoSyNE and at the same time it provides better optimization performance. This advantage further enables NSP to have fewer function evaluations in the initialisation phase as verified by the results in Table 3.7. The initialisation phase has been one of the reasons for the significant improvement in the total optimization time of NSP over CoSyNE in most cases. The other reason is due to the degree of non-separability exhibited by the different problem decomposition methods. Note that CoSyNE views the neural network as a fully separable problem as it used a separable subcomponent for each synapse. Canonical neuro-evolution with a single population views the network as fully non-separable, while ESP and NSP views the network as partially separable. The success of NSP shows that the given recurrent network problem is partially separable. The same can be said about feedforward networks for the given pattern classification problems.

The generalisation performance of NSP has been similar to the other methods for the case of feedforward and recurrent networks. This indicates that NSP has achieved the same solution quality with lower optimization time and better success rate which reflects robustness.

The NSP can be used for learning long-term dependency problems. This is due to the non-gradient requirement of neuro-evolution in optimising the weights of the network. The length of the strings or time lags does not matter to neuro-evolution. This is evident as the strings of length 15-25 in the Tomita 3 and 4 problem were successfully trained by NSP which would have not been possible with the back-propagation-through-time algorithm [226].

.

3.4 Chapter Summary

This chapter introduced a new problem decomposition method (NSP) for the cooperative neuro-coevolution of feedforward and recurrent neural networks. The method has been tested on pattern classification and grammatical inference problems for feedforward and recurrent neural networks, respectively.

The investigation began with a study for the cost of evaluation of each encoding scheme in the initialisation phase of the evolutionary process. The NSP represents the problem with fewer subcomponents when compared to CoSyNE.

An important question raised from this research was to evaluate the optimal depth of search in the subcomponents for the respective paradigms. The results show that the depth of search is significant for CoSyNE only. The depth of search for 1 generation only gives acceptable results in the CoSyNE algorithm in all the given problems. This was seen for both feedforward and recurrent neural networks. The results showed that NSP has been able to achieve the similar solution quality when compared to other methods in terms the generalisation performance. In general, NSP has shown better optimisation performance in terms of time and success rate than its counterparts for training feedforward and recurrent networks.

The next chapter deals with adaptation using local search in cooperative coevolution. The chapters ahead will mostly focus on the optimization performance in terms of the optimization time and success rate. This is due to the results from this chapter which indicated that the proposed method has been able to achieve the same level of generalisation as the other methods.

Chapter 4

Memetic Cooperative Neuro-evolution

This chapter presents a memetic cooperative coevolution method that employs local search. The proposed framework is used for training feed-forward and recurrent neural networks. The chapter begins with a brief overview of memetic algorithms and the motivation for the new memetic cooperative neuro-evolution method. The details of the method are then given with results and discussion.

4.1 Introduction

Cooperative coevolution has the feature of decomposing a problem using several sub-populations which provides greater diversity that enforces global search. Memetic computing provides further enhancement to evolutionary algorithms with local search (LS). The success of local search in memetic algorithms gives the motivation of using local search in cooperative coevolution. This chapter presents a memetic cooperative neuro-evolution method that utilises the strength of local search. The crossover-based local search is used as the local search method. The proposed method is called *crossover-based local search in cooperative neuro-evolution* (XLCC).

XLCC is used for training feedforward and recurrent neural networks. Pattern classification and grammatical inference problems are used for feedforward and recurrent networks, respectively. Crossover-based local search [147] is appropriate for local search in training recurrent neural networks as no gradient-information is needed in the search which has been a problem in learning long-term dependency problems.

In memetic algorithms, much research has been done in the relationship between the diversification using the population of candidate solutions and intensification using the local search. In the memetic cooperative neuro-evolution framework, the diversification is provided by the sub-populations in cooperative coevolution and the intensification is provided by the particular local search method. The *local search intensity* (LSI) and *local search interval* (LS-Interval) are important parameters that determine when and how long to apply local search in the evolutionary process. These parameters will be adjusted in order to balance diversification with intensification according to the network architecture and the nature of the problem. A heuristic for adapting the local search intensity during evolution is also presented.

4.1.1 Global and Local Search in Evolutionary Algorithms

Global search traverses over several neighbourhood of solutions while local search limits itself within a single solution neighbourhood. The neighbourhood $N(v)$ of a vertex v is the sub-graph that consists of the vertices adjacent to v (not including v itself) [225].

Local search seen is also seen as hill climbing and fine tunes or refines the solution. Evolutionary search methods begin with global search with large difference between candidate solutions in the population. As the search progresses, with evolutionary operators such as selection and recombination, the search points to a single solution neighbourhood and the candidate solutions are closer to each other. Local search is encouraged

towards the end of the search when the distance between the candidate solutions get smaller. The distance between the candidate solutions is often used as a termination criterion. The evolutionary algorithm is seen to be 'converged' when all candidate solutions are similar. The similarity between candidate solutions can be checked with some distance measure between the candidate solutions.

It is common to view search algorithms that work with a single candidate solution as local search and evolutionary algorithms which have a population of candidate solution as global search. However, this is not true. It is not the number of candidate solutions, but the way the new solutions are formed and accepted that determines the difference between global and local search. Simulated annealing is considered as a global search technique where a single solution is used [113, 223]. In simulated annealing, global search is enforced by the way new solutions are formed and accepted. In the beginning of the search, weaker solutions are accepted according to a probability distribution and towards the later stages, only the improved solution is accepted which enforces local search.

Therefore, global and local search can be summarised as follows.

1. Global search is employed if the search traverses several candidate solution neighbourhoods. In evolutionary algorithms, the populations with large distance between the candidate solutions and evolutionary search operators enforce this. In simulated annealing, the criteria of accepting candidate solutions where weaker solutions are at times accepted encourages global search.

2. Local search is enforced when only a single solution neighbourhood is considered. The solutions are accepted only if improved, and evolutionary search operators must provide refinement so that the solution quality is enhanced.

Global search is also known as diversification as the search considers diverse range of candidate solutions. Local search is also called intensifica-

tion as the solution is intensified or refined in terms of accuracy. The candidate solution(s) selected for local search is known as meme in memetic algorithms.

In the proposed memetic cooperative neuro-evolution, crossover-based LS is used. This LS method has been introduced as a local search method by [147]. Crossover-based LS employs a small population of candidate solutions. When LS is needed, the elite solutions from the global search population are added to the local search population which contains previous elite solutions. The LS population is evolved with efficient crossover (such as parent-centric crossover [42]) operator which has properties for refining the solution.

4.2 Memetic Cooperative Neuro-evolution

In the literature review given in Chapter 2, the discussion on how the neural network learning problem can be decomposed into subcomponents has been given.

Memetic algorithms have mainly been developed using evolutionary algorithms that have a single population of individuals. In the case of building a memetic computation framework for multiple sub-populations in cooperative coevolution, the computational cost of having individual local search for each sub-population has to be considered. The individual in a sub-population that undergoes local search only represents a subset of the large problem. In order to apply local search, the respective individual has to be concatenated with the best individuals in the rest of the sub-populations. Therefore, given n sub-populations, n local searches are required which adds to the computational cost as shown in Figure 4.1.

This section give the details of a new method which efficiently takes the advantage of the local search and it takes in account, the computational cost of having a separate local search for every sub-population. Rather than employing a local search for each sub-population, the pro-

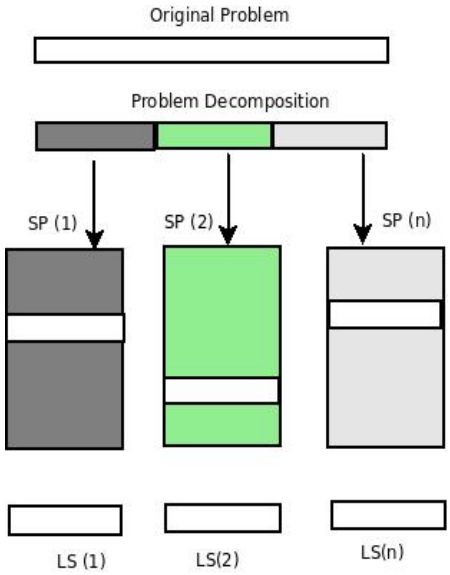


Figure 4.1: Problem faced by cooperative coevolution in employing n local searches (LS) to each sub-population (SP)

posed framework employs local search only when a *cycle* in the sub-population is complete. The completion of a cycle indicates that all the respective sub-populations have been evolved for a given number of generations in a round-robin fashion.

The meme is the individual that goes through local search. The details of the memetic cooperative neuro-evolution method is given in Algorithm 6 . The algorithm assumes that it has been given the best parameters for the evolutionary algorithm such as the sub-population size, crossover and mutation rate.

The algorithm begins by encoding the neural network into the sub-population according to the respective cooperative coevolution encoding scheme (either Synapse or Neural level). The specific encoding scheme for this work is NSP, which was introduced in Chapter 3.

The crossover-based local search employs a population of few individuals, which is also referred as the local search population. The goal of

Alg. 6 Memetic Cooperative Coevolution

– Encode the neural network using an appropriate encoding scheme
– Randomly initialise all sub-populations
– Cooperatively evaluate each sub-population

while NOT termination **do**
  for LS-Interval **do**
    for each sub-population **do**
      for depth of n generations **do**
        Create new individuals using genetic operators
      end for
    end for
  end for

  – Concatenate the best individuals from each sub-population into meme M
  – Local search on M for LSI

  i) Decompose M according to the respective sub-populations
  ii) Replace the worst individuals of the respective sub-populations with decomposed M
end while

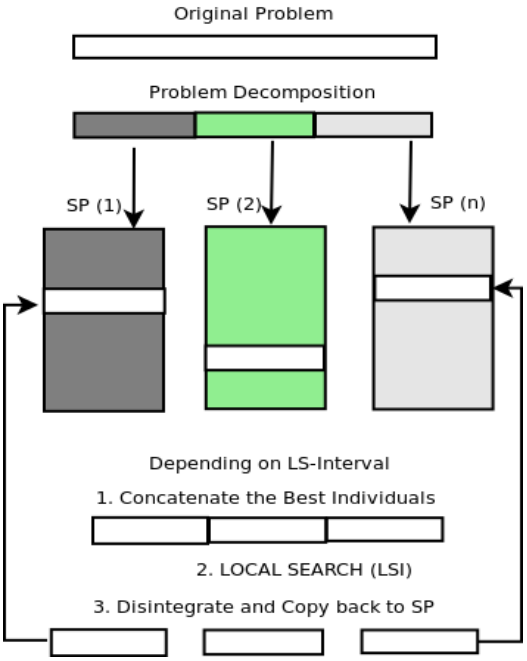


Figure 4.2: The memetic cooperative neuro-evolution framework employs LOCAL SEARCH after concatenating the best individuals from each sub-population (SP) at the end of each cycle.

the cooperative coevolution sub-populations is to promote diversity [168], and the local search population provides intensification. All the individuals of the respective sub-population are initialised with random real values. Each individual chromosome is then concatenated with the best individuals of the rest of the sub-populations and then encoded into a neural network and evaluated as done in [168].

The algorithm proceeds as a standard evolutionary algorithm which employs genetic operators such as selection, crossover and mutation to create new offspring for all the sub-populations. Each sub-population is evolved for a depth of search of n generations in a round-robin fashion and the cycle is completed. This process is repeated according to the LS-interval. After the specified LS-Interval has been reached, the best individuals from all the sub-populations are concatenated into a meme which is further refined as shown in Figure 4.2. The meme replaces the weakest individual in the local search population. The meme is then refined using the local search population for a given number of generations as defined by the LSI. The refined meme is then disintegrated and copied to the respective sub-populations. The refined meme replaces the weakest individual in each of the sub-populations. Note that even if the refined meme is not improved, it replaces the weakest individuals as it may have features that will be used later in evolution. However, the best memes in the local search population are always retained. Although crossover-based local search is used as the designated method, the framework can employ any other local search method.

Meta-Lamarckian learning can also be used in this framework. In meta-Lamarckian learning, several local searches can be employed and the suitable memes are chosen from the pool of local searchers as discussed in [161]. However, for the case of neural network training, where function evaluation is costly, employing multiple local searches may not be practical for the given problem. Nevertheless, it may be suitable for other types of problems. use an efficient evolutionary algorithm in the sub-

populations of the cooperative coevolution framework which is described in detail in the next sub-section.

4.2.1 G3-PCX for Crossover-based Local Search

As discussed in the literature section, the use of EA for local search has been effective [109, 131, 147]. In the XLCC, the G3-PCX evolutionary algorithm [42] with a small population size is used as the EA for crossover-based local search. The G3-PCX is also used as the EA for the sub-populations of cooperative coevolution. The parent-centric crossover operative of the G3-PCX has features to provide good local search, therefore, it needs large population size (of more than 90) even for small 2 dimensional problems as discussed in [171]. A small population size for the G3-PCX will ensure that it becomes local search intensive and therefore it is used as a local search method.

The individuals in the population of the crossover-based local search are randomly seeded in the beginning of the evolutionary process. The cooperative coevolution sub-populations are seeded at the same time. During the evolutionary process, the cooperative coevolution sub-populations transfer the meme, which is the best solution, to the crossover-based local search population. This is done by concatenating the best solutions from all the sub-populations as discussed in the previous section. This transfer is also dependent on the local search interval. Once the meme is transfered, the local search population is evolved according to the local search intensity. This population consists of the current meme and other candidate solutions left from the previous time when this population was used.

Once the local search population has been evolved according to the local search intensity, the best solution is transferred to the sub-populations of the cooperative coevolution. The remaining individuals in the local search population are kept and used in future local search evolution. This is done in order to maintain diversity, i.e. these individuals can be used

to produce more fit offspring with the next meme that contains the best solution from cooperative coevolution.

4.3 Simulation

This section presents an empirical study on the proposed memetic cooperative neuro-evolution method for training feedforward and recurrent networks. The simulation is done to evaluate the performance of the proposed memetic cooperative coevolution method. The simulation evaluates the relationship between the local search interval and intensity and presents a heuristic to adapt the local search intensity during evolution.

The G3-PCX evolutionary algorithm [42] is employed in the respective CC frameworks and also used for crossover-based local search. The crossover-based local search has a fixed population of 20 individuals. The cooperative co-evolutionary framework has 100 individuals in all sub-populations. These parameters were determined in trial experiments.

The G3-PCX employs the mating pool size of 2 offspring, the family size of 2 parents, and the generation gap model for selection for all the sub-populations in cooperative coevolution and the population for local search. This set-up has been used in [42] and has shown good results for general optimisation problems. This set-up has also been used in Chapter 3. The sub-populations are seeded with random real numbers in the range of [-5, 5] in all experiments. The memetic cooperative neuro-evolution method employs the NSP encoding scheme for evolving feedforward and recurrent networks, respectively . Other encoding schemes from the literature can also be used.

4.3.1 Feedforward Neural Networks

Classification Problems and Configuration

The n-bit parity is a classical problem and has been used to evaluate neural network training algorithms [94]. The 4-bit-parity problem is used where an even parity is determined by the even number of 1's in the input. The *Wine*, *Iris*, *Breast-Cancer* and *Heart-Disease* classification problems are also used to evaluate the performance of the proposed method [6].

Table A.1 in the Appendix gives further details of the problems. The maximum training time for Iris, Wine and Breast-Cancer problems are fixed to 15000, the 4-bit-parity and Heart-Disease disease problem has 30000 and 50000 function evaluations, respectively. Note that these values have been changed in order to make the problem more difficult when compared to Chapter 3. Lowering the maximum limit of the optimisation time will have impact in terms of the success rate of convergence. Although optimisation is the focus of this section, the generalisation performance is also reported. 50 independent runs are done for each case in the experimental set-up.

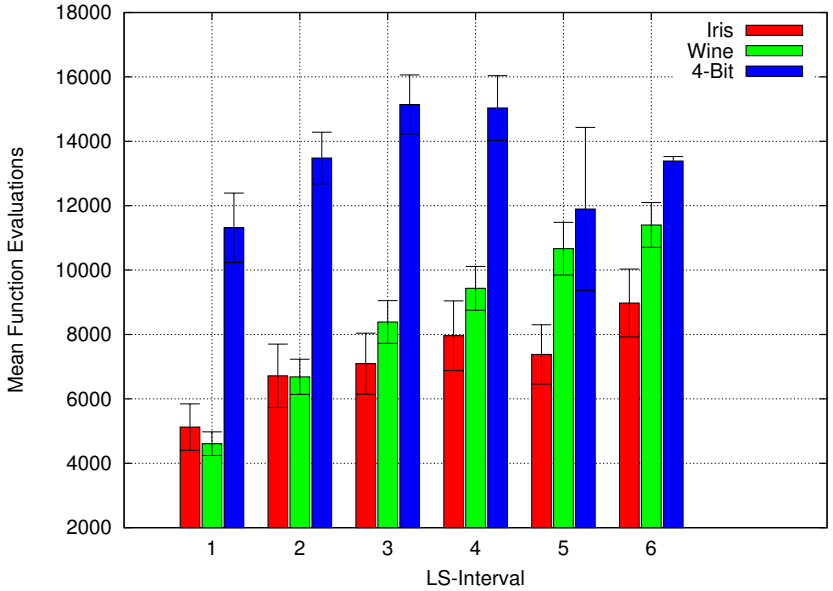
Relationship Between Local Search Interval and Intensity

It is important to establish the relationship between the interval and intensity of local search in order to take full advantage for the memetic cooperative coevolution framework. The local search interval determines when to apply local search, i.e, after how many consecutive cycles of undergoing cooperative coevolution. We used a fixed local search intensity of 10 generations to evaluate the local search interval. We used the 4-Bit Parity, Iris and Wine classification problems for this experiment. We employed 5 neurons in the hidden layer for these problems and used information for termination from Table A.1. The results are shown in Figure 4.3. A total of 50 experimental runs have been done.

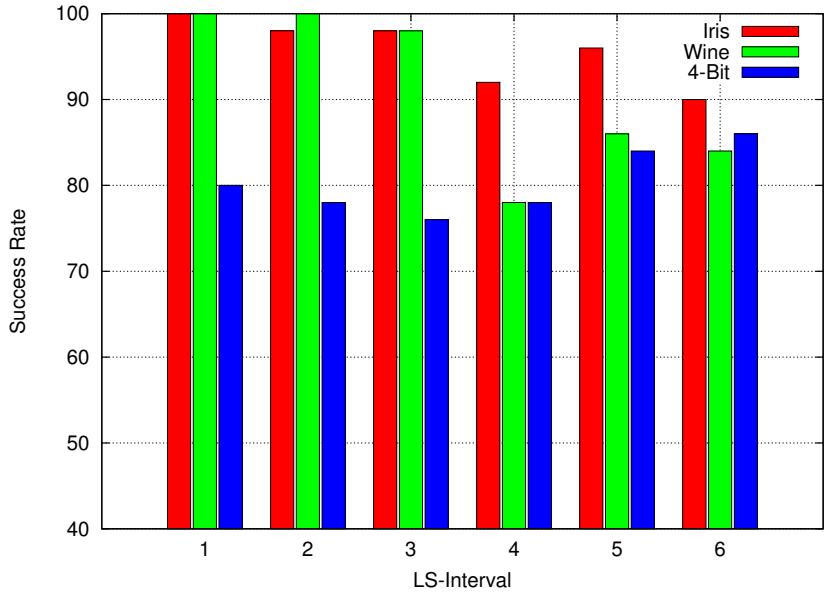
Figure 4.3 shows that the local search interval of 1 for the three prob-

lems gives the best performances in terms of the least function evaluations in (a) with high success in (b). The performance deteriorates when the interval is increased. The performance deteriorates when the LS-Interval is increased. When the LS-Interval is increased, more diversity is enforced as the evolutionary process features evolution from the sub-populations of cooperative coevolution. In the XLCC, more diversity is encouraged by allocating more time to CC for evolution. CC through the sub-populations has more diversity when compared to the local search population. When the LS-Interval is increased in XLCC, more time is allocated for CC based evolution.

It has been observed that the LS-Interval of 1 gives the best performance. One reason for this can be that the local search population provides a means for co-adaptation for the sub-populations in cooperative coevolution. With the local search population, the search is carried on individuals that represent the whole solution rather than the sub-solutions in the sub-populations. Another reason for this is due to the intensification process provided by the local search population.



(a) Function Evaluations for evaluating the LS-Interval



(b) Success Rate for evaluating the LS-Interval

Figure 4.3: The evaluation of the LS-Interval using the 4-bit-parity, Iris and Wine classification problems. The LS-Interval of 1 shows the highest success rate and least number of function evaluations for all three problems.

Performance of the Local Search method

In the field of hybrid and memetic evolutionary algorithms, it is common to see that the performance of local search methods are not evaluated independently. It is important to find if the local search method can converge on its own and what contribution the local search has on the overall convergence of the hybrid algorithm.

To evaluate the crossover-based local search method, some experiments were done using the 4-Bit, Iris and Wine classification problems. The local search method employs the G3-PCX algorithm with a population of 20 individuals. All of the problems employed 5 neurons in the hidden layer with a maximum of 15,000 function evaluations for the Iris and Wine problems and 30,000 function evaluations for the 4-Bit problem. The results revealed that out of 50 interdependent experimental runs, none of the runs converged within the maximum time for the three problems. All runs suffered from local convergence which indicates that the small population size was not sufficient for these problems. The small population size enforces local search in the G3-PCX.

The strengths and limitations of the G3-PCX has been discussed in Section 2.3.2 of Chapter 2. It is important to note that the G3-PCX algorithm requires a large population size (of at least 100 individuals) in order to maintain diversity in the generalized generation gap (G3) model [171]. Furthermore, the G3-PCX has limitations in multi-modal problems. Training neural networks is a type of multi-modal problem and the small population size in the G3-PCX makes it hard to converge.

Adaptation in Local Search Intensity

In the previous subsection, it has been established that the local search interval of 1 gives the best performance for XLCC. It is important to use the right local search intensity. In the evolutionary process, global search is required in the initial stages and local search in later stages. Hence the local

search intensity should increase during the later stages. In consideration, a method for determining the local search intensity is shown in Equation 4.1

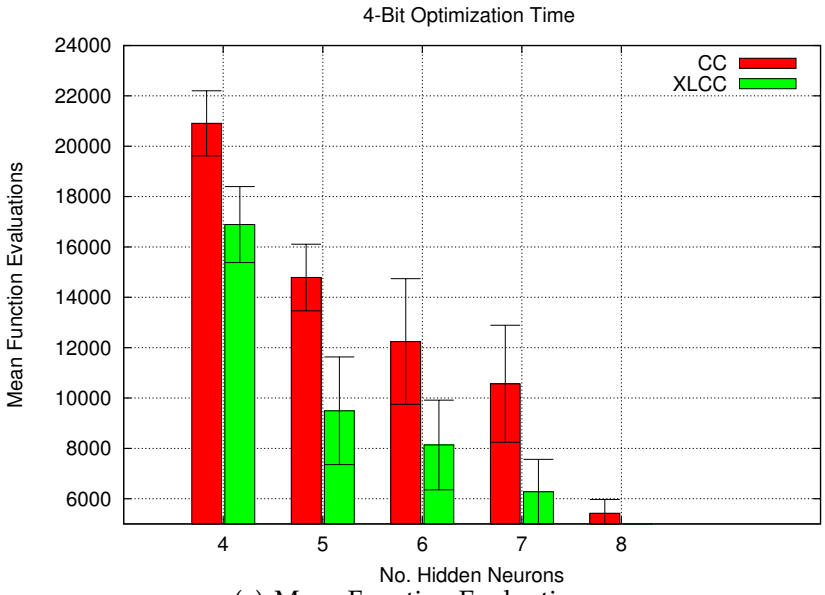
$$LSI = 1 + (\frac{t}{m} * k) \tag{4.1}$$

where, t is the number of function evaluations, m is the maximum number of function evaluations and k is a constant which specifies the maximum intensity of local search to be employed in the final stages. This heuristic ensures that the intensity of local search increases with the number of function evaluations. $k = 30$ is used in all the problems; this value was determined in trial experiments, it is an initial condition for the given problems. This value may vary for other types of problems.

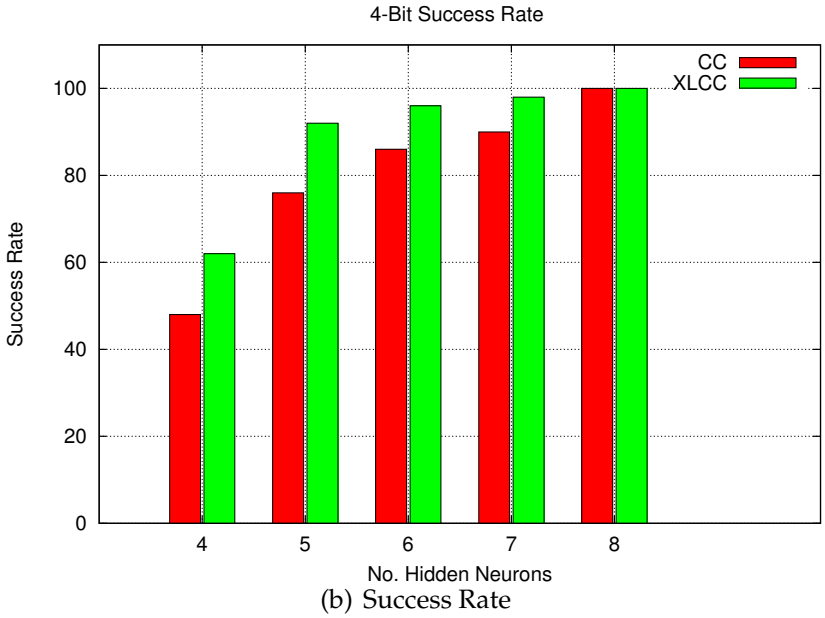
Table 4.1: Generalisation Performance

Iris			**Wine**		
Hidden	CC	XLCC	Hidden	CC	XLCC
4	$93.67_{0.84}$	$94.90_{0.77}$	4	$92.96_{1.06}$	$93.35_{1.10}$
5	$94.30_{0.78}$	$94.80_{0.70}$	5	$94.20_{0.87}$	$93.85_{0.89}$
6	$95.75_{0.61}$	$95.56_{0.60}$	6	$94.35_{0.93}$	$93.32_{0.98}$
7	$95.85_{0.56}$	$95.75_{0.58}$	7	$93.55_{0.92}$	$94.20_{0.96}$
8	$95.51_{0.63}$	$95.75_{0.64}$	8	$92.95_{1.07}$	$93.85_{0.94}$
Heart			**Cancer**		
Hidden	CC	XLCC	Hidden	CC	XLCC
7	$79.71_{0.96}$	$79.17_{0.69}$	5	$96.30_{0.52}$	$96.83_{0.36}$
8	$81.35_{1.15}$	$81.48_{0.48}$	6	$96.22_{0.46}$	$96.32_{0.41}$
9	$81.05_{0.96}$	$81.22_{0.67}$	7	$95.97_{0.50}$	$96.10_{0.36}$
10	$81.11_{1.01}$	$81.51_{0.46}$	8	$95.84_{0.35}$	$96.05_{0.44}$
11	$79.72_{0.92}$	$81.19_{0.44}$	9	$95.63_{0.41}$	$96.07_{0.41}$

The heuristic proposed in Equation 4.1 is used in Algorithm 6 with a local search interval of 1 for training feedforward networks. The details are given in Table A.1. The results are given in Figures 4.4 - 4.8 where a comparison of the memetic cooperative coevolution framework (XLCC)

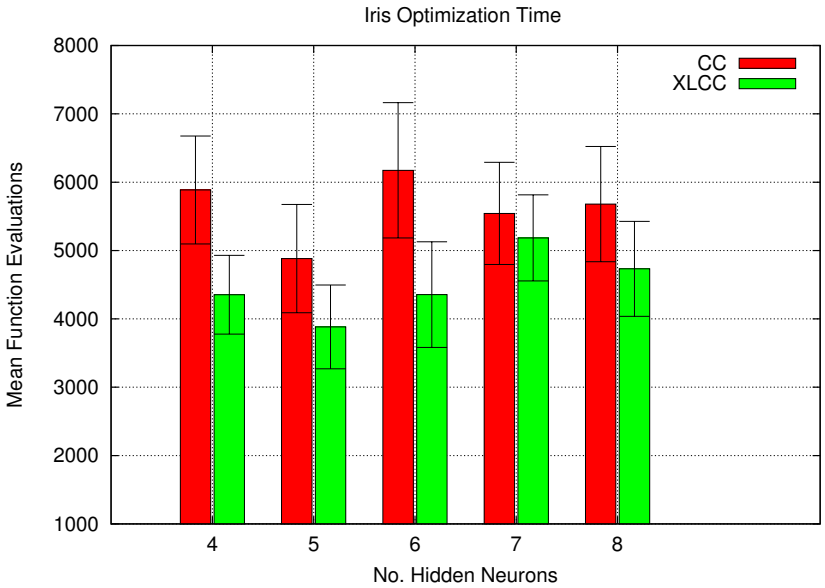


(a) Mean Function Evaluations

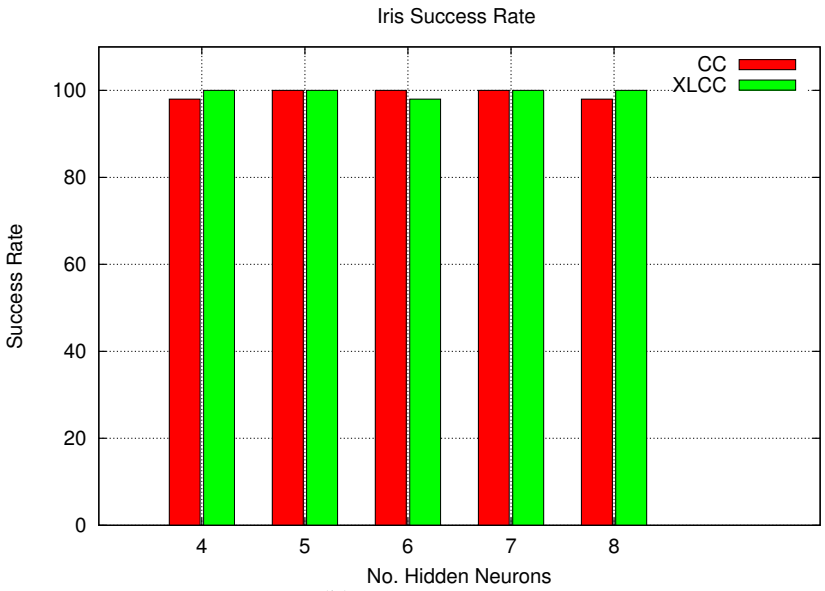


(b) Success Rate

Figure 4.4: The 4-Bit parity problem.

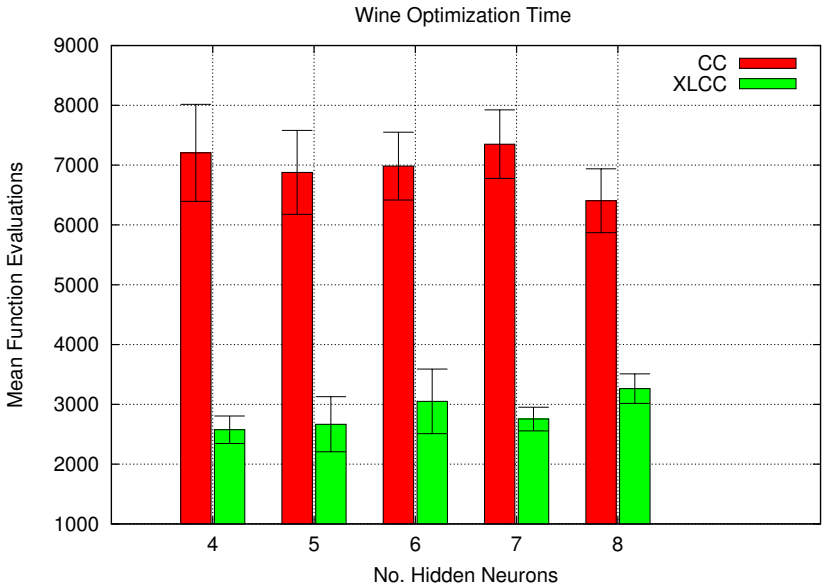


(a) Mean Function Evaluations

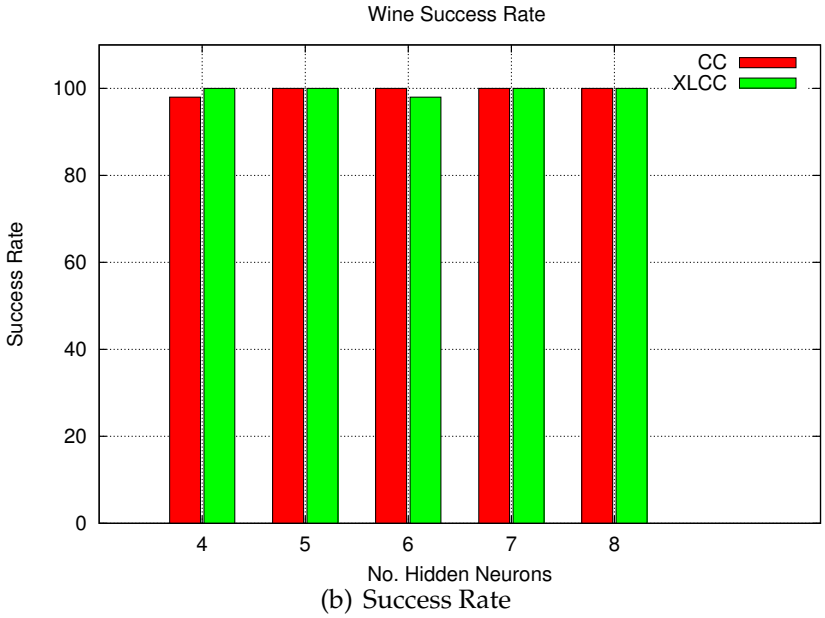


(b) Success Rate

Figure 4.5: The Iris classification problem.

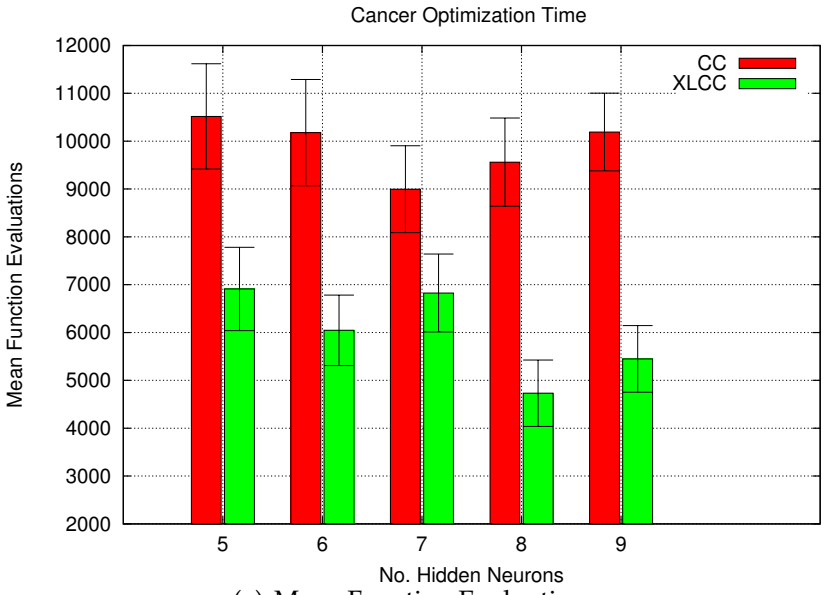


(a) Mean Function Evaluations

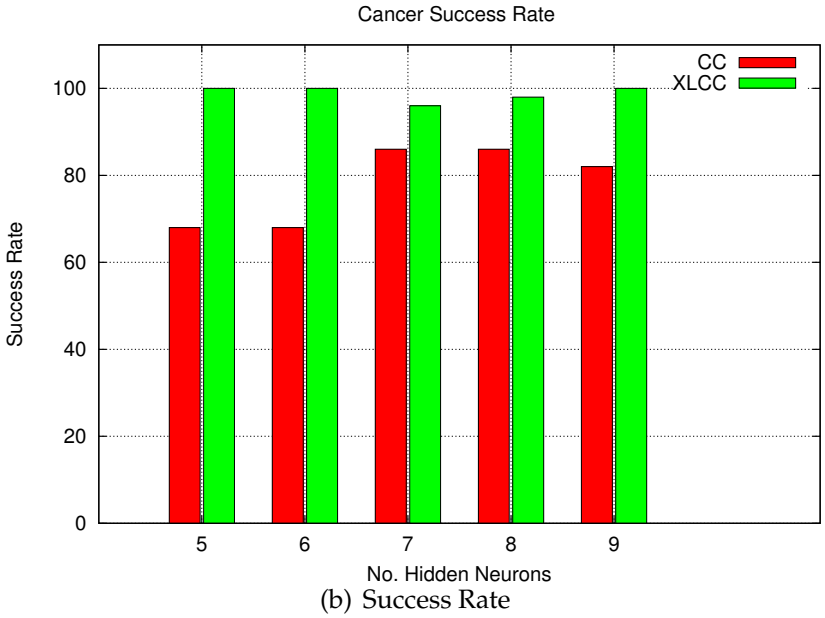


(b) Success Rate

Figure 4.6: The Wine classification problem.

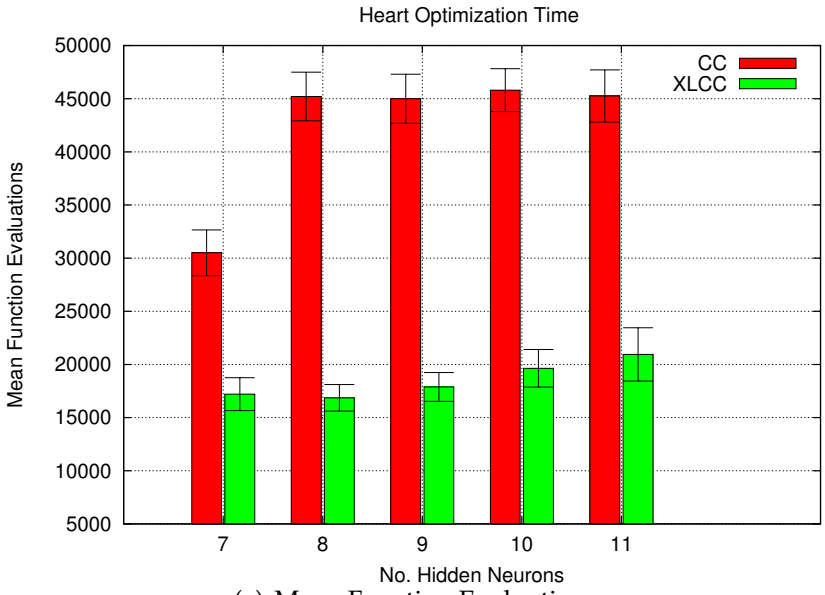


(a) Mean Function Evaluations

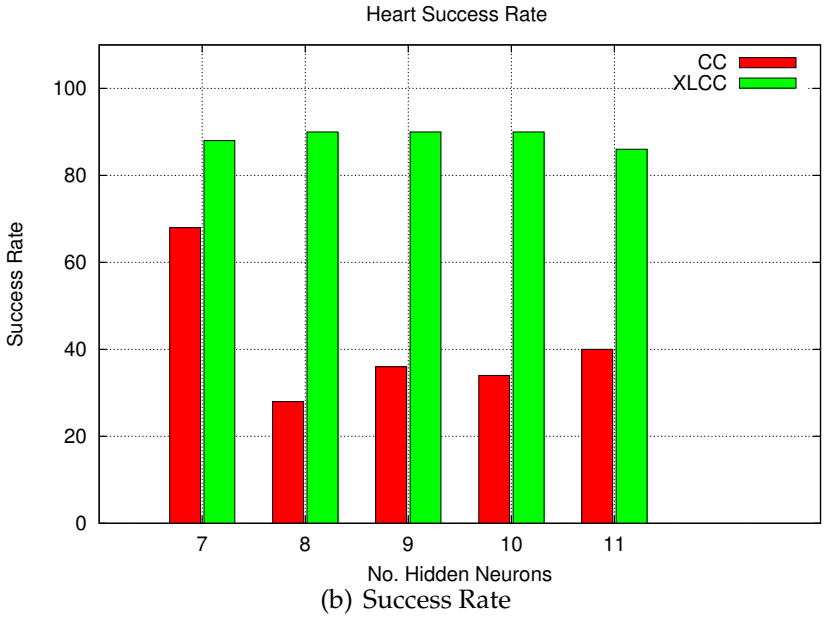


(b) Success Rate

Figure 4.7: The Breast Cancer classification problem.



(a) Mean Function Evaluations



(b) Success Rate

Figure 4.8: The Heart Disease classification problem.

with conventional cooperative coevolution (CC) is given. Note that the NSP based encoding scheme with G3-PCX is used in both methods.

Table 4.1 gives the details of the generalisation performance of the two algorithms. The results shown here indicate that there is not a major difference in the generalisation performance. Note that only the successful runs have been included in the test for genralisation as done in Chapter 3. The 4-Bit problem is not tested for generalisation as 100 % of its data was used for training.

The results in Figures 4.4 - 4.8 show that XLCC has achieved improved performance given different number of hidden neurons for all the problems. In these problems, higher success rate and lower optimisation time is taken by XLCC in comparison to CC which reflects on better scalability and robustness.

4.3.2 Recurrent Neural Networks

In this subsection, the performance of XLCC is evaluated for training recurrent neural networks on grammatical inference problems. The datasets for the FFA and Tomita problems (T1-T4) were generated as done in Section 3.3.2. In this case, in the Tomita problems, the strings of length 10 to 15 were used. This was done in order to have a smaller dataset to save computational time, as in this section, more emphasis is given in testing different parameters of the proposed methods other than the learning ability of RNNs.

Local Search Interval and Intensity

The FFA problem employs 5 neurons in the hidden layer. 2 neurons in the hidden layer for T2 and 3 neurons for the hidden layer for T3 and T4 are used. The maximum number of function evaluations in T2, T3 and T4 is 2000, 5000 and 5000, respectively. The FFA problem has 7000 has the maximum number of function evaluations.

Figures 4.9 and 4.10 give the results of the behaviour of XLCC on different LS-Interval for the 4 problems. 95 % confidence interval for 100 experiments is shown as error bars in the histograms. The fixed LSI of 8 generations is used. The figure shows that the interval of 1 gives the best results in terms of the minimum function evaluations in Figure 4.15(b) with greater success rates in Figure 4.15(a). The LS-Interval higher than 1 require a greater number of function evaluations. The same behaviour is seen in Figure 4.10.

Adaptive Local Search Intensity

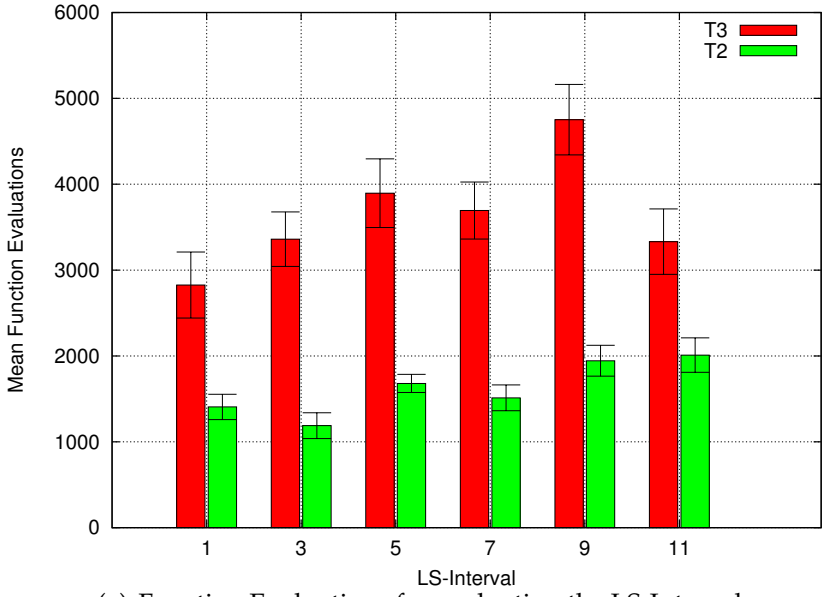
In the previous subsection, it has been established that the local search interval of 1 gives the best results. An adaptive method for determining the local search intensity as shown in Equation 4.1 is used where $k = 30$ for all the problems.

In these experiments, the maximum number of function evaluation for T1 and T2 are 2000. T3, T4 and T5 use 5000. All problems use different number of hidden neurons to demonstrate scalability and robustness.

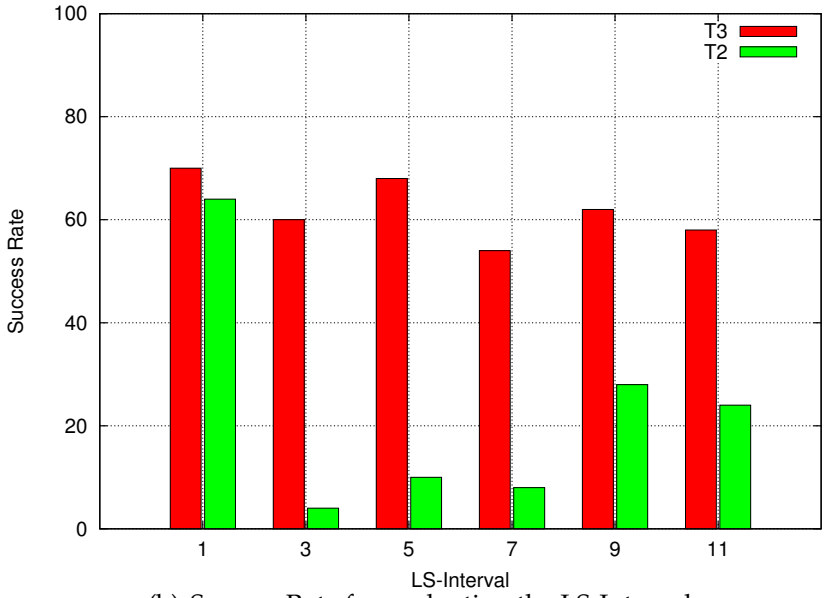
The results are shown in Figures 4.11-4.15 where the 95 % confidence interval for 100 experiments are shown as error bars in the histograms. The comparison of XLCC with standalone cooperative coevolution (CC) shows that XLCC has given better performance in most cases in terms of the optimisation time given by the number of function evaluations and the success rate. The only case where CC has done better (although not significant) is in Figure 4.13, where 4 hidden neurons are used. In this case, it seems local search has not been significantly beneficial.

4.3.3 Discussion

The results in general show that the LS-Interval of 1 gives the best performance in all the problems used for evolving feedforward and recurrent networks. This indicates that the local search has to be applied most fre-

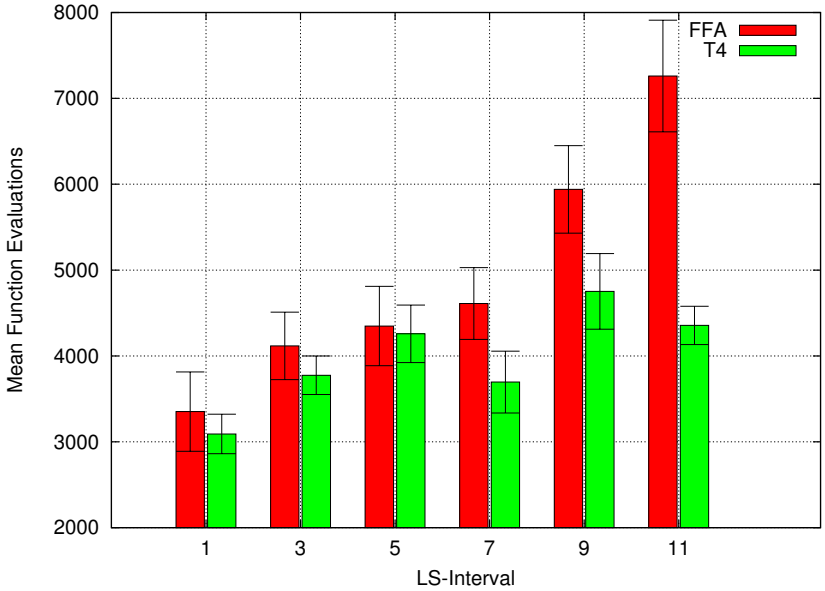


(a) Function Evaluations for evaluating the LS-Interval

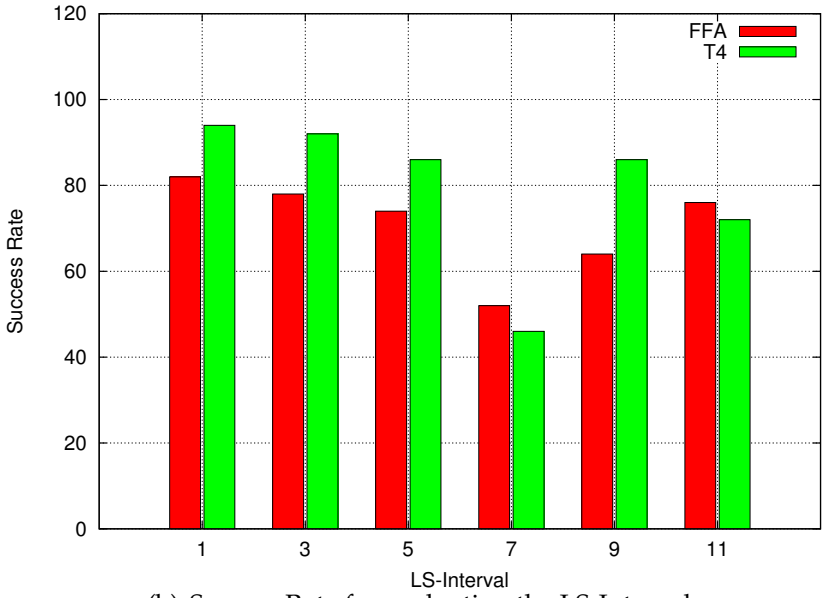


(b) Success Rate for evaluating the LS-Interval

Figure 4.9: The evaluation of the LS-Interval for the T2 and T3 grammatical inference problems. The LSI of 8 generations is used as a fixed parameter in all problems. The interval of 1 shows the highest success rate and least number of function evaluations for all problems.

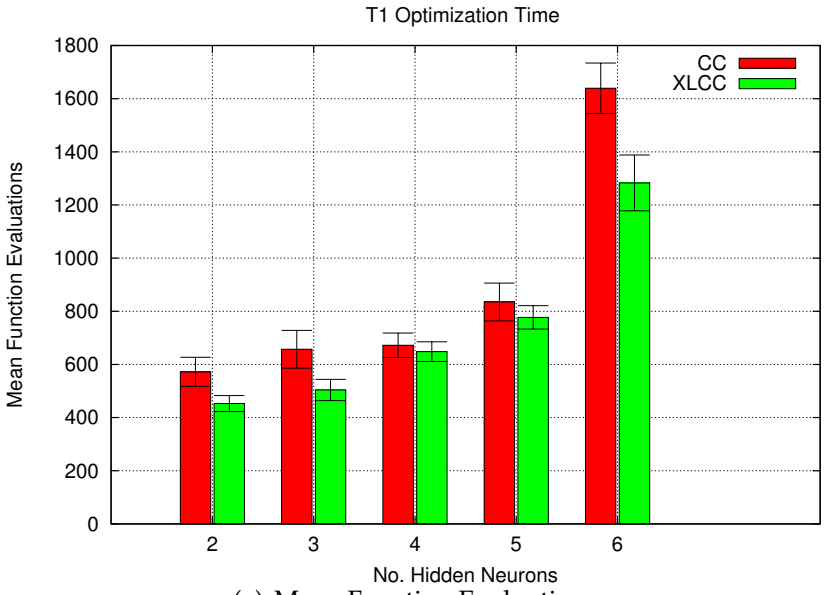


(a) Function Evaluations for evaluating the LS-Interval

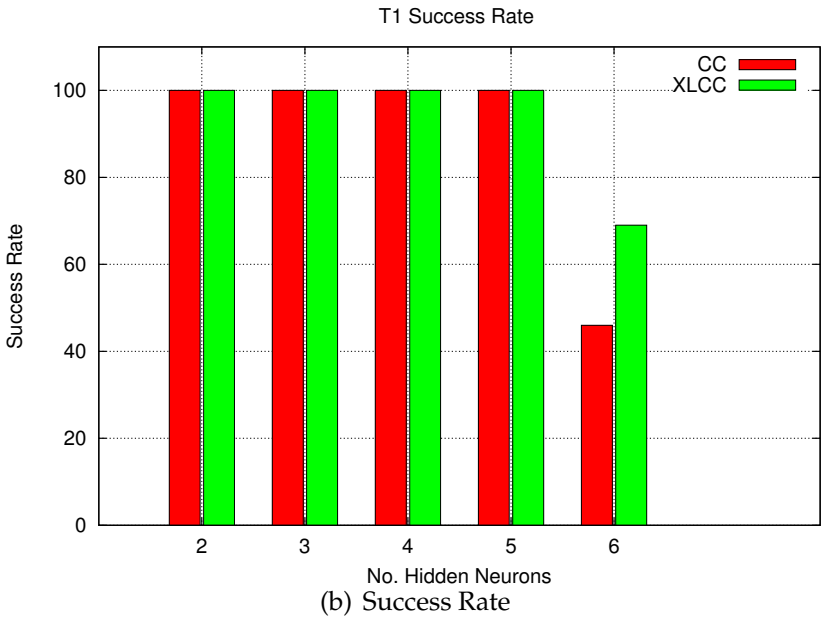


(b) Success Rate for evaluating the LS-Interval

Figure 4.10: The evaluation of the LS-Interval for the FFA and T4 grammatical inference problems. The LSI of 8 generations is used as a fixed parameter in all problems. The interval of 1 shows the highest success rate and least number of function evaluations for all problems.

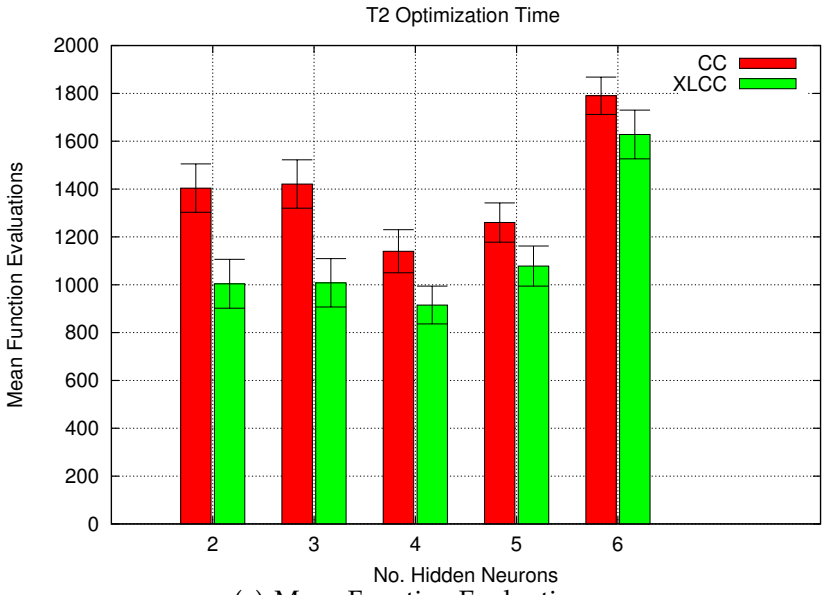


(a) Mean Function Evaluations

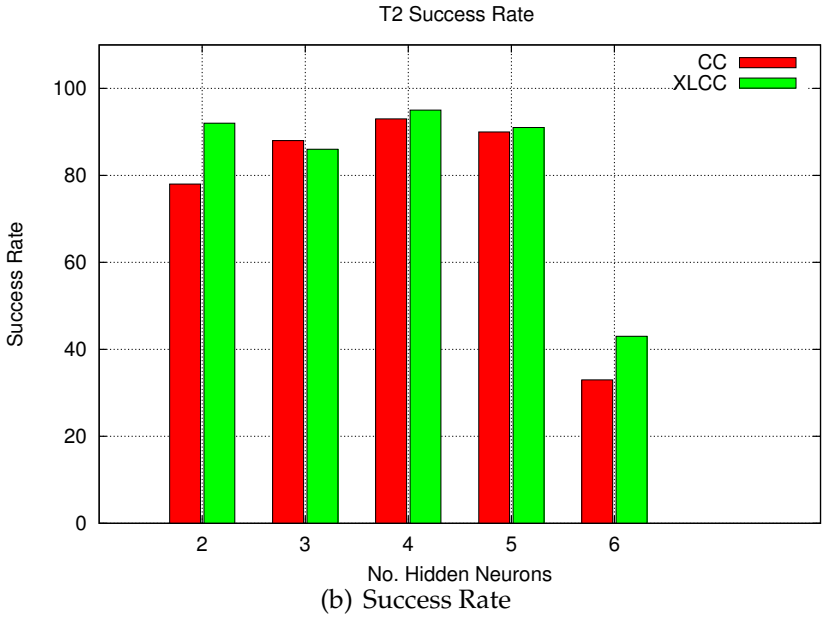


(b) Success Rate

Figure 4.11: The T1 problem.

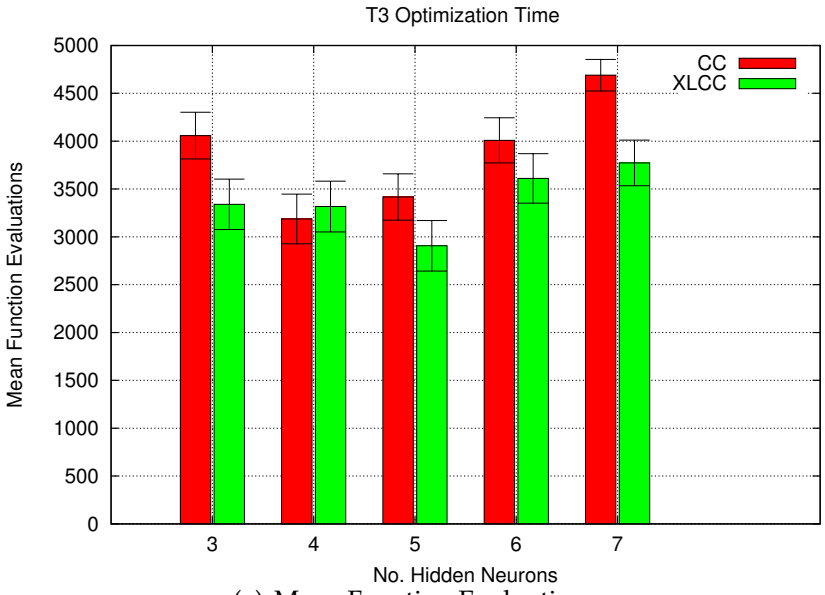


(a) Mean Function Evaluations

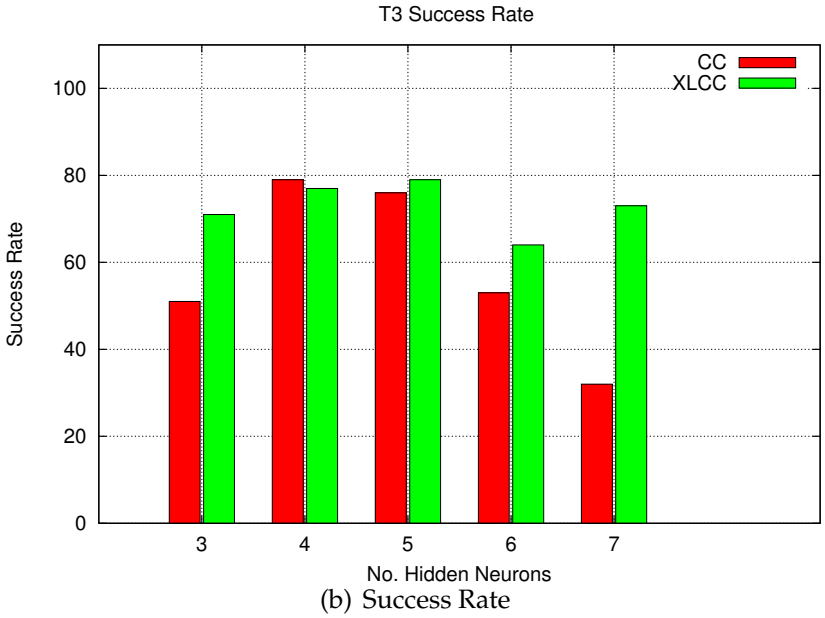


(b) Success Rate

Figure 4.12: The T2 problem.

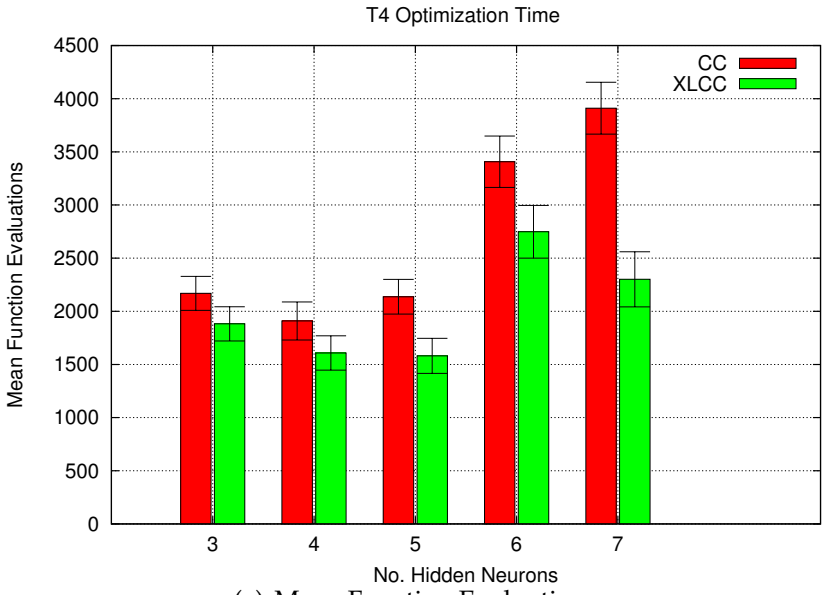


(a) Mean Function Evaluations

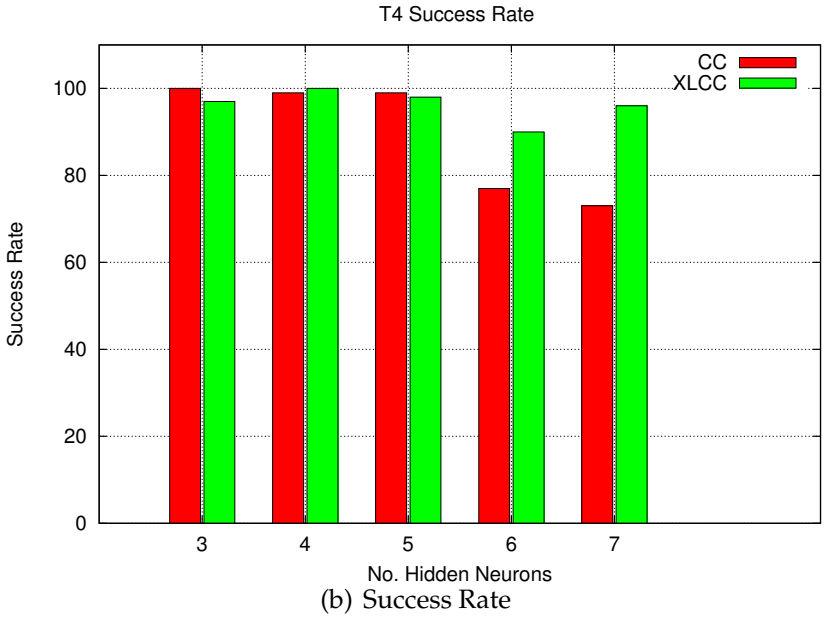


(b) Success Rate

Figure 4.13: The T3 problem.

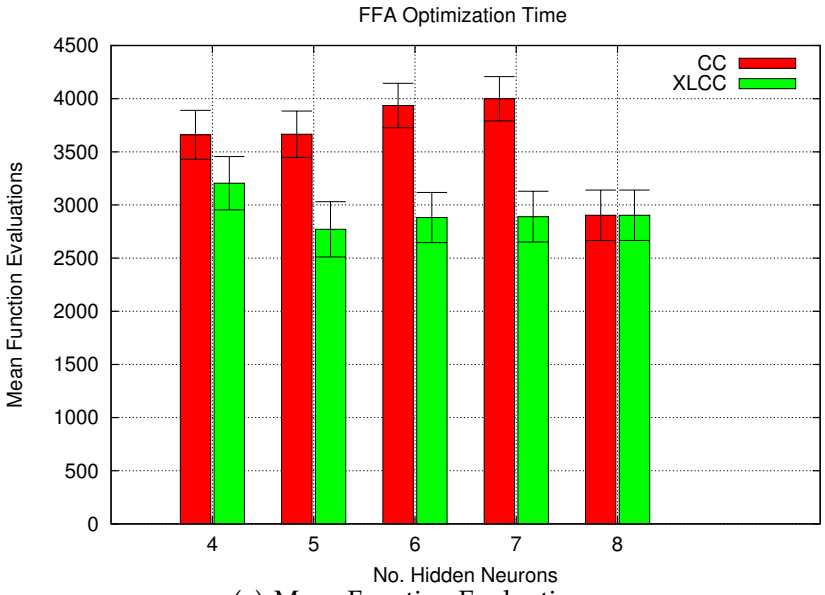


(a) Mean Function Evaluations

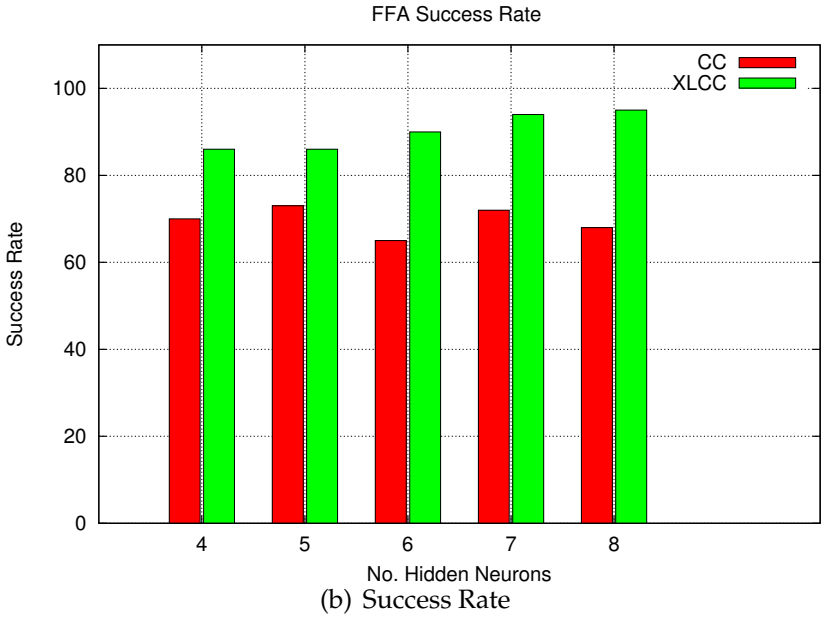


(b) Success Rate

Figure 4.14: The T4 problem.



(a) Mean Function Evaluations



(b) Success Rate

Figure 4.15: The FFA problem.

quently. The memetic framework has to take maximum advantage of local search after every cycle in cooperative coevolution in order to balance global and local search.

The results also show that local search is important and its intensity is dependent on the nature of the problem. The heuristic which scales the LSI has shown to be effective for most of the problems.

In general, comparison of the memetic cooperative framework with standalone cooperative coevolution shows improved performance in most of the given problems. The generalisation performance in Table 4.1 shows that there has not been a major difference in the performance of the two methods which implies that the memetic cooperative coevolution framework can achieve similar solution quality with better robustness and scalability. The proposed framework has the feature of efficiently utilizing local search without adding to the overall computational cost in terms of function evaluations. This indicates that it is important to employ local search in cooperative coevolution for training feedforward and recurrent neural networks.

The local search population has also provided the means for applying co-adaptation between the sub-populations of cooperative coevolution. Co-adaptation is necessary in cooperative coevolution, especially in the case when the problem is difficult to decompose. It is difficult to decompose neural networks into subcomponents as the interaction between the synapses depends on the network architecture, the stage of the evolutionary process and the nature of the problem, i.e training data. The local search population has also provided the means for applying co-adaptation between the sub-populations of cooperative coevolution. This population provides the means for selected individuals to be exchanged with different sub-populations using the crossover operation in the local search population.

4.4 Chapter Summary

The main problem in this chapter was to efficiently utilise local search in the respective sub-populations. This problem has been efficiently solved through the proposed framework which decomposes the locally refined solution and incorporates it into the sub-populations. The memetic framework progresses with a global search with cooperative coevolution and as a specified time is reached (in terms of the local search interval), the algorithm incorporates local search into the sub-populations. The memetic framework also provides the feature of co-adaptation among the sub-populations of cooperative coevolution using the local search population.

The results in general show improved performance of XLCC in feed-forward and recurrent networks. This implies that adaptation through local search is important in cooperative neuro-evolution. The next chapter presents further work in enhancing the adaptation properties in cooperative neuro-evolution by incorporating different problem decomposition methods at different stages of evolution.

Chapter 5

Adaptive Modularity in Cooperative Neuro-evolution

Adaptation during evolution has been an important focus of research in training neural networks. Cooperative coevolution has played a significant role in improving standard evolution of neural networks by organising the training problem into subcomponents or modules and independently solving them. This chapter introduces a method for the adaptation of the number of modules during evolution. The method is called adaptive modularity cooperative neuro-evolution and is used for training feedforward and recurrent neural networks.

The chapter begins with an introduction which overviews modularity and interacting variables. Afterwards, the adaptive modularity cooperative coevolution method is presented and then the simulation, results and discussions are given. The chapter ends with a summary of the findings.

5.1 Introduction

In the original cooperative co-evolutionary method, the problem is decomposed by having a separate subcomponent for each variable [169]. It was later found that the strategy was only effective for problems that are

separable [127]. In separable problems, there is no interdependency between the decision variables whereas in non-separable problems, interdependencies exist. It has been discussed in the literature, that a function of n variables is separable if it can be written as a sum of n functions with just one variable [165]. The parameters of a separable problem are called independent variables. Non-separable problems have interdependencies between decision variables as opposed to separable ones. Cooperative co-evolution naturally appeals to separable problems as there is no interaction amongst the subcomponents during evolution [183]. The decomposition of the problem influences its performance. In most problems, groups of interacting variables exist which determine the *degree of non-separability* which has been analysed in Section 3.1.1 of Chapter 3.

In order to take advantage of cooperative co-evolution, it is important to group interacting variables within a single subcomponent. However, it is difficult to identify the interacting variables. van den Bergh and Engelbrecht [221] used a de-compositional strategy that decomposed the n dimensional problem into m s-dimensional sub-problems, which showed better performance than the original CC method. Much work has been done in the use of cooperative co-evolution in large scale function optimization and the concentration has been on non-separable problems. Several methods have been proposed which group interacting variables for global optimization problems. Yang et. al. [233] have presented the cooperative coevolution method that uses a *random grouping* and *adaptive weighting* strategy with differential evolution (DECC-G) for its subcomponents. The method groups interacting variables into the same separate subcomponents heuristically. Yang et al. also presented a multi-level cooperative coevolution method (MLCC) [234] which adapts the size of the subcomponents in DECC-G in order to group interacting and non-interacting variables. The method starts with small sized subcomponents and adapts to bigger subcomponents sizes from a predefined set. MLCC showed better performance than DECC-G for non-separable problems of

up to 1000 dimensions. Omidvar et. al. [156] made amendments to the random grouping approach in DECC-G. They presented a *more frequent random grouping* approach which outperformed its counterpart in several non-separable problems of up to 1000 dimensions.

The number of subcomponents used in the respective encoding schemes plays an important role during evolution. Each encoding scheme groups interacting variables and makes an assumption on the degree of non-separability regardless of the problem. CoSyNE for instance, views the neural network training as a separable problem and has been only effective for training RNNs on pole balancing problems. CoSyNE performed poorly in training feedforward networks for pattern classification problems in Chapter 3. ESP and NSP decomposes the problem with different degrees of non-separability. NSP showed better performance than the other problem decomposition methods (ESP and CoSyNE and CME) for training feedforward networks in pattern classification and recurrent network in grammatical inference problems given in Chapter 3. The results in Chapter 3 have demonstrated that different problem decomposition methods are of advantage to different network architectures and application problems. Problem decomposition methods such as CoSyNE, ESP and NSP exhibit different degrees of non-separability by grouping interacting variables. In literature, it has not been investigated if the nature of the neural network training problem changes during evolution, i.e if different problem decomposition methods are needed at different stages of the evolutionary process.

In the use of cooperative coevolution for neuro-evolution [168, 64, 76, 187] , little attention has been given on the issue of separability and interacting variables in the literature. In the literature, it has been discussed that multi-level cooperative coevolution (MLCC) [234] adapts the size of the subpopulation at different levels of evolution and improved performance for global optimization problems. A major problem of the MLCC approach is that it merges the sub-populations based on a predefined set

which does not cater for the interacting variables between the sub-populations. In order to make adaptation of problem decomposition in neural networks, the interacting variables must be considered and the sub-populations should be merged using some of the established problem decomposition methods.

The analysis in Section 3.1.1 of Chapter 3 has shown that the interaction between the weights increases during the later stages of evolution as the problem is being learnt. Therefore, it is reasonable to adapt the problem decomposition method (modularity) as different levels of search (in terms of non-separability) are needed at different stages of evolution. In the rest of the discussion, the subcomponents in cooperative neuro-evolution are referred to as *modules* and the level of modularity corresponds to the problem decomposition method. This chapter presents an adaptive modularity cooperative neuro-evolution method (AMCC) for training feedforward networks for pattern classification and recurrent neural networks for grammatical inference problems. Instead of using a fixed level of modularity during the entire evolutionary process (as in NSP, CoSyNE and ESP), the AMCC method adapts the number of modules at different stages of the evolution. The performance of the adaptive modularity method is compared with CoSyNE and NSP.

5.2 Adaptive Modularity in Cooperative Neuro-evolution

The general idea behind the adaptive modularity cooperative neuro-evolution (AMCC) method is to use the strength of a particular problem decomposition method that reflects on the degree of non-separability when needed during evolution. The adaptive method can be visualised as a car driving on a hill where the driver changes the gear according to the steepness of the hill. Similarly, the adaptive method changes its level of modularity

during evolution. It uses a modularity with greater level of flexibility (allowing evolution for separable search space) during the initial stage and decreases the level of modularity during the later stages of evolution.

The modularity of the general AMCC method for training recurrent neural networks can transform from synapse level encoding (CoSyNE) to neuron level encoding (NSP) and finally to network level encoding (standard neuro-evolution where one population is used). The adaptation of modularity using all the three levels of encoding forms the general AMCC method. The levels of encoding are further described as follows:

1. **Synapse level encoding**: Decomposes the network into its lowest level to form a single module [77]. The number of connections in the network determines the number of modules.

2. **Neuron level encoding**: Decomposes the network into neuron level. The number of neurons in the hidden, state and output layer determines the number of modules [27, 28].

3. **Network level encoding**: The standard neuro-evolutionary encoding scheme where only one individual represents the entire network. There is no decomposition present in this level of encoding.

The AMCC method is given in Algorithm 7 which follows three main stages of evolution. At the beginning, all the sub-populations of the synapse level, neuron level and network level encoding are initialised with random real values in a range. In Stage 1, the sub-populations at synapse level encoding are cooperatively evaluated only. Neuron level and Network level encoding are left to be cooperatively evaluated at later stages. In the rest of the discussion, we assume that the algorithm terminates with a specified minimum error (MinError).

Stage 1 uses synapse level encoding where the sub-populations are evolved until the maximum number of function evaluations (MaxGlobal) is reached. The optimal solution is returned if the minimum error is reached

(MinError). However, if MaxGlobal and MinError are not satisfied, the change of modularity is done and the method proceeds to Neuron level encoding in Stage 2. The best individuals with their fitness are transferred to the sub-populations of the Neuron level at Stage 2. All the sub-populations are cooperatively evaluated. During Neuron level evolution, if the MinError is not reached within the MaxGlobal, then the change of modularity is done in order to proceed to the Network level encoding at Stage 3. The best individuals at Neuron level from each sub-population are concatenated and added to the population at the Network level. Note that the Neuron level sub-populations and the Network level population initially contain randomly initialised genetic material. The best individual from the previous stage is added to the sub-populations at the next stage and then the evolution proceeds. The condition that enforces the modularity change is given by *ChangeMod* in Algorithm 7. ChangeMod will be further discussed in detail later in Section 5.2.3.

Alg. 7 Adaptive Modularity in Cooperative Coevolution

Initialise Synapse, Neuron and Network level

Stage 1: Synapse level encoding
Cooperatively evaluate Synapse level only

while *(FuncEval* $\leq$ *MaxGlobal) OR ChangeMod* **do**
    foreach *each Subpopulation at Synapse level* **do**
        Genetic Operators
        Cooperative Evaluation
    end
end
Stage 2: Neuron level encoding
i. Carry best individuals from Synapse level into Neuron level
ii. Cooperatively evaluate Neuron level

while *(FuncEval* $\leq$ *MaxGlobal) OR ChangeMod* **do**
    foreach *each Subpopulation at Neuron level* **do**
        Genetic Operators
        Cooperative Evaluation
    end
end
Stage 3: Network level encoding
i. Carry best individuals into Network level
ii. Evaluate Network level

while *FuncEval* $\leq$ *MaxGlobal* **do**
    Network level Genetic Operators

    if *(FuncEval* $\leq$ *MinGlobal)* **then**
        Go back to Neuron level encoding
    end
end

Note that in Stage 3, if the minimum number of function evaluations (MinGlobal) has not been reached and the problem has not been solved,

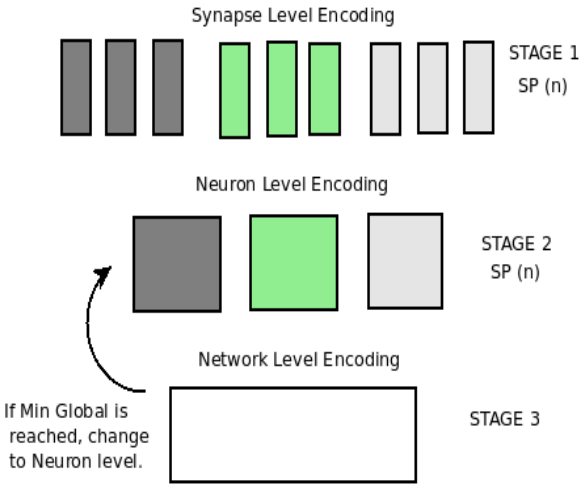


Figure 5.1: The 3 Stage AMCC method. The figure shows how the method transforms the evolutionary process with different levels of encoding. The sub-populations *SP (n)* at Synapse level and Neuron level are shown. Note that in Stage 3, if the problem is not solved before reaching the global minimum time, then the modularity is changed from Network to Neuron level.

then the method assumes that there is a convergence to a local minimum. Therefore, the modularity is changed to the Neuron level in order to employ a different level of evolution. In this case, the best individual from the Network level is also copied to the Neuron level along with its fitness. This is also shown in the general AMCC method in Figure 5.1 which gives a visualization of how the method transforms the evolutionary process with different levels of encoding.

5.2.1 Function Evaluation During Initialisation

As discussed, there are two main phases of evolution in the cooperative coevolution method. These are the *initialisation phase* and the *evolution phase*.

The computational cost in terms of the number of function evaluations for each level of encoding during initialisation in the respective neural networks architecture can be evaluated. In feedforward networks, the cost

of function evaluations during initialisation is calculated as follows.

1. *Synapse level encoding*
   $Synapse = ((I * H) + (H * O) + H + O) * P$ (This includes the biases)

   Where I, H and O are the number of Input, Hidden and Output neurons, respectively. P is the Population size.

2. *Neuron level encoding*
   $Neuron = (H + O) * P$

   Where H and O are the number of Hidden and Output neurons, respectively. P is the Population size.

3. *Network level encoding*
   $Network = P$

   Where P is the Population size.

In recurrent networks, the cost is as follows.

1. *Synapse level encoding*
   $Synapse = ((I * H) + (H * O) + (H * H) + H + O) * P$ (This includes the biases)

   Where I, H and O are the number of Input, Hidden and Output neurons, respectively. P is the Population size.

2. *Neuron level encoding*
   $Neuron = (2H + O) * P$

   Where H and O are the number of Hidden and Output neurons, respectively. P is the Population size.

3. *Network level encoding*
   $Network = P$

   Where P is the Population size.

In general, the AMCC method will include the cost of functional evaluations during initialisation in all three stages.

$$AMCC = Synapse + Neuron + Network$$

5.2.2 Transfer of Valuable Information

The transition from one level of modularity to another should ensure that the information gained using the existing level of modularity is transferred to the next level of encoding. In the AMCC method, the best individuals in each population at each level are transferred to the next level. The transfer of information requires the best individuals from the respective sub-populations in the Synapse and the Neuron level to be concatenated into a single individual.

Note that the number of sub-populations used for each level is different. The Synapse level encoding has more sub-populations than the neuron level, however, information from the best individuals synapse level must be added to Neuron level. The transfer of information from each level is further described as follows.

1. *Synapse to Neuron level*

   The best individuals from all the sub-populations from the synapse level are concatenated. The concatenated string is broken down into neuron level and added to the respective sub-population along with its fitness.

2. *Neuron to Network level*

   Note that the neuron level encoding contains several sub-populations while network level encoding contains one population only. The

best individuals from all the sub-populations are concatenated and copied into the single population of the Network level.

3. *Network to Neuron level*

   In this case, the best individual from the network level will be decomposed and transfered to the neuron level.

5.2.3 The Heuristic to Change Modularity

The general idea in changing modularity is to check whether the neural network is making a good improvement in terms of the overall network *Error* over time. If no or little improvement is made, then a change in modularity will be done.

Alpha and *Beta* are two small constants which act as a threshold and determine how much difference is acceptable, and if these thresholds are passed, then change in modularity is enforced. The constants are small as they check the change in the overall error of the network during evolution. *Alpha* and *Beta* can be chosen empirically with trial experiments. *Alpha* will be usually bigger than *Beta* as there are greater differences in change of error during the earlier stages of evolution. During the later/final stages of evolution, the method will transform from Neuron to Network level of modularity. In this stage, the change in error is lower than the initial stages.

ChangeMod is dependent on the change in error (ChangeError) which can be calculated as follows.

$$ChangeError = Error[current - n] - Error[current]$$

where $Error$ is the *mean-squared-error* or *sum-squared-error* of the neural network. $current$ is the current cycle or generation and n is the number of time steps the change of error is monitored.

The heuristic of changing the modularity from the Synapse level to Neuron level is given as follows.

$ChangeError = Error[current - n] - Error[current]$

If($ChangeError < Alpha$)
Set $ChangeMod$ to TRUE

This heuristic is used in the AMCC method as shown in Algorithm 7 . Once ChangeMod is TRUE, the modularity is changed from Synapse level to Neuron level. A similar approach is used for Neuron level to Network level where $Beta$ is used as the threshold.

The optimal value for the change of modularity can be determined by experimental runs with different values for Error change in the Synapse-Neuron level and the Neuron-Network level. This is shown in the experimental section and is compared with the heuristic method for the change in modularity. Note that this method is not only confined to three stages (Synapse-Neuron-Network) of modularity adaptation only. More stages can be incorporated according to the nature of the problem; however, the performance may deteriorate if more levels are used than required. In Algorithm 7 , in Stage 3, the system will return to the Neuron level if convergence is not achieved within a specified time which is close to Max-time. This implies that if local convergence occurs at the Network level, the AMCC method will revert to Neuron level. Note that the terminology "ChangeMod" is similar to the "Halting Windows" approach used for multi-modal problems [11].

5.3 Simulation and Analysis

This section presents an experimental study of the proposed AMCC method and compares it with the Neuron Level (NL) and the Synapse Level (SL) encodings for training feedforward and recurrent neural networks on pat-

tern classification and grammatical inference problems, respectively.

Note that AMCC for feedforward networks is referred to as AMCC-FNN and AMCC-RNN is used for recurrent neural networks for the rest of the chapter.

The same evolutionary algorithm [42] is used in SL, NL and the AMCC method for a fair comparison. The population size of 100, the pool size of 2 offspring and the family size of 2 parents is used. This set-up has also been used in Chapters 3 and 4. The individuals in the respective populations are seeded with random real numbers in the range of [-5, 5] in all the experiments.

5.3.1 Feedforward Neural Networks

Benchmark Problems and Neural Network Configuration

Table A.1 of the Appendix shows the neural network configuration and dataset details used for all the experiments. The 4-Bit, Iris, Wine and Zoo classification problems are selected. The maximum training time given by the number of function evaluations in all the problems is fixed as 15000 for Iris and Wine classification and 30000 for Zoo classification. In addition to the three real world problems, the 4-Bit parity problem is used and the network is trained until the sum-squared error goes below 0.001 or when the maximum number of function evaluations has reached 20000. Note that 70% of the data is used for training and the remaining 30% is used for testing the generalisation performance.

AMCC for Feedforward Networks

This section evaluates the performance of the AMC-FNN method and compares its performance with Neuron and Synapse level encoding. The 2-Stage AMCC-FNN method is also used here where only the Neuron and Network levels are used; i.e. Neuron level is used initially and later modularity is adapted to the Network level (Synapse level is not used here).

The 3-Stage AMCC-FNN method employs Synapse, Neuron and Network level encodings as shown in Figure 5.1 and Algorithm 7 . In both cases (2-Stage and 3-Stage AMCC), the method reverts to Neuron level encoding from Network level if desired convergence is not achieved within 70 percent of the maximum time. This is done in order to ensure that local convergence has not been reached using Network level encoding.

In the change of modularity for the 3-Stage method, the *Alpha* of 1E-2 is used to change from Synapse to Neuron level. The *Beta* of 1E-5 is used to change from Neuron to Network level. In the 2-Stage method, 1E-5 is used to change from Neuron to Network level. These values were determined during trial experiments.

The goal of the experimental study is to observe the performance of the respective algorithms in relation to a particular topology; i.e. fixed number of hidden neurons. This reflects on scalability and robustness.

The results are given in Figures 5.2 to 5.5. The confidence interval is shown as error bars in the histograms showing the optimization time. In the wine classification problem shown in Figure 5.2, NL and SL do not scale well for all the different number of hidden neurons in (a) and (b). The 2-Stage method shows the best performance given by the least optimization time in (a) and the success rate in (b). It shows better performance when compared to the 3-Stage method. SL encoding failed to converge in all the cases.

In the results of the 4-Bit parity problem shown in Figure 5.3, the 3-Stage method gives the best performance in terms of scalability shown by the optimization time in (a) and the success rate in (b). NL shows better performance than the 2-Stage method in terms of the optimization time in (a). The SL encoding scheme fails to converge for 3 hidden neurons and shows the worst performance.

In the results of the Iris classification problem shown in Figure 5.4, the 2-Stage method gives the best results in terms of the optimization time for 4 and 5 hidden neurons as shown in (a). In (b), the 2-Stage method shows

similar performance to the 3 Stage method which is better than NL. The NL encoding shows best results for 6, 7 and 8 neurons which is also shown in (a). The SL encoding scheme fails to converge within the maximum number of function evaluations in all the cases.

In the results of the Zoo classification problem shown in Figure 5.5, the 2-Stage and SL method shows poor performance. The NL and 3-Stage method shows better performance, however they do not scale well for different number of hidden neurons. The best performance is achieved by the 3-Stage method for 4 and 6 hidden neurons, while NL gives the best performance for 5, 7 and 8 hidden neurons.

In general, SL gives poor performance in all four problems. EA gives poor performance in all problems. In most cases, it had difficulty to converge within the maximum number of function evaluations. In Wine classification, the 2 Stage framework showed better performance, while in the 4-Bit parity problem, the 3-Stage framework showed the best performance. In the Iris classification, the 2 Stage framework showed better performance for lower number of hidden neurons where NL encoding scheme showed better for greater number of hidden neurons. In Zoo classification, better performance is achieved by the 3-Stage framework and NL.

Although the main goal of this section is not the generalisation performance, Table 5.1 gives a comparison of NL and 3-Stage AMCC framework for Iris, Wine and Zoo problems. The Iris classification performance shows minor improvement for 5 or more hidden neurons while the Wine and Zoo classification problem achieves similar generalisation using both methods. The subscript for the corresponding values in Table 5.1 gives the 95 % confidence interval. Note that 4-bit parity will not be tested for generalisation since this is a different type of problem.

Discussion: AMCC in Feedforward Networks

The results in general show that AMCC (2 or 3 Stage) method achieves the best performance in most cases.

Table 5.1: Generalisation Performance of 3-Stage AMCC and NL

Problem	Hidden	3-Stage AMCC	NL
Iris	4	$95.35_{0.46}$	$95.58_{0.47}$
	5	$96.46_{0.31}$	$94.74_{0.73}$
	6	$96.94_{0.25}$	$95.93_{0.47}$
	7	$97.03_{0.26}$	$95.37_{0.61}$
	8	$97.31_{0.14}$	$95.24_{0.54}$
Wine	4	$93.36_{0.76}$	$93.93_{1.44}$
	5	$91.56_{0.95}$	$92.50_{0.92}$
	6	$90.44_{0.88}$	$90.90_{1.13}$
	7	$94.66_{0.87}$	$93.39_{0.81}$
	8	$92.34_{1.23}$	$91.75_{1.96}$
Zoo	4	$73.91_{2.59}$	$73.23_{1.38}$
	5	$73.79_{1.84}$	$73.92_{1.34}$
	6	$73.00_{1.96}$	$72.51_{1.17}$
	7	$72.29_{1.58}$	$73.07_{1.20}$
	8	$72.58_{2.73}$	$73.75_{1.39}$

The difference in the performance of the four methods (NL, SL, 2-Stage, 3-Stage) indicates that the nature of the neural network training changes during evolution, which is why adaptation is necessary. The nature of the change during training is dependent on the problem type, learning difficulty and adaptation in terms of the degree of non-separability.

The nature of the problem also changes when different number of hidden neurons is used to represent the problem. The problem becomes too difficult to be solved when there are not sufficient neurons present in the hidden layer or when there are more than enough neurons present and the problem size increases. This can be seen in the performance of the Iris classification in Figure 5.4 where the 2-Stage method shows best performance for 4 and 5 hidden neurons and deteriorates in performance for 6 -8 neurons. The success of the 2 Stage evolution in this case indicates that when a lower number of neurons are present, a higher degree of non-separability is needed and hence the change in modularity is needed. However, for 6 to 8 neurons, the success of NL where no modularity is adapted indicates that the degree of non-separability is not increased during later stages in evolution. Here, it seems that the degree of non-separability can be related to the number of hidden neurons.

The 4-bit parity problem is of a different nature when compared to pattern classification problems. The 3-Stage method shows the best performance as the size of the network is increased.

5.3.2 Recurrent Neural Networks

This section evaluates the performance of the AMCC method for training recurrent neural networks using grammatical inference problems. The datasets for the FFA and Tomita problems (T1-T4) were generated as done in Section 3.3.2.

In all problems, the RNN is trained until the mean-squared-error (MSE) has reached below 0.0005. This is done in order to make the problem more

challenging than the RNN problems in Chapters 3 and 4. The maximum number of function evaluations for T1 and T2 is 5000. T3, T4 and FFA problem use a maximum of 10000 function evaluations. The goal of this section is to observe the performance of AMCC-RNN during training. The optimization time and success rate are the two main performance measures for training.

The Change of Modularity

The simulation begins with an observation of the training performance according to the mean-squared-error (Error) of the network. The change of modularity is done when the current level of encoding reaches the specified Error from *Synapse-Neuron* level and *Neuron-Network* level. The average number of function evaluations is used as the main measure of optimisation time in the AMCC-RNN method. The run is considered successful if it fulfils the minimum error specified (0.0005) before reaching the maximum number of function evaluations for the problem.

The change in modularity helps in understanding the nature of the neural network optimisation problem and how the level of modularity and its encoding scheme contributes to evolution in terms of the degree of non-separability.

The results are shown in Figure 5.6 and Figure 5.7. The optimal values takes in account the optimization time taken and the success rate from 100 independent experimental runs. The T1 and T2 problems employ 2 hidden neurons, T3 and T4 problems employ 3 hidden neurons and the FFA problem employs 4 hidden neurons in these experiments.

In Figure 5.6, for the FFA problem, the Error of 0.1 (in Synapse–Neuron level axis) and 0.05 (in Neuron–Network level axis) shows the best success rate in (a). The corresponding least optimization time taken is shown in (b). The performance deteriorates when the Error in the Synapse–Neuron level becomes lower than 0.05.

In Figure 5.7, for the T3 problem, the Error of 0.2 (in Synapse–Neuron

level axis) and 0.05 (in Neuron–Network level axis) shows the best success rate in (a). The corresponding least optimization time taken is shown in (b).

The performance of the heuristic method is compared with the iterative search method. The heuristic method allows the AMCC-RNN method to change modularity without undergoing an iterative search for the parameters. This can save computation time that is used for parameter setting. Table 5.2 shows the comparison of the Heuristic (Heuris.) and Iterative (Itera.) method for the change in modularity in the AMCC-RNN method. An iterative search is done to find the best values for the parameters in Synapse to Neuron level (Sy–Neu) and Neuron to Network level (Neu-Net). The results of the iterative method for the change of modularity for the T3 and FFA problems are shown in Figures 5.6 and 5.7.

The heuristic method observes the change in the network error over a predefined number of cycles. If the change in error is less than the predefined *Alpha* (for Synapse–Neuron level) and *Beta* (Neuron–Network level), then the modularity changes to the next level. The *Alpha* of 0.05 and *Beta* of 0.01 is used in all the experiments. These values are determined in trial experiments.

Problem	Hidden	Method	Sy–Neu	Neu-Net	FuncEval	Success
T2	2	Itera.	0.2	0.05	$2487_{\pm 232}$	88
	2	Heuris.	–	–	$2606_{\pm 187}$	91
T3	3	Itera.	0.2	0.05	$6535_{\pm 451}$	80
	3	Heuris.	–	–	$6528_{\pm 384}$	84
T4	3	Itera.	0.15	0.05	$4115_{\pm 315}$	97
	3	Heuris.	–	–	$4396_{\pm 228}$	98
FFA	4	Itera.	0.1	0.05	$8390_{\pm 494}$	59
	4	Heuris.	–	–	$8476_{\pm 409}$	56

Table 5.2: A comparison of the Heuristic and Iterative method in-order to determine when to change modularity

The results in Table 5.2 show that there is not a major difference be-

tween the heuristic method and the best values taken from the iterative method for the four problems. The 95 % confidence interval is given in the subscript of the values for the mean function evaluations. In Table 5.2, it has been reported that the two methods get similar results for all the problems. The iterative method requires a search for the best parameter setting, and hence more computation times is needed. The heuristic method eliminates parameters, and saves the computational time. Therefore, it is important to use the heuristic method in the AMCC-RNN method. In the following subsections, the AMCC-RNN will employ the heuristic method for comparison with other methods.

AMCC for Recurrent Networks

This section evaluates the comparative performance of the AMCC-RNN method with Neuron and Synapse level encoding. The AMCC-RNN uses the heuristic in change of modularity as discussed in previous subsection.

The goal is to observe the performance of the respective methods (NL, SL, AMCC) in relation to a particular network topology; i.e. fixed number of hidden neurons. Note that the number of hidden neurons directly influences the difficulty of the learning problem. It is more difficult to learn the problem if insufficient neurons are present in the hidden layer. It is also difficult for evolutionary algorithms to optimise a problem when the number of variables increases according to the number of hidden neurons in the hidden layer.

Figures 5.8 - 5.12 shows the comparison of the performance of each method in relation to the number of hidden neurons. The confidence interval is shown as error bars in the histograms showing the optimization time. The number of successful runs out of 100 experiments is shown in part (a) while the optimization time in terms of the number of average function evaluations is shown in part (b) of the respective figures. The goal of AMCC-RNN is to obtain maximum success with the least optimization time. Note that the average function evaluations consists of both

successful and unsuccessful runs.

In Figures 5.8 and 5.9, for the T1 and T2 problems, the performance of AMCC-RNN is consistent as the number of hidden neurons change in terms of the optimization time (a) and best success rates (b). The performance of SL and NL is good only for certain number of hidden neurons and deteriorates otherwise.

In Figure 5.10, for the T3 problem, AMCC outperforms SL and NL for all the different number of hidden neurons in terms of the optimization time (a) and best success rates (b). In the T4 problem given in Figure 5.11, the performance of NL is slightly better than AMCC for 3, 4 and 5 neurons. AMCC outperforms NL for 5 and 6 neurons. The performance of SL is comparable to AMCC and NL for 4 neurons only. The SL performance deteriorates otherwise.

In the FFA problem given in Figure 5.12, AMCC outperforms NL in all the cases. SL performs better than AMCC for 6 hidden neurons only. Its performance deteriorates otherwise.

In summary, in the majority of the cases, AMCC has outperformed the other methods. NL and SL seem to perform well only for certain number of hidden neurons. The results have shown that the AMCC method scales better than NL and SL when the size of the problem increases. The number of variables increases significantly as the number of hidden neurons increases.

Discussion: AMCC for Recurrent Networks

Neural network training is similar to any other optimisation problem which requires a global search in the initial stage and a local search in the final stage for further refining the solution. The are two concerns, 1) global and local search and 2) degree of non-separability. In most problems, a global search is needed in the beginning and local search is employed during the final stages. Different evolutionary algorithms balance the intensity of the global-local search using different forms of crossover and mutation opera-

tors. In memetic cooperative coevolution, additional local search is used to balance the global exploration with local exploitation [200, 164]. This has been well investigated in Chapter 4 where it was found that local refinement is important for cooperative neuro-evolution. The second issue is to achieve balance with the adaptation of the training algorithm according to the degree of non-separability.

The AMCC-RNN and AMCC-FNN method utilize the strengths of both synapse, neuron, and network level encoding. This gives it the flexibility to apply a particular problem decomposition method at a given situation in the evolutionary process. The neural network training is a non-separable problem as it has interacting variables. At different stages of evolution, the iteration between variable changes and therefore, the degree of non-separability changes. AMCC performs well as it has the flexibility to adapt the search which relates to the different degrees of non-separability. In the initial stages, the neural network training problem has a lower degree of non-separability. As the problem is being learnt, the interaction between interacting variables increases which requires a change of modularity and hence the neuron and network level encodings perform well when needed.

The AMCC-RNN method uses the synapse level encoding which has its strength in separable problem (lower degree of non-separability) where there are lower number of interacting variables. The synapse level encoding provides more flexibility, greater global search capability as each synapse in the network uses a separate sub-population. As the evolution proceeds, the error of the neural network decreases, and the interaction amongst the variables (synapses) increases, this has been shown in Section 3.1.1 of Chapter 3. Therefore, it is reasonable to use larger subcomponents to group interacting variables and hence the neuron level encoding is employed. During the final stages of the evolution, the interaction among variables becomes stronger. The problem which has been decomposed into smaller sized subcomponents at the neuron level has to be merged

and solved as a single large problem. At this stage, the network level encoding is used where a single population is used to represent the entire network.

The results have also indicated that the nature of the problem changes while training recurrent neural network, i.e the degree of non-separability increases which further validates the analysis shown in Section 3.1.1 of Chapter 3. The problem has shown to be separable initially (where Synapse level encoding helps) and later, the degree non-separability increases where network level encoding with one population is more effective. The improved performance of the AMCC method in comparison with the Neuron and the Synapse level gives justification that the degree of non-separability changes during evolution. The Neuron and Synapse level encodings failed to deliver good or acceptable solutions in all cases as they did not have the feature of adapting their search according to the requirements of the problem. It has been shown that the Neuron and Synapse level encodings do not scale as well as AMCC-RNN when the size of the problem increases due to the number of hidden neurons.

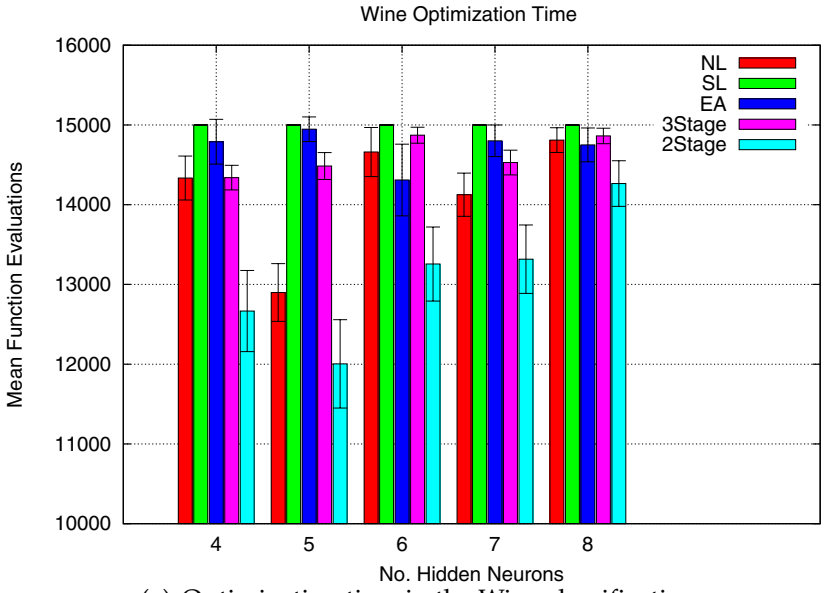
All the given problems show that, unlike the synapse and neuron level encodings, AMCC-RNN maintains good performance when a greater number of hidden neurons are present in the hidden layer. Therefore, it has the ability to preserve the recurrent state information when the number of modules and the size of the problem significantly increases.

5.4 Chapter Summary

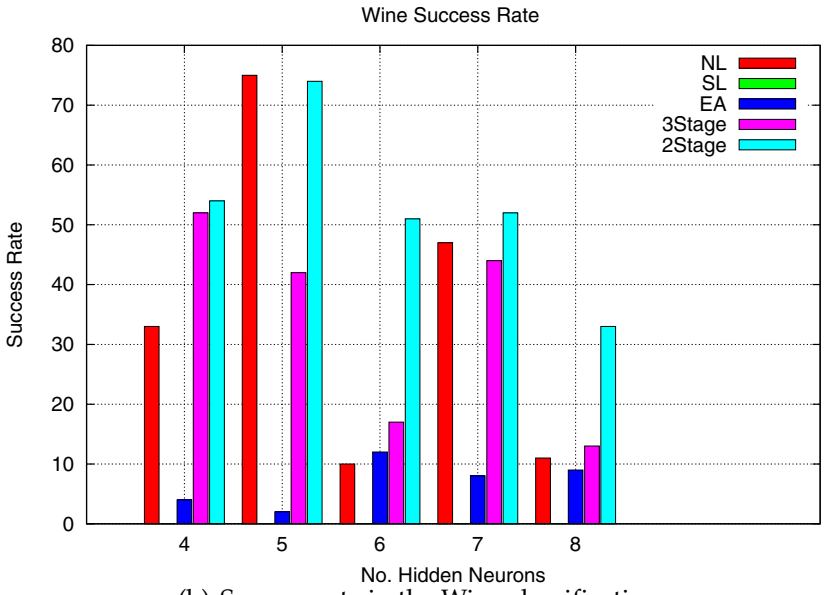
This chapter introduced a novel cooperative co-evolution method for adapting modularity during evolution for training feed-forward and recurrent neural networks. The method provided a better understanding of the degree of non-separability and its relationship to problem decomposition in cooperative neuro-evolution.

The results have shown that due to the change in the nature of prob-

lem (in terms of non-separability) modularity adaptation is needed in the training of feedforward and recurrent networks. The level of non-separability changes at different stages of evolution and hence, the change of modularity is needed. Adaptation through changing modularity has shown to play an important role in cooperative neuro-evolution.

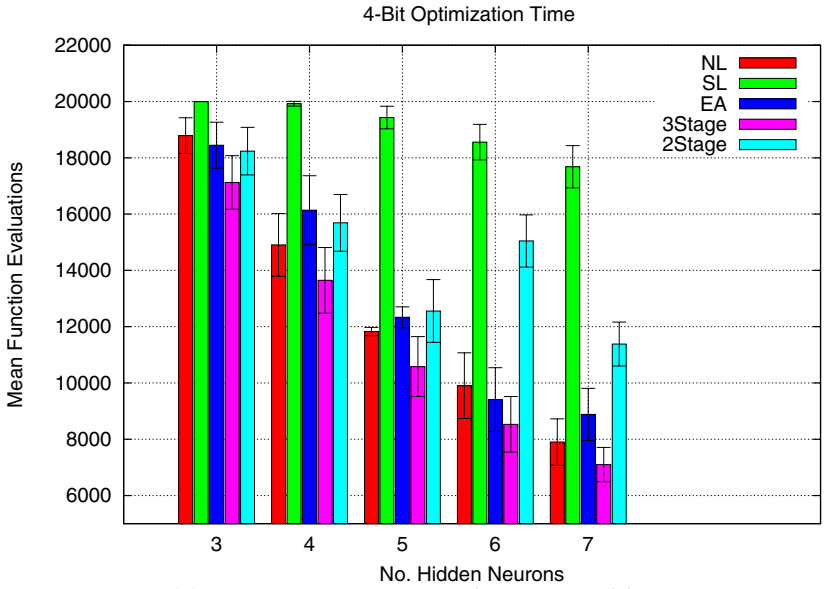


(a) Optimization time in the Wine classification

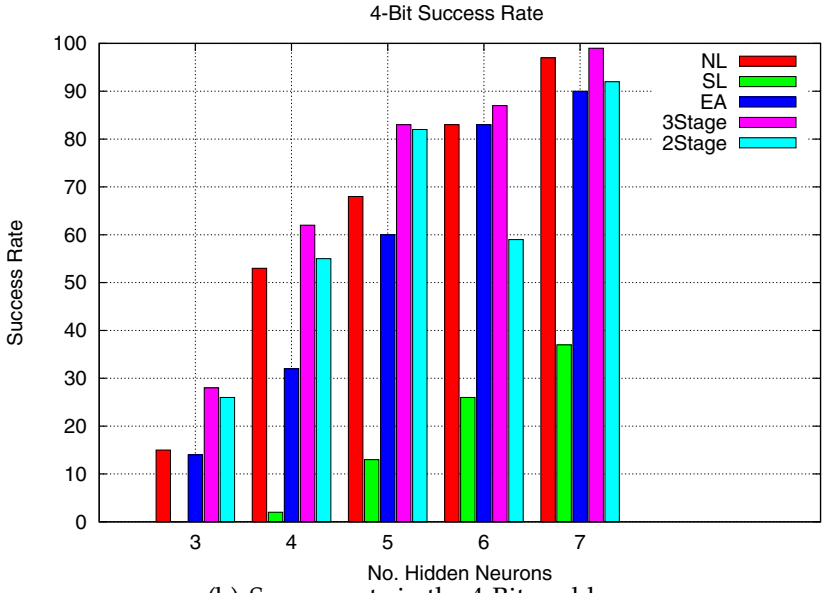


(b) Success rate in the Wine classification

Figure 5.2: The performance the 2 Stage and 3 Stage AMCC-FNN method on different number of hidden neurons for the Wine classification problem. The performance of NL and SL is also given.

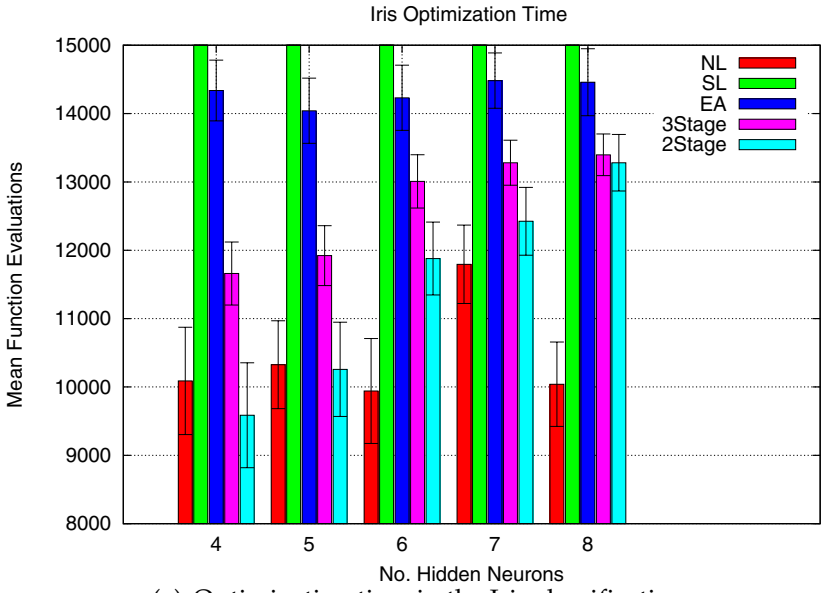


(a) Optimization time in the 4-Bit problem

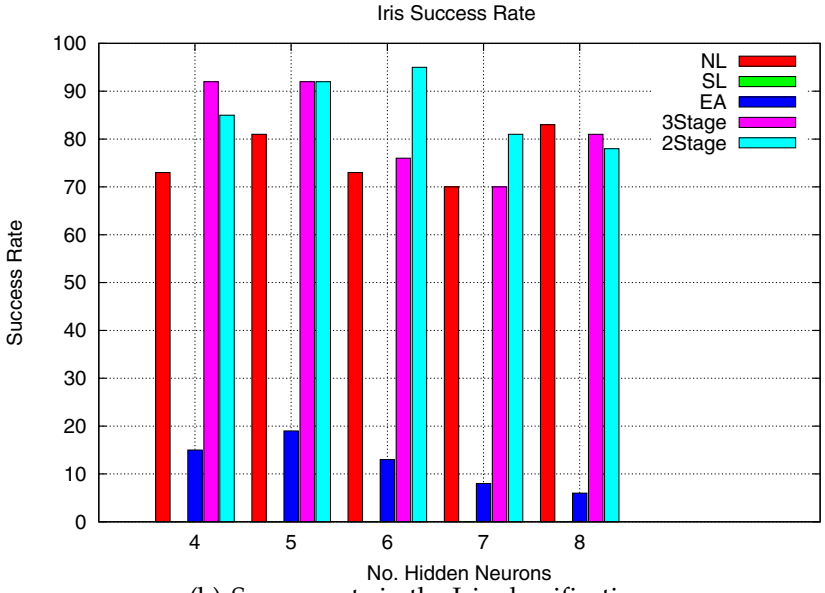


(b) Success rate in the 4-Bit problem

Figure 5.3: The performance of 2 Stage and 3 Stage AMCC-FNN on different number of hidden neurons for the 4-Bit problem. The performance of NL and SL is also given.

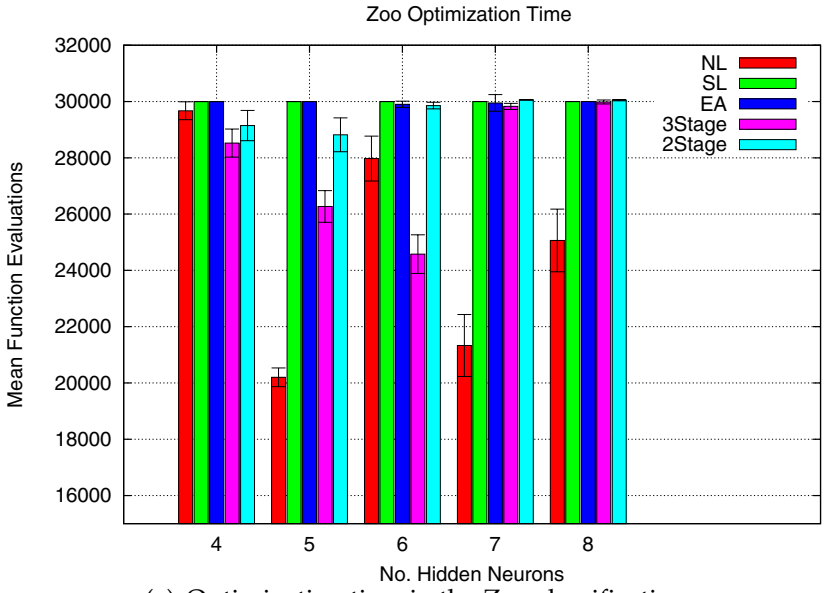


(a) Optimization time in the Iris classification

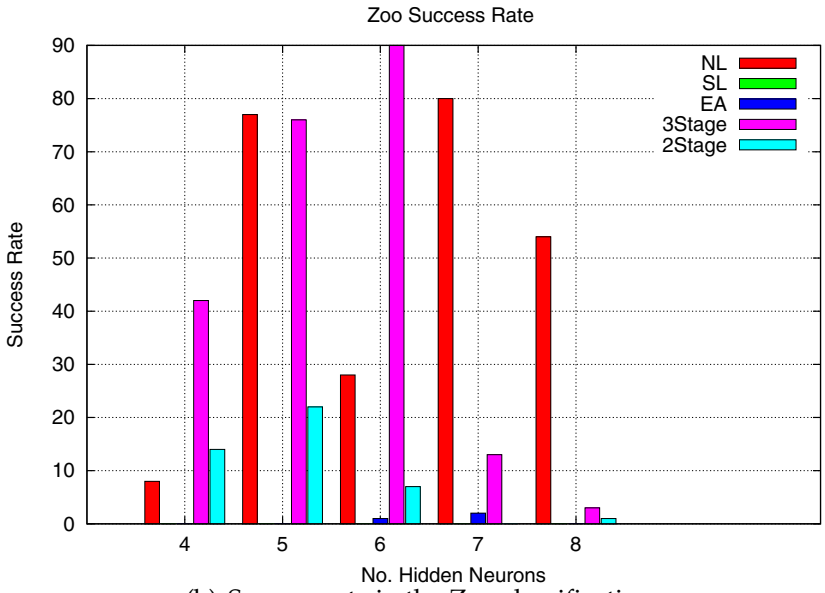


(b) Success rate in the Iris classification

Figure 5.4: The performance of the 2 Stage and 3 Stage AMCC-FNN method on different number of hidden neurons for the Iris classification problem. The performance of NL and SL is also given.

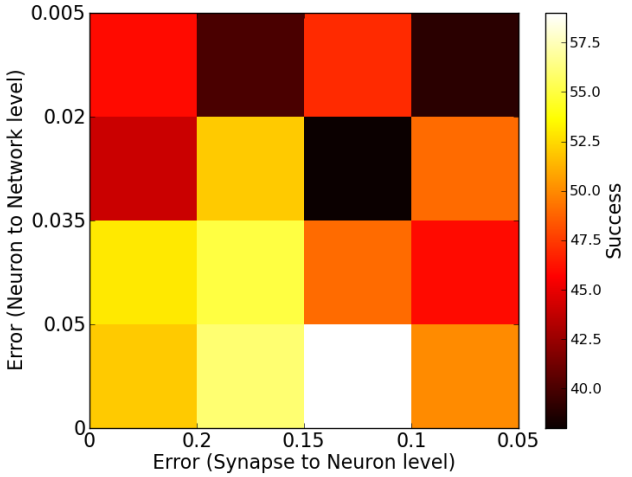


(a) Optimization time in the Zoo classification

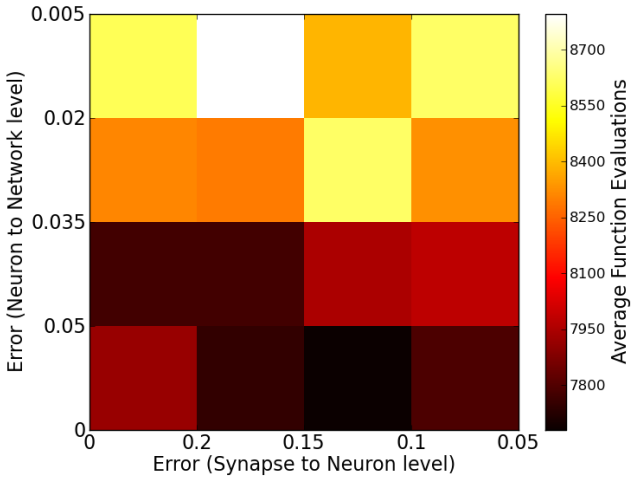


(b) Success rate in the Zoo classification

Figure 5.5: The performance the 2 Stage and 3 Stage AMCC-FNN method for the Zoo classification problem. The performance of NL and SL is also given.

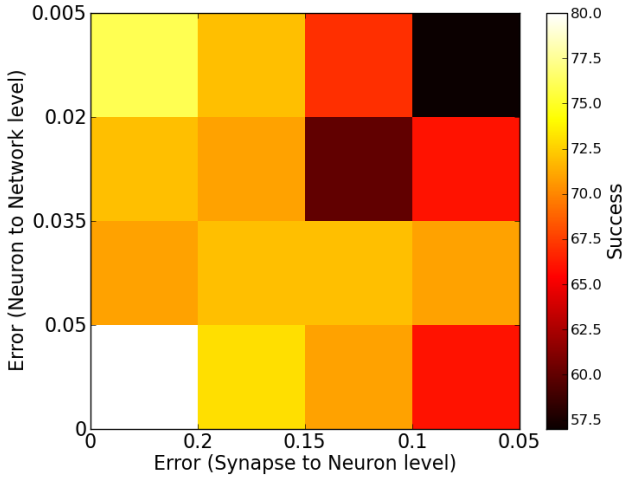


(a) Success rate in the FFA Problem

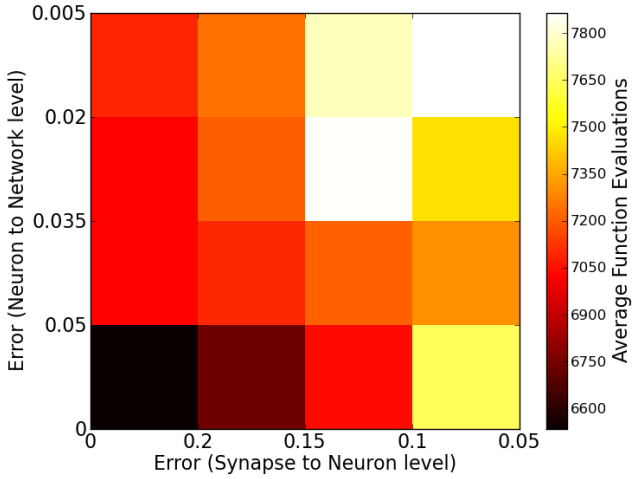


(b) Optimisation time in the FFA Problem

Figure 5.6: The heat-map shows the performance of the AMCC-RNN method on different values for Error (mean-squared-error) for Synapse–Neuron level and Neuron–Network level changes in modularity for the FFA problem. The number of successful runs out of 100 experiments is shown in (a) while the optimisation time is shown in (b). The goal of AMCC-RNN is to obtain maximum success with the least optimization time. The Error of 0.1 in Synapse–Neuron level and 0.05 in Neuron–Network level shows the best success rate in (a) with corresponding least number of function evaluations in (b). Note that the optimisation time consists of both successful and unsuccessful runs.

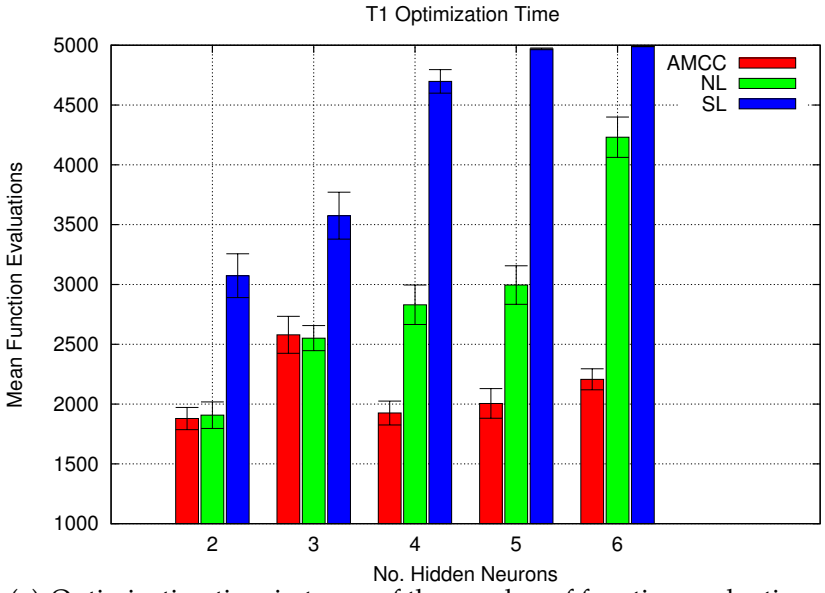


(a) Success rate in the T3 Problem

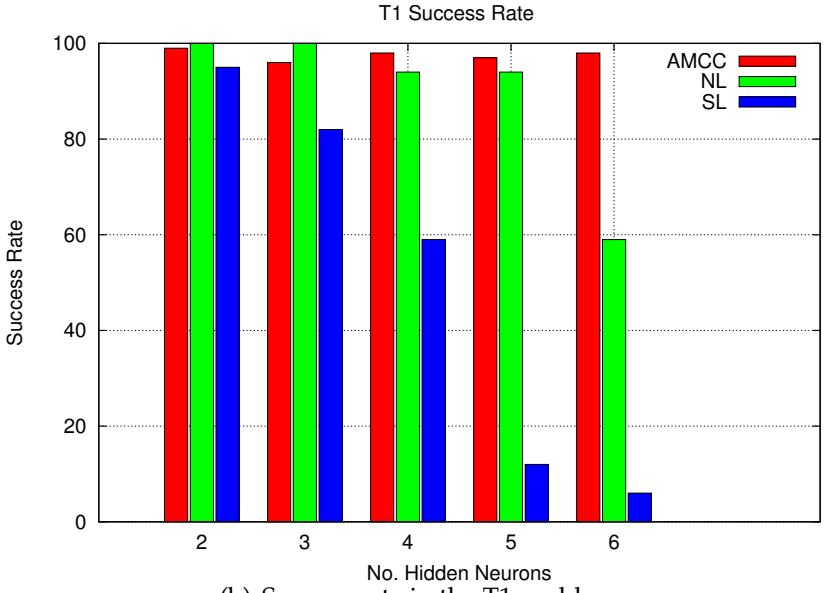


(b) Optimization time in the T3 Problem

Figure 5.7: The heat-map shows the performance of the AMCC-RNN method on the T3 problem. The number of successful runs out of 100 experiments is shown in (a) while the optimization time is shown in (b). The Error of 0.2 in Synapse–Neuron level and 0.05 in Neuron–Network level shows the best success rate in (a) with corresponding least optimization time in (b).

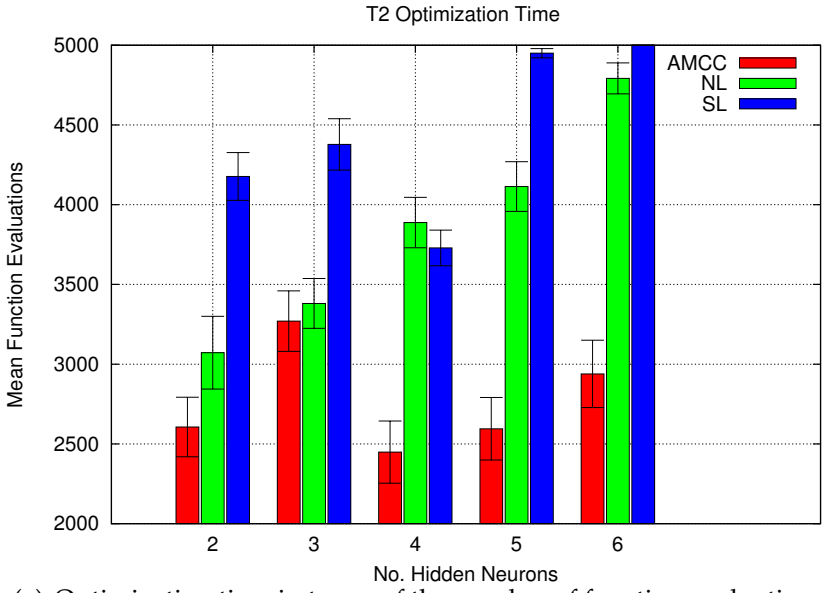


(a) Optimization time in terms of the number of function evaluations

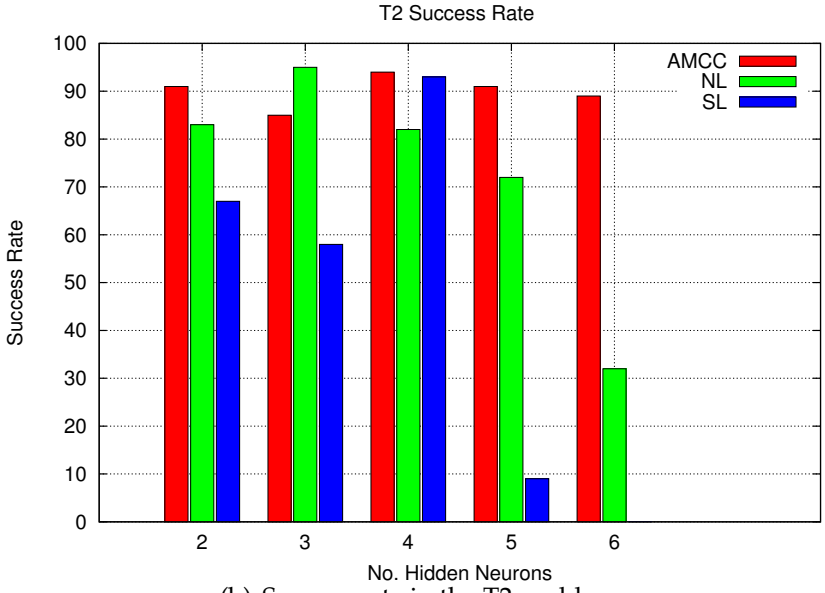


(b) Success rate in the T1 problem

Figure 5.8: The performance of AMCC, NL and SL on different number of hidden neurons for the T1 problem. The optimization time in terms of function evaluations is shown in (a) while the success rate is shown in (b).

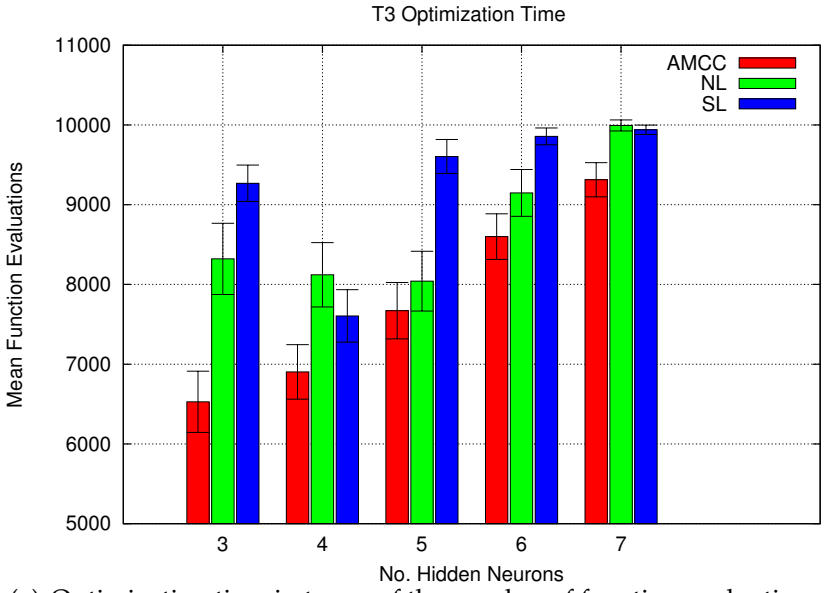


(a) Optimization time in terms of the number of function evaluations

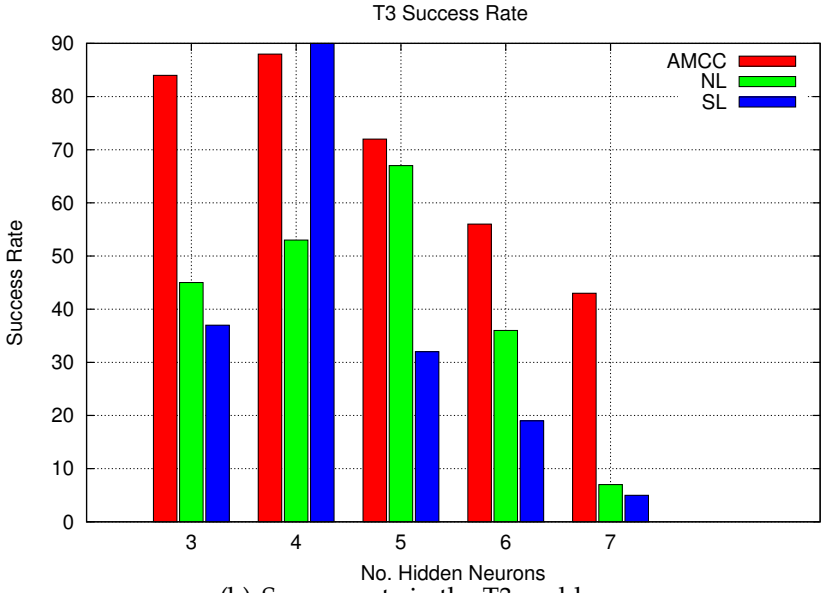


(b) Success rate in the T2 problem

Figure 5.9: The performance of AMCC, NL and SL on different number of hidden neurons for the T2 problem. The optimization time in terms of function evaluations is shown in (a) while the success rate is shown in (b).

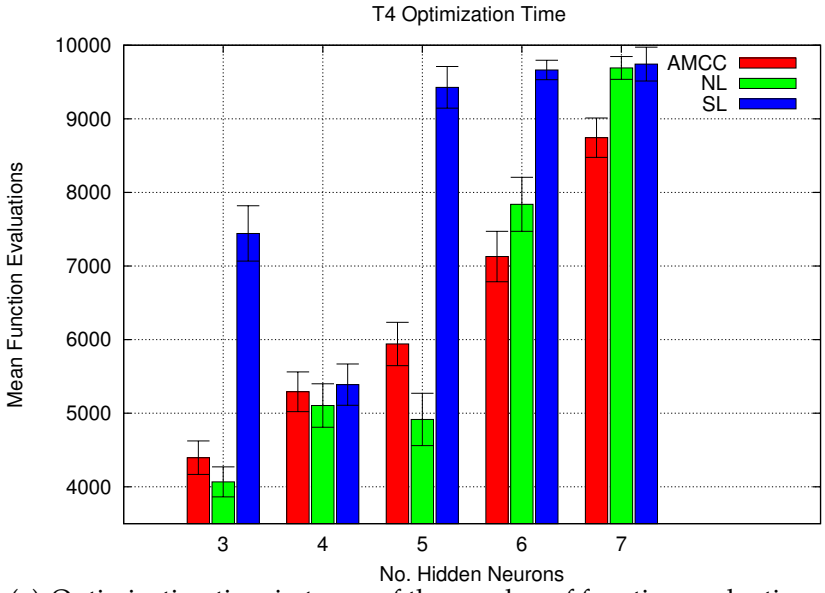


(a) Optimization time in terms of the number of function evaluations

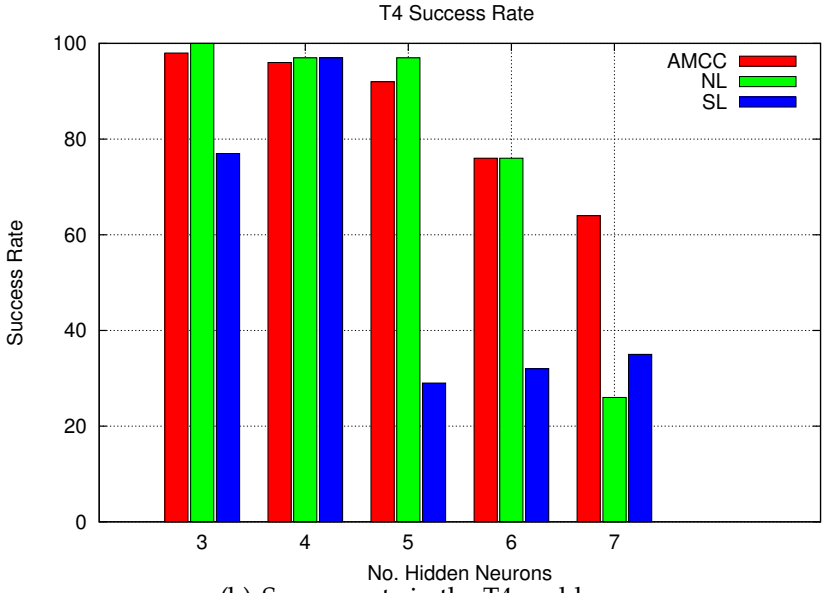


(b) Success rate in the T3 problem

Figure 5.10: The performance of AMCC, NL and SL on different number of hidden neurons for the T3 problem. The optimization time in terms of function evaluations is shown in (a) while the success rate is shown in (b).

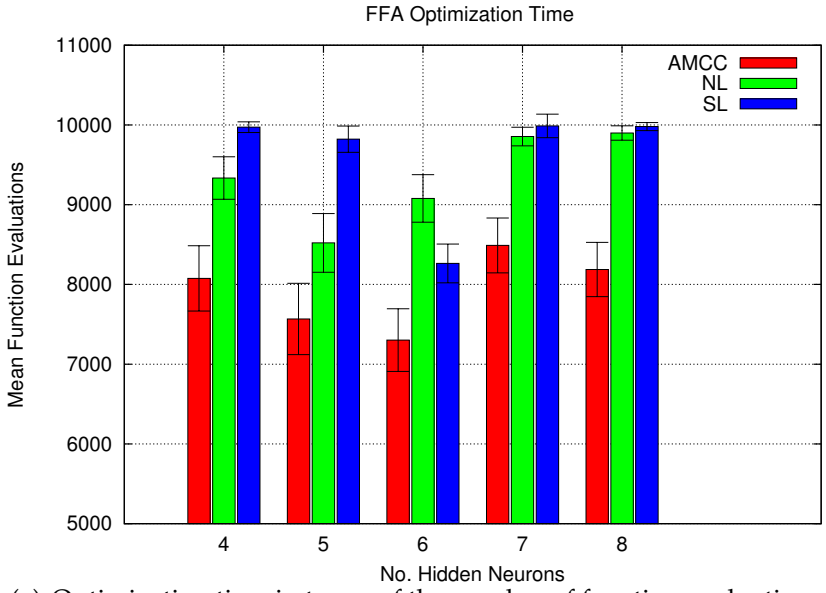


(a) Optimization time in terms of the number of function evaluations

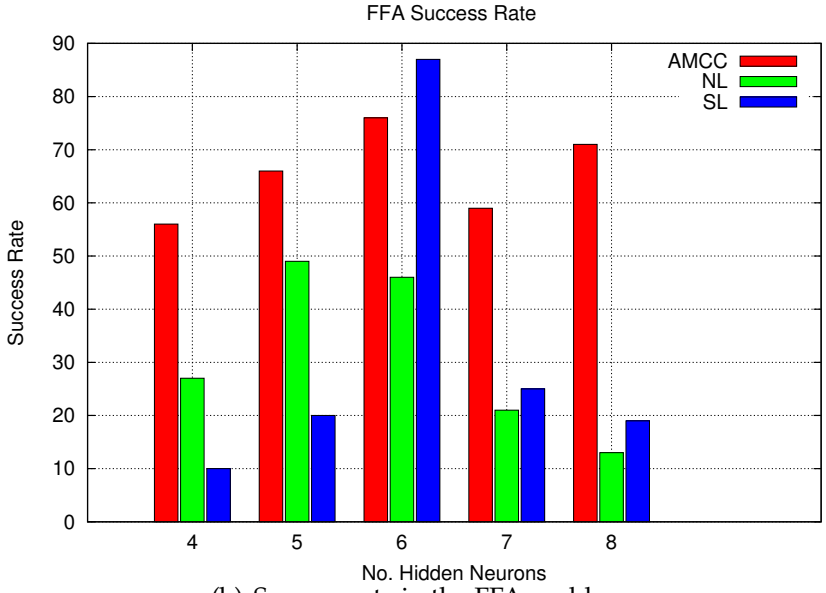


(b) Success rate in the T4 problem

Figure 5.11: The performance of AMCC, NL and SL on different number of hidden neurons for the T4 problem. The optimization time in terms of function evaluations is shown in (a) while the success rate is shown in (b).



(a) Optimization time in terms of the number of function evaluations



(b) Success rate in the FFA problem

Figure 5.12: The performance of AMCC, NL and SL on different number of hidden neurons for the FFA problem. The optimization time in terms of function evaluations is shown in (a) while the success rate is shown in (b).

Chapter 6

Application to Chaotic Time Series Prediction

The aim of this chapter is to use cooperative coevolution problem decomposition methods with adaptation for training recurrent neural networks on chaotic time series problems. Therefore, this chapter is an application of the methods from Chapters 3 and 5. The prediction of chaotic time series is chosen as it has a wide range of applications such as in finance, signal processing, power load, weather forecasting and sunspot prediction.

The chapter begins with an overview of time series prediction and gives details for the cooperative co-evolutionary methods. The simulation is presented and a discussion on the analysis of the results is given.

6.1 Introduction

The prediction of chaotic time series has a wide range of applications such as in finance [38], signal processing [110], power load [108], weather forecast [129], and sunspot prediction [117, 182, 67]. Chaos theory is used to study the behaviour of dynamical systems that are highly sensitive to initial conditions such as noise and error [128, 205]. The initial conditions can make large differences in the behaviour of the system. This is known as the *butterfly effect* which makes long-term prediction difficult.

Artificial neural networks and related computational intelligence methods have been successfully deployed as models for chaotic time series prediction. These include multilayer perception [4, 67], support vector machines [25], radial basis networks [67, 179], fuzzy and neuro-fuzzy methods [111, 103], wavelet neural networks [210], Elman recurrent neural networks [67, 144], evolutionary recurrent neural networks [136], neural fuzzy network with cultural cooperative particle swarm optimisation [125] , nonlinear autoregressive model process with exogenous input (NARX) networks [47, 141] and hybrid of Elman and NARX networks with residual analysis [3].

Adaptation of problem decomposition in different phases of evolution has been effective for training feedforward neural networks on pattern recognition problems and recurrent neural networks on grammatical inference problems in Chapter 5. The results have shown that it is reasonable to adapt the problem decomposition method which enforces different levels of modularity during evolution. They showed that the neural network training problem changes at different stages of evolution in terms of the degree of non-separability.

This chapter employs the adaptive modularity cooperative coevolution method given in Chapter 5 for training recurrent neural networks on chaotic time series problems. Chaotic time series problems have been chosen as they are difficult to model and resemble many real-world predic-

tion problems [128, 110, 67]. Recurrent networks have been chosen for this as they are good for modelling temporal sequences due to their architectural properties. The Elman recurrent neural network [48] is used and three different chaotic time series problems are used which consists of two simulated and one real-world problem. The Lorenz and Mackey-Glass are the simulated time series while the Sunspot is the real-world time series. The G3-PCX evolutionary algorithm is employed in the subpopulations of AMCC. The performance of AMCC is further compared with neuron, synapse and network level problem decomposition methods. The behaviour of the respective methods are evaluated on different neural network topologies which are given by different numbers of hidden neurons. The results are further compared with computational intelligence methods from literature.

6.1.1 Embedding Theorem and Time Series Prediction

Embedding is the process of finding a space in which the dynamics are smooth and no overlaps or intersections occur in the orbits of the attractor. Taken's embedding theorem provides the conditions under which a chaotic time series can be reconstructed into a $D-dimensional$ vector with two conditions which are the *time delay* and the *embedding dimension* [207].

Given an observed time series $x(t)$, an embedded phase space $Y(t) = [(x(t), x(t-T), ..., x(t(D-1)T)]$ can be generated, where, T is the time delay, N is the length of the original time series and D is the embedding dimension, $t = 0, 1, 2, ..., N - DT - 1$ [207].

Taken's theorem [207] expresses that the vector series reproduces many important characteristics of the original time series. The right values for D and T must be chosen in order to efficiently apply Taken's theorem [57]. Taken's proved that if the original attractor is of dimension d, then $D = 2d + 1$ will be sufficient to reconstruct the attractor [207].

Several methods have been proposed in the past to determine the val-

ues for the embedding dimensions as discussed in [24]. Evolutionary algorithms have also been used to determine the optimal values of the embedding dimensions [136, 135].

The reconstructed vector is used to train the recurrent network for one-step-ahead prediction where 1 neuron is used in the input and output layer. In this way, the recurrent network can effectively take in account of the input at the previous time steps. The recurrent network unfolds k steps in time which is equal to the embedding dimension D [117, 144].

The root mean squared error (RMSE) and normalised mean squared error (NMSE) are used to measure the prediction performance of the recurrent neural network. These are given in Equation 6.1 and Equation 6.2.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{6.1}$$

$$NMSE = \left(\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}\right) \tag{6.2}$$

where $y_i, \hat{y}_i$ and $\bar{y}_i$ are the observed data, predicted data and average of observed data, respectively. N is the length of the observed data.

6.2 Adaptation of Modularity

As given in Chapter 5, the general idea behind the adaptive modularity cooperative coevolution (AMCC) method is to use the strength of a different problem decomposition method which reflects on the degree of non-separability when needed during evolution. AMCC employs modularity (problem decomposition or encoding scheme) with a greater level of flexibility (allowing evolution for separable search space) during the initial stage and decreases the level of modularity during the later stages of evolution.

The transformation is from synapse level to neuron level modularity

and finally to network level. Note that network level is standard neuroevolution where one population is used. This is given in the AMCC method in Section 5.2.

The AMCC algorithm given in Chapter 5 worked well for the pattern classification and grammatical inference problems. However, trial experiments revealed that it had problems in time series problem due to the different nature of the problem. Several trial experiments revealed that it would be beneficial to migrate all the individuals of the subpopulations to the next level of evolution. In the version of AMCC given in Chapter 5, only the best individual was migrated to the next level.

The AMCC method presented in this section is similar, however, slightly different. It transforms from one level of modularity into another based on the training time rather than the neural network error. It employs synapse level encoding for the first α portion of the maximum training time. It then transforms into neuron level and then into network level and repeats this level until termination. This is shown in Figure 6.1. The neuron and network level are encoded for β portion of the total training time given by number of function evaluations.

In the transformation from one level of encoding to another, all the individuals of the sub-populations are transferred rather than only the best ones. In the case when the transfer is from synapse to neuron or neuron to network level (transfer into higher levels), the transfer is done by simply merging all the individuals in the same position from the respective sub-populations. The order is maintained and once merged; the fitness of each individual is again evaluated. In the case when the transfer is from network to neuron level (transfer to lower level), then the single population of the network level is disintegrated into sub-populations and then the fitness of the sub-populations is evaluated.

The AMCC method is given in Algorithm 8. Initially, all the sub-populations of the synapse level, neuron level and network level encodings are randomly initialised with random real values in a range. In Stage 1, the sub-

populations at the synapse level encoding are cooperatively evaluated. The Neuron level and network level encodings are left to be cooperatively evaluated at Stage 2.

Stage 1 employs synapse the level encoding where the sub-populations are evolved until α portion of the maximum time. The sub-populations of the synapse level encoding are merged into the neuron level. The individuals with their fitness are transferred to the sub-populations of the neuron level in Stage 2. The algorithm proceeds to the neuron and network level encoding in Stage 2. All the sub-populations are cooperatively evaluated. The neuron level encoding is then evolved for β portion of the maximum time. The sub-populations for the neuron level are then merged into a single population for the network level and evaluated. The network level encoding is evolved for β portion of the maximum time. The population of the network level is then broken down and encoded for the neuron level evolution phase and evaluated. The procedure in Stage 2 is repeated until the maximum training time has been reached or if the minimum network error given by RMSE has been reached.

α and β determines the number of function evaluations the algorithm will evolve at each level. Note that the main termination condition is given by the maximum number of function evaluations. Therefore, alpha determines the portion of training time given to synapse level from the total time. $\beta = (1 - \alpha)/n$, where n is the desired number of transformations in the evolution between neuron and network level. Neuron and network level use the same β as they are meant to evolve for the same number of function evaluations.

Cooperative evaluation of individuals in the respective sub-populations is done by concatenating the chosen individual from a given sub-population with the best individuals from the rest of the sub-populations. The concatenated individual is encoded into the recurrent neural network and the fitness is calculated by the RMSE.

The transition of modularity has to ensure that the information gained

is transferred to the next level of encoding. Note that the number of sub-populations at each level is different. The synapse level encoding has more sub-populations than the neuron level, however, the individuals in the sub-populations of the synapse level must be merged into neuron level. The sub-populations are merged when the transformation is from synapse to neuron level and from the neuron level to the network level. The transformation from the network to neuron level requires the population to be broken down and encoded as neuron level.

Alg. 8 Adaptive Modularity in Cooperative Coevolution

Stage 1: Synapse level encoding (SL)
Cooperatively evaluate Synapse level only

while *FuncEval* $\leq \alpha \times$ *MaxGlobal* **do**
    foreach *each Sub-population at Synapse level* **do**
        Create new offspring
        Cooperative Evaluation
    end
end
Stage 2: Neuron level (NL) and Network level (NetL) encoding
i. Merge individuals from Synapse level into Neuron level
ii. Cooperatively evaluate Neuron level
while *FuncEval* $\leq$ *MaxGlobal* **do**
    while *FuncEval* $\leq \beta \times$ *MaxGlobal* **do**
        foreach *each Sub-population at Neuron level* **do**
            Create new offspring
            Cooperative Evaluation
        end
    end
    i. Merge all individuals into Network level
    ii. Evaluate Network level

    while *FuncEval* $\leq \beta \times$ *MaxGlobal* **do**
        Create new offspring
    end
    i. Break all individuals from Network to Neuron level
    ii. Evaluate Neuron level

end

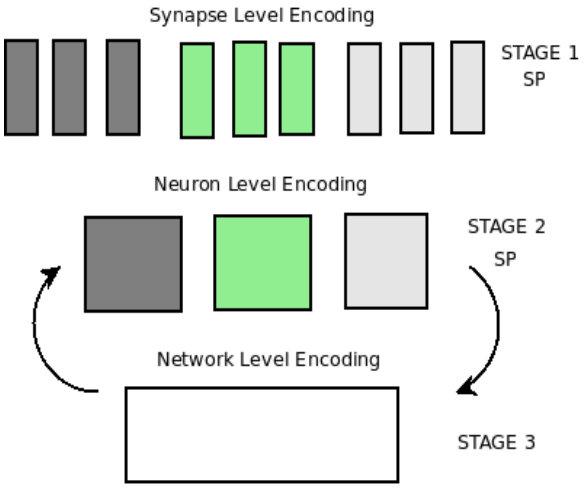


Figure 6.1: The AMCC method used for training RNN on chaotic time series. The sub-populations (SP) at synapse level and neuron level are shown.

6.3 Simulation and Analysis

This section presents an experimental study of AMCC for training recurrent neural networks on chaotic time series problems. The neuron level (NL) and synapse level (SL) and network level (NetL) problem decomposition methods are used for comparison. The Mackey Glass times series [138] and Lorenz time series [128] are the two simulated time series while the real-world problem is the Sunspot time series [195]. The behaviour of the respective methods are evaluated on different recurrent network topologies. The results are further compared with computational intelligence methods from literature. The details of the three different problems are given as follows.

6.3.1 Problem description

The *Mackay Glass time series* has been used in literature as a benchmark problem due to its chaotic nature [138]. The differential equation used to

generate the Mackey Glass time series is given in Equation 6.3.

$$\frac{\delta x}{\delta t} = \frac{ax(t-\tau)}{[1+x^c(t-\tau)]} - bx(t) \tag{6.3}$$

In Equation 6.3, the delay parameter τ determines the characteristic of the time series, where $\tau > 16.8$ produces chaos. The selected parameters for generating the time series is taken from the literature [103, 179, 67, 3] where the constants $a = 0.2$, $b = 0.1$ and $c = 10$. The chaotic time series is generated by using time delay $\tau = 17$ and initial value $x(0) = 1.2$.

The experiments use the chaotic time series with length of 1000 generated by Equation 6.3. The first 500 samples are used for training the Elman network while rest of the 500 samples are used for testing. The time series is scaled in the range [0,1]. The phase space of the original time series is reconstructed with the embedding dimensions $D = 3$ and $T = 2$.

The *Lorenz time series* was introduced by Edward Lorenz who has extensively contributed to the establishment of Chaos theory [128]. The Lorenz equation are given in Equation 6.4.

$$\begin{aligned} \frac{dx(t)}{dt} &= \sigma[y(t) - x(t)] \\ \frac{dy(t)}{dt} &= x(t)[r - z(t)] - y(t) \\ \frac{dz(t)}{dt} &= x(t)y(t) - bz(t) \end{aligned} \tag{6.4}$$

where η, r, and b are dimensionless parameters. The typical values of these parameters are $\eta = 10$, $r = 28$, and $b = 8/3$ [5, 67, 136, 180, 3]. The x-coordinate of the Lorenz time series is chosen for prediction and 1000 samples are generated. The time series is scaled in the range [-1,1]. The first 500 samples are used for training and the remaining 500 is used for testing. The phase space of the original time series is reconstructed with the embedding dimensions $D = 3$ and $T = 2$.

The *Sunspot time series* is a good indication of the solar activities for solar cycles which impacts Earth's climate, weather patterns, satellite and

space missions [182]. The prediction of solar cycles is difficult due to its complexity. The monthly smoothed Sunspot time series has been obtained from the World Data Center for the Sunspot Index [195]. The Sunspot time series from November 1834 to June 2001 is selected which consists of 2000 points. This interval has been selected in order to compare the performance the proposed methods with those from literature [67, 3]. The time series is scaled in the range [-1,1]. The first 1000 samples are used for training while the remaining 1000 samples are used for testing. The phase space of the original time series is reconstructed with the embedding dimensions $D = 5$ and $T = 2$.

Note that the scaling of the three time series in the range of [0,1] and [-1,1] are done as in the literature in order to provide a fair comparison.

6.3.2 Experimental set-up

The Elman recurrent network employs sigmoid neurons in the hidden layer of the three different problems. In the output layer, the sigmoid neurons are used for the Mackey Glass time series while hyperbolic tangent neuron is used for Lorenz and Sunspot time series. The RMSE and NMSE given in Equation 6.1 and Equation 6.2 are used as the main performance measures of the recurrent network.

The G3-PCX evolutionary algorithm [42] is employed in the respective CC methods. The sub-populations are seeded with random real numbers in the range of [-5, 5] in all the experiments. 50 individuals make up the respective sub-populations. The G3-PCX is also used with the same parameters for training the recurrent networks with a single population for network level encoding (NetL).

In the respective CC methods for recurrent networks (SL and NL) shown in Algorithm 8, each sub-population is evolved for a fixed number of generations in a round-robin fashion. This is considered as the *depth of search*. The results in Chapter 3 have shown that the depth of search of 1 genera-

tion gives optimal performance for both NL and SL encodings [29]. Hence, 1 is used as the depth of search in all the experiments. Note that all subpopulations evolve for the same depth of search.

The termination condition of the three problems is when a total of 100 000 function evaluations has been reached by the respective evolutionary training algorithm. $\alpha = 0.2$ and $\beta = 0.1$ in the AMCC method given in Algorithm 8. These values have been determined in trial experiments using the respective time series datasets.

6.3.3 Results and discussion

This section reports the performance of AMCC for training the Elman recurrent network on the chaotic time series problems. Note that the best performance is given by the least RMSE and NMSE.

Initially, the number of hidden neurons is empirically evaluated and the mean and the best value of the RMSE is from 30 experimental runs. The number of hidden neurons directly influences the difficulty of the learning problem. It is more difficult to learn the problem if enough neurons are not present in the hidden layer. The results of the test set are shown in Tables 6.1 - 6.3 where the performances of AMCC, NL, SL, NetL are given. The best results are highlighted in bold.

The best results from Tables 6.1 - 6.3 are chosen and further details are given in Table 6.4 where the mean and 95 % confidence interval (CI) of RMSE and NMSE is given with the best performance out of 30 experimental runs. Note that the mean and CI is expressed using $\pm$ which is equivalent to (μ - CI; μ + CI). The best mean prediction performance on the test dataset is highlighted in Table 6.4 and shown in Figures 6.2 - 6.4.

Table 6.1: The prediction performance (RMSE) on the test dataset of the Lorenz time series

	SL		NL		NetL		AMCC	
Hidden	Mean	Best	Mean	Best	Mean	Best	Mean	Best
3	3.0E-2	1.2E-2	3.6E-2	1.8E-2	5.1E-2	2.1E-2	2.2E-2	1.1E-2
5	**1.9E-2**	6.4E-3	2.7E-2	9.1E-3	2.5E-2	8.6E-3	1.9E-2	6.8E-3
7	2.2E-2	7.0E-3	2.3E-2	9.7E-3	**2.1E-2**	9.4E-3	1.5E-2	7.6E-3
9	3.8E-1	9.8E-3	2.4E-2	5.0E-3	**2.1E-2**	8.7E-3	**1.3E-2**	5.1E-3
11	8.2E-1	1.2E-2	**1.8E-2**	8.2E-3	2.9E-2	1.1E-2	1.6E-2	6.9E-3
13	1.1	2.2E-2	**1.8E-2**	6.6E-3	3.2E-1	2.2E-1	1.7E-2	6.3E-3

Table 6.2: The performance (RMSE) on the test dataset of the Mackey Glass time series

	SL		NL		NetL		AMCC	
Hidden	Mean	Best	Mean	Best	Mean	Best	Mean	Best
3	2.1E-2	1.2E-2	2.3E-2	9.7E-3	3.0E-2	1.3E-2	1.8E-2	8.9E-3
5	1.5E-2	7.9E-3	2.0E-2	1.0E-2	1.5E-2	9.5E-3	1.5E-2	6.9E-3
7	1.4E-2	9.0E-3	1.6E-2	1.0E-2	1.4E-2	8.9E-3	1.2E-2	6.8E-3
9	1.2E-2	7.6E-3	1.4E-2	8.3E-3	**1.2E-2**	7.2E-2	1.2E-2	8.1E-3
11	1.0E-2	5.7E-3	1.3E-2	8.0E-3	2.1E-2	8.8E-3	**1.1E-2**	7.5E-3
13	**9.4E-3**	6.3E-3	**1.2E-2**	8.3E-3	9.6E-2	6.9E-2	1.3E-2	8.0E-3

Table 6.3: The performance (RMSE) on the test dataset of the Sunspot time series

	SL		NL		NetL		AMCC	
Hidden	Mean	Best	Mean	Best	Mean	Best	Mean	Best
3	**6.9E-2**	1.7E-2	**5.6E-2**	2.6E-2	9.2E-2	4.6E-2	**4.4E-2**	2.4E-2
5	1.0E-1	2.4E-2	7.5E-2	2.2E-2	7.2E-2	2.1E-2	7.9E-2	1.7E-2
7	1.0E-1	1.9E-2	5.9E-2	2.2E-2	**5.7E-2**	1.8E-2	8.0E-2	2.4E-2
9	1.3E-1	2.17E-2	8.9E-2	1.7E-2	7.7E-2	1.7E-2	7.4E-2	1.9E-2

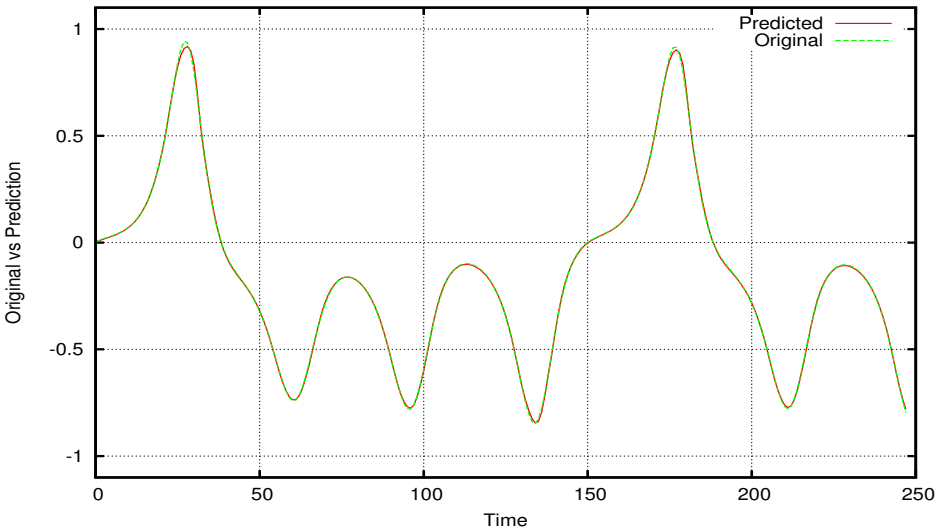
Table 6.4: The performance (RMSE and NMSE) of AMCC, NL, SL, and NetL on the test dataset of the three problems. The mean and 95 % confidence interval (CI) is given with the best performance out of 30 independent experimental runs.

			RMSE		NMSE	
Problem	Method	Hidden	Mean and CI	Best	Mean and CI	Best
Mackey	SL	13	9.39E-3 $\pm$ 5.57E-4	6.33E-3	6.31E-4 $\pm$ 7.60E-5	2.79E-4
	NL	13	1.23E-2 $\pm$ 9.16E-4	8.28E-3	1.11E-3 $\pm$ 1.77E-4	4.77E-4
	EA	9	1.19E-2 $\pm$ 9.47E-4	7.24E-3	1.04E-3 $\pm$ 1.69E-4	3.65E-4
	AMCC	11	1.11E-2 $\pm$ 1.01E-3	7.53E-3	9.17E-4 $\pm$ 1.78E-4	3.90E-4
Lorenz	SL	5	1.95E-2 $\pm$ 2.59E-3	6.36E-3	8.28E-3 $\pm$ 1.98E-3	7.72E-4
	NL	11	1.82E-2 $\pm$ 2.82E-3	8.20E-3	7.48E-3 $\pm$ 2.60E-3	1.28E-3
	EA	7	2.06E-2 $\pm$ 2.79E-3	9.43E-3	9.24E-3 $\pm$ 2.83E-3	1.69E-3
	AMCC	9	1.35E-2 $\pm$ 2.01E-3	5.06E-3	4.04E-3 $\pm$ 1.31E-3	4.88E-4
Sunspot	SL	3	6.88E-2 $\pm$ 2.66E-2	1.66E-2	5.48E-2 $\pm$ 5.19E-2	1.47E-3
	NL	3	5.58E-2 $\pm$ 8.01E-3	2.60E-2	1.92E-2 $\pm$ 5.3eE-3	3.62E-3
	EA	7	5.74E-2 $\pm$ 1.14E-2	1.82E-2	2.28E-2 $\pm$ 1.07E-2	1.76E-3
	AMCC	3	4.39E-2 $\pm$ 5.61E-3	2.41E-2	1.16E-2 $\pm$ 3.05E-3	3.11E-3

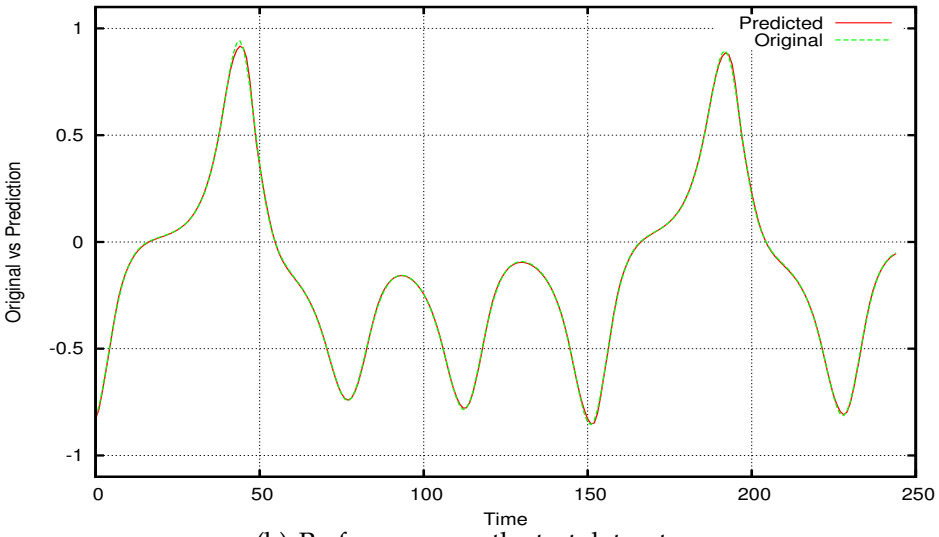
The performances of the given methods are first evaluated in terms of the mean results. In the results for the Mackey time series, SL gives the best performance in terms of the RMSE, however, AMCC performance is close to SL. NL and SL use the same number of hidden neurons (13), however, the performance of SL is better. The results for the Lorenz time series in Table 6.4 show that AMCC gives the best performance with 9 hidden neurons in terms of RMSE when compared to the other methods. Note that each method has its strengths with different number of hidden neurons. SL performs best with the least number of hidden neurons (5) while NL requires the most number of hidden neurons (11 and 13). NetL performs the best with 7 hidden neurons. This is due to the difference in their encoding schemes and the application problem. Although the Mackey and Lorenz time series are both chaotic in nature, they have different characteristics that contribute to the nature of the network training problems in terms of non-separability and the balance between global and local search. In the Sunspot time series, the AMCC gives the best performance with 3 hidden neurons. Note that this is a real-world problem that contains noise.

AMCC has shown to better maintain its performance with different number of hidden neurons as compared to other problem decomposition methods shown in Tables 6.1 - 6.3. This reflects on scalability and robustness. The other methods (SL, NL and NetL) have greater difference when the best results are compared to the worst ones in terms of the number of hidden neurons.

Figures 6.2 - 6.4 show that the RNN has been able to give good prediction performance. The RNN has been able to cope with the noise in the Sunspot time series given in Figure 6.4.

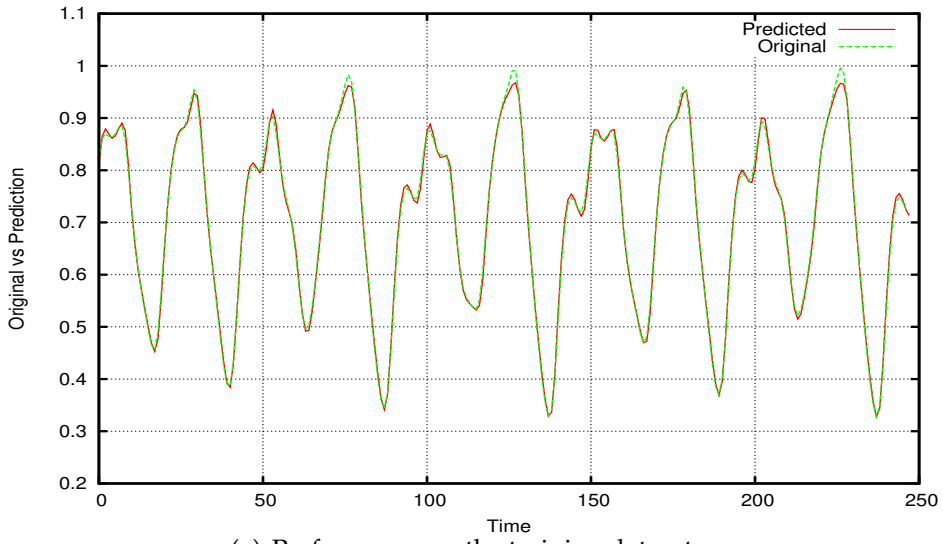


(a) Performance on the training dataset

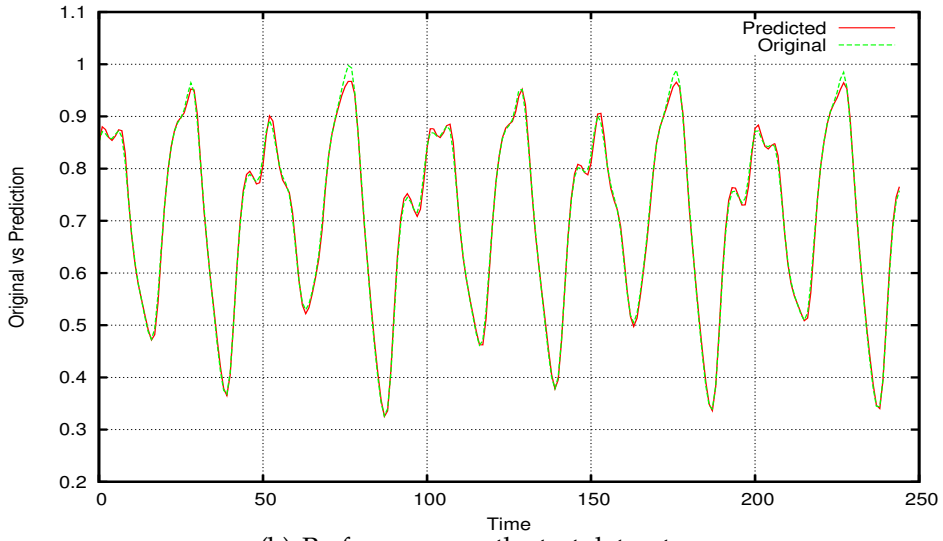


(b) Performance on the test dataset

Figure 6.2: Typical prediction given by AMCC for Lorenz time series

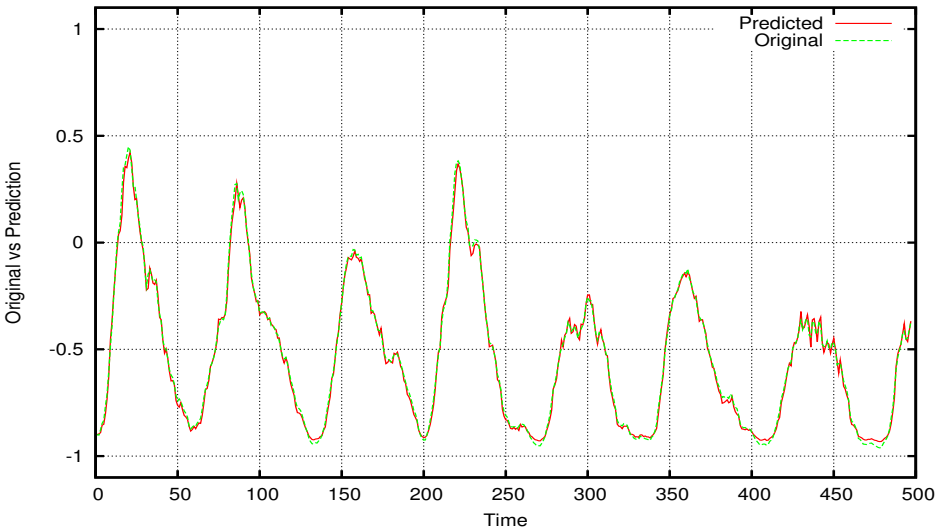


(a) Performance on the training dataset

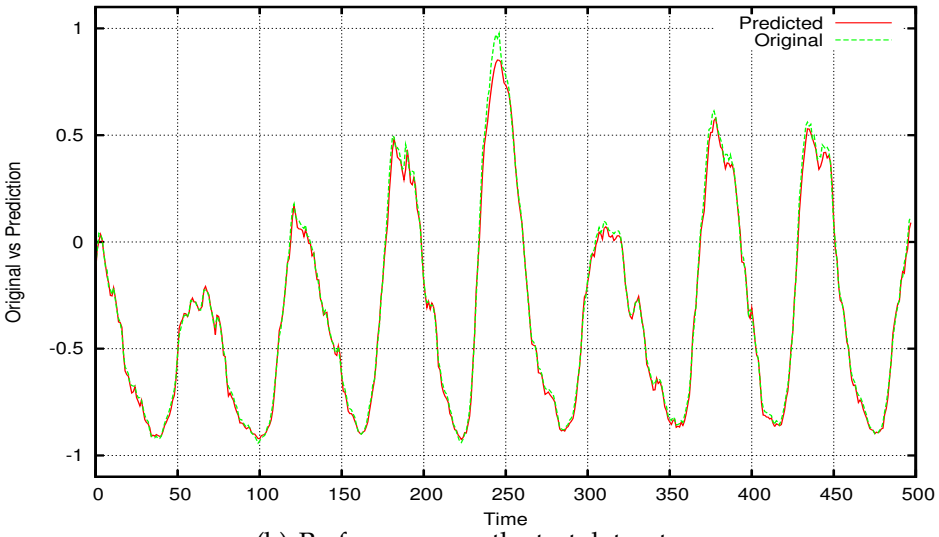


(b) Performance on the test dataset

Figure 6.3: Typical prediction given by AMCC for Mackey Glass time series



(a) Performance on the training dataset



(b) Performance on the test dataset

Figure 6.4: Typical prediction given by AMCC for Sunspot time series

Table 6.5: A comparison with the results from the literature [67, 144, 3] on the Lorenz time series

Prediction Method	RMSE	NMSE
Pseudo Gaussian - radial basis neural network (2002)	9.40E-02	
Radial basis network with orthogonal least squares (RBF-OLS)(2006) [67]		1.41E-09
Locally linear neuro-fuzzy model - Locally linear model tree (LLNF-LoLiMot) (2006) [67]		9.80E-10
Boosted recurrent neural networks (2006) [5]		3.77E-03
Evolutionary RNN (2007) [136]	8.79E-06	9.90E-10
Auto regressive moving average with neural network (ARMA-ANN)(2008) [180]	8.76E-02	
Backpropagation-through-time (BPTT-RNN) (2010) [144]		1.85E-03
Real time recurrent learning (RTRL-RNN) (2010) [144]		1.72E-03
Recursive Bayesian LevenbergMarquardt (RBLM-RNN) (2010) [144]		9.0E-04
Hybrid NARX-Elman RNN with Residual Analysis (2010) [3]	1.08E-04	1.98E-10
Backpropagation neural network and genetic algorithms with residual analysis (2011) [4]	2.96E-02	
Proposed SL-CCRNN	6.36E-03	7.72E-04
Proposed NL-CCRNN	8.20E-03	1.28E-03
Proposed AMCC-RNN	5.06E-03	4.88E-04

Table 6.6: A comparison with the results from the literature [5, 3, 4] on the Mackey Glass time series

Prediction Method	RMSE	NMSE
Autoregressive model	1.9E-01	
Backpropagation neural network	2.0E-02	
Cascade correlation network	6.0E-02	
Adaptive neuro fuzzy inference system (ANFIS) (1993) [103]	1.5E-03	
Genetic fuzzy predictor ensemble (1997) [111]	3.8E-02	
Pseudo Gaussian - radial basis neural network (2002) [179]	2.8E-03	
Auto regressive moving average with neural network (ARMA-ANN)(2008) [180]	2.5E-03	
Radial basis network with orthogonal least squares (RBF-OLS)(2006) [67]	1.02E-03	
Locally linear neuro-fuzzy model - Locally linear model tree (LLNF-LoLiMot) (2006) [67]	9.61E-04	
Boosted recurrent neural networks (2006) [5]		1.60E-04
Evolutionary RNN (2007) [136]		3.15E-08
Neural fuzzy network and hybrid of cultural algorithm and cooperative particle swarm optimisation (CCPSO) (2009) [125]	8.42E-03	
Neural fuzzy network and particle swarm optimisation (PS0) (2009) [125]	2.10E-02	
Neural fuzzy network and cooperative particle swarm optimisation (CPS0) (2009) [125]	1.76E-02	
Neural fuzzy network and differential evolution (DE) (2009) [125]	1.62E-02	
Neural fuzzy network and genetic algorithm (GA) (2009)[125]	1.63E-02	
Hybrid NARX-Elman RNN with Residual Analysis (2010) [3]	3.72E-05	2.70E-08
Backpropagation neural network and genetic algorithms with residual analysis (2011) [4]	1.30E-03	
Proposed SL-CCRNN	6.33E-03	2.79E-04
Proposed NL-CCRNN	8.28E-03	4.77E-04
Proposed AMCC-RNN	7.53E-03	3.90E-04

Table 6.7: A comparison with the results from the literature [67, 3] on the Sunspot time series

Prediction Method	RMSE	NMSE
Sello nonlinear method [182]		3.40E-01
Waldmeier [182]		5.60E-01
McNish-Lincoln [182]		8.00E-02
Multi-layer perceptron (1996) [117]		9.79E-02
Elman RNN (1996) [117]		9.79E-02
FIR Network (MLP) (1996) [117]		2.57E-01
Wavelet packet multilayer perceptron (2001)[210]		1.25E-01
Radial basis network with orthogonal least squares (RBF-OLS)(2006) [67]		4.60E-02
Locally linear neuro-fuzzy model - Locally linear model tree (LLNF-LoLiMot) (2006) [67]		3.20E-02
Hybrid NARX-Elman RNN with Residual Analysis (2010) [3]	1.19E-02	5.90E-04
Proposed SL-CCRNN	1.66E-02	1.47E-03
Proposed NL-CCRNN	2.60E-02	3.62E-03
Proposed AMCC-RNN	2.41E-02	3.11E-03

The performance of the cooperative coevolution methods on the different problems are then compared to some of the results published in literature as shown in Tables 6.5 - 6.7. The best values from the results in Table 6.4 are used to compare with the results from literature. Note that the results from literature use the same dataset with different neural network architectures (feedforward and recurrent). These neural networks are also trained using hybrid training methods and hybrid neural network architectures are also used. Therefore a fair comparison with literature is not possible as the proposed methods only deals with a single training algorithm. Some of the methods from literature have more advantage in terms of network architecture and training algorithms than the proposed methods.

In Table 6.5, the performance of the cooperative coevolution methods are better than some of the existing methods. However, the best results are of the evolutionary recurrent neural network (ERNN)[136] and the Hybrid NARX-Elman network [3]. These are also seen for the Mackey Glass time series given in Table 6.6. This is because the ERNN has also optimised the values for the embedding dimensions for the phase space reconstruction. The Hybrid NARX-Elman network has the advantage of architectural properties of the two methods (NARX and Elman network) with residual analysis which further improves the results. These methods can be combined with the proposed method in future work to improve the results.

In Table 6.6, the proposed methods have given better performance than similar evolutionary approaches such as training neural fuzzy networks with hybrid of cultural algorithms and cooperative particle swarm optimisation (CCPSO), cooperative particle swarm optimisation (CPSO), genetic algorithms and differential evolution (DE) [125]. In Table 6.7, the proposed methods have given better performance than most of the methods from literature with the only exception being the Hybrid NARX-Elman networks [3].

6.4 Chapter Summary

This chapter has introduced problem decomposition in cooperative coevolution to the field of chaotic time series prediction. The original time series have been embedded in state space using the embedding theorem. Recurrent neural networks have been trained by cooperative co-evolutionary problem decomposition methods for the prediction of chaotic time series which include Lorenz, Mackey Glass and Sunspot time series.

The cooperative coevolution methods for chaotic time series problems have shown promising results. The next chapter concludes the thesis with discussion on future research.

Chapter 7

Conclusions and Future Work

Problem decomposition and adaptation have been the major focus of the thesis. This chapter concludes the results and discussion from Chapters 3-6 with directions for future research.

7.1 Conclusions

Problem decomposition, local search and modularity adaptation have been used for enhancing cooperative neuro-evolution with application for the prediction of chaotic time series. The overall research goal of this thesis was to explore and improve problem decomposition and adaptation in cooperative neuro-evolution of feedforward and recurrent networks. The overall research goal has been achieved and the following conclusions have been reached for each of the contributions in the thesis.

7.1.1 Problem decomposition

In Chapter 3, an analysis on the degree of non-separability for feedforward networks has been given. A problem decomposition method called neuron-based sub-population (NSP) has been introduced and used for training feedforward and recurrent neural networks.

1. In feedforward networks, NSP has achieved better performance than other encoding schemes in terms of the optimisation time and the success rate for all the given problems. The major advantage of NSP is that it can represent the same problem in a smaller number of subcomponents than the synapse level encoding and at the same time it provides better training performance. NSP also exhibits good scaling properties for different numbers of hidden neurons.

2. In recurrent networks, NSP has shown better properties than synapse level encoding in terms of the total optimisation time for grammatical inference problems. There have been some cases where the synapse level encoding performed better in terms of the success rate. This has been observed for different network topologies, which indicates that the degree of non-separability also changes with the size of the neural network. In these cases, the synapse level encoding has been more appropriate. This has motivated the adaptation of modularity during evolution for a more robust method.

3. In feedforward networks, NSP has shown to efficiently decompose the problem by grouping interacting variables into separate subcomponents. The depth of search directly relates to the degree of non-separability. A good performance with a large depth of search indicates that there is little interaction among the subcomponents during evolution and the problem decomposition method has been effective. This has also been the case for recurrent neural networks.

4. In feedforward networks, NSP provided better performance than CME due to the following reasons. 1) NSP provides more diversity than CME, hence it is less prone to local convergence. 2) NSP groups the interacting variables in the output layer weights which is not done by CME. It has been shown in Section 3.1.1 that the output layer weights interact with each other and it is important to group them as done in NSP.

5. Synapse level has more diversity, but does not have any feature of grouping interacting variables. The neural network problem is partially separable and requires interacting variables to be grouped which is not possible through synapse level problem decomposition.

7.1.2 Memetic cooperative neuro-evolution

In Chapter 4, a novel method for incorporating local search into a memetic cooperative neuro-evolution method has been introduced for evolving feedforward and recurrent networks. Crossover-based local search has been used as the local search method. .

The main problem in this chapter was to efficiently utilise local search in the respective sub-populations. This problem has been efficiently solved through the framework which decomposes the locally refined solution and incorporates it into the sub-populations. The memetic framework progresses with a global search through diversity given by the sub-populations of cooperative coevolution, and as a specified time is reached (in terms of local search interval), the algorithm incorporates local search into the sub-populations.

1. In feedforward networks, the memetic cooperative coevolution method has performed better for all the given problems when compared to the performance of cooperative coevolution alone. This opens the road for further research in using other local search methods.

2. In all of the problems used in feedforward and recurrent networks, the best results have been given when the memetic cooperative coevolution method used the local search interval of 1 implying that local search is most frequently needed. The local search intensity must be tailored for different problems. The memetic cooperative coevolution method gave better performance than conventional cooperative neuro-evolution. The method is general, therefore, any evolutionary

algorithm in the subpopulations can be used with any suitable local search depending on the application.

3. The memetic framework also provides the feature of co-adaptation among the sub-populations of cooperative coevolution using the local search population. This local search population provides the means for selected individuals to be exchanged with different subpopulations using the crossover operation in the local search population.

7.1.3 Adaptation of Modularity

In Chapter 5, the adaptation of modularity in cooperative neuro-evolution is done by employing different levels of problem decomposition in the evolutionary process.

1. In feedforward and recurrent networks, the adaptive modularity method gave better performance than the neuron and synapse level problem decomposition. The proposed adaptive modularity method adapts from one level of problem decomposition to another based on the change of error of the neural network during the evolutionary process. The heuristic for determining the change in modularity reduces the additional computational cost of parameter setting for the network architecture and the problem.

2. The main strength of the AMCC in feedforward and recurrent networks is its ability to perform better than the other methods when the problem size has been increased. AMCC performed better than the other methods with different number of hidden neurons. The nature of some problems changes when the dimension of the problem is increased. For instance, the Rosenbrock function is uni-modal for 3 dimensions and multi-modal for greater than 3 dimensions. The change in the number of hidden neurons of the network also alters

its nature in terms of multi-modality and adaptation is necessary in the evolutionary process.

3. The results have also shown that the nature of neural network training changes during evolution in terms of the degree of non-separability increases which further validates the analysis shown in Section 3.1.1 of Chapter 3.

7.1.4 Application to Chaotic Time Series Prediction

In Chapter 6, the cooperative coevolution problem decomposition methods with adaptive modularity have been used for training recurrent neural networks for chaotic time series predictions. The Lorenz, Mackey Glass and real-world Sunspot time series have been used and the results have been compared to other computational methods from literature.

1. The two different problem decomposition methods (synapse and neuron level) in cooperative coevolution have performed better than a standard evolutionary algorithm (network level) in general. The adaptation of problem decomposition has shown to be beneficial in some cases. The results show that the given methods have been able to predict chaotic time series with good level of accuracy. The coevolutionary methods have performed very well on the real world Sunspot time series that contain noise.

2. The comparison of results with literature has shown that the proposed cooperative coevolution methods performed better than several other methods. However, it has not outperformed some methods that have additional enhancements such as the optimisation of the embedding dimensions and strength of architectural properties of hybrid neural networks with residual analysis [136, 3]. These can be added to cooperative coevolution to further improve the results. The cooperative coevolution methods outperformed several other

methods on the real-world Sunspot time series that contained noise. This reflects on robustness.

7.2 Further Findings and Discussion

Apart from the major conclusions which have fulfilled the research goals, there are other findings which are discussed as follows.

1. In neural networks, interacting variables exist that determine the degree of non-separability. The degree of non-separability depends on the particular neural network architecture and application problem. Different degrees of non-separability are exhibited by the different problem decomposition methods as they give varied performance for the problems in feedforward and recurrent networks. The synapse level encoding views the neural network as a fully separable problem as it employs a separate subcomponent for each synapse. Network level encoding views the network as fully non-separable, while ESP and NSP views the network as partially separable. The success of NSP shows that the given neural network problems are partially separable.

2. The depth of search in the subcomponents made a major difference for synapse level encoding in feedforward and recurrent networks. A significant difference in performance was not observed in NSP than the other neuron level encodings such as CME and ESP. This implies that the neuron level encodings are able to group the interacting variables more efficiently in the partially separable neural network problem. The synapse level encoding does not provide any grouping of interacting variables. Therefore, the depth of 1 generation enables the synapse level encoding to deal with the higher level of separability. A larger depth of search in synapse level encoding would have given good performance if the problem was fully separable.

3. In both cases (feedforward and recurrent networks), it has been observed that the nature of the problem changes as the problem is being learnt. Apart from the problem nature in terms of multi-modality, the nature of the problem changes in terms of the degree of separability. The success of the proposed adaptive modularity method indicates that during the later stages of evolution, the bond between the interacting variables increases and the problem becomes more non-separable which requires larger subcomponents, and finally, evolution with a single population only. This is the main reason that standard cooperative co-evolution problem decomposition methods like neuron level and synapse level encoding failed to perform as they had a fixed problem decomposition method throughout entire evolution.

7.3 Limitations

The limitations of the methods developed in this thesis are as follows.

1. In the proposed cooperative coevolution methods, there has not been any means of ensuring that diversity is retained in the search process. Diversity can be maintained by employing some distance measurements among the individuals in the sub-populations and also by adapting the number of hidden neurons during evolution.

2. The results have been presented in a way where premature convergence is separated from late convergence. The success rate measures the robustness of the methods, however it does not distinguish the type of convergence (either premature or later convergence) in unsuccessful runs. This measurement can help in analyzing the existing approaches to further enhance them in future research.

3. In AMCC, it would be useful to examine the number of times the ChangeMod is fired in the evolutionary process, i.e, the number of

times NL and NetL levels are employed before convergence. This information will be useful in designing improvements in the AMCC method in future research.

4. Although evolutionary computation methods are known to be global optimization methods; their use for neuro-evolution would require more computational time when compared with gradient based methods. The application of these methods is intended for problems where the use of gradient descent based methods have local convergence.

7.4 Future Research Directions

1. The proposed memetic cooperative coevolution framework has performed better for all the given problems when compared to the performance of standard cooperative coevolution. This opens the road for further research in using other local search methods. In the case of training neural networks, back-propagation can also be utilised. It can replace the crossover-based local search or be used as an additional tool. This will enable the memetic cooperative coevolution method to incorporate gradient information from back-propagation into the evolutionary search process. In the case of recurrent neural networks, backpropagation-through time can be used.

2. The interacting variables from the training data also has some effect on neural network training. This has been observed for the four different problems for the case of feedforward networks where the different modularity adaptation methods (2-Stage and 3-Stage) showed different performance exhibiting their strengths and limitations. For instance, in the 4-bit parity problem, the 3-Stage framework has been effective and for the Wine classification, the 2-Stage framework has been effective. This shows that the training data is a major factor in determining the change of modularity. Hence, future work can be

directed towards building neural network architectures and training algorithms that consider the nature of the training data; i.e. dependencies among the attributes in the training data.

3. Neural network training requires a global search in the initial stage and a local search in the final stage for further refining the solution. The two main issues of concern, 1) global and local search and 2) degree of non-separability. In most problems, a global search is needed in the beginning and refinements using local search is required during the final stages. The second issue is the balance in the degree of the non-separability. There seems to be a relationship between these global and local search and the degree of non-separability. Future research questions can relate on how the degree of non-separability affects global-local search.

4. The performance of cooperative coevolution on chaotic time series problems compares well with the results given by other computational intelligence techniques from literature. This motivates further research in using evolutionary computation methods for chaotic time series prediction. The results can be further improved by incorporating boosting techniques, gradient based local search, residual analysis and evolving the neural network topology during evolution.

5. In future work, it will be interesting to apply the proposed methods for other real-world applications such as speech recognition, gesture recognition and signature verification using neural networks. The same method can also be used for training other neural network architectures and also be extended for general global optimisation problems.

Bibliography

[1] A., K., AND A., H. J. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems* (1992), pp. 950–957.

[2] ANGELINE, P., SAUNDERS, G., AND POLLACK, J. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks 5*, 1 (1994), 54 –65.

[3] ARDALANI-FARSA, M., AND ZOLFAGHARI, S. Chaotic time series prediction with residual analysis method using hybrid elman-narx neural networks. *Neurocomputing 73*, 13-15 (2010), 2540 – 2553.

[4] ARDALANI-FARSA, M., AND ZOLFAGHARI, S. Residual analysis and combination of embedding theorem and artificial intelligence in chaotic time series forecasting. *Appl. Artif. Intell. 25* (January 2011), 45–73.

[5] ASSAAD, M., BON, R., AND CARDOT, H. Predicting chaotic time series by boosted recurrent neural networks. In *Neural Information Processing*, I. King, J. Wang, L.-W. Chan, and D. Wang, Eds., vol. 4233 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 831–840.

[6] ASUNCION, A., AND NEWMAN, D. UCI machine learning repository, 2007.

[7] AUGER, A., AND HANSEN, N. Performance Evaluation of an Advanced Local Search Evolutionary Algorithm. In *IEEE Congress on Evolutionary Computation* (2005), pp. 1777–1784.

[8] BAKER, J. E. Reducing bias and inefficiency in the selection algorithm. In *Proc. of the 2nd Intl Conf on GA* (1987), Lawrence Erlbaum Associates, Inc. Mahwah, NJ, USA, pp. 14–21.

[9] BARRICELLI, N. A. Esempi numerici di processi di evoluzione, 1954.

[10] BAUM, E. B., AND HAUSSLER, D. What size net gives valid generalization? *Neural Comput. 1* (March 1989), 151–160.

[11] BEASLEY, D., BULL, D. R., AND MARTIN, R. R. A sequential niche technique for multimodal function optimization. *Evol. Comput. 1*, 2 (1993), 101–125.

[12] BELLMAN, R. E. *Adaptive control processes: a guided tour*. Princeton University Press, 1961.

[13] BENGIO, Y., SIMARD, P., AND FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks 5*, 2 (1994), 157–166.

[14] BENNETT, K. P., AND PARRADO-HERNÁNDEZ, E. The interplay of optimization and machine learning research. *J. Mach. Learn. Res. 7* (December 2006), 1265–1281.

[15] BLANCO, A., DELGADO, M., AND PEGALAJAR, M. C. A real-coded genetic algorithm for training recurrent neural networks. *Neural Netw. 14*, 1 (2001), 93–105.

[16] BLANCO, A., DELGADO, M., AND PEGALAJAR, M. C. A real-coded genetic algorithm for training recurrent neural networks. *Neural Netw. 14*, 1 (2001), 93–105.

[17] BLUM, A. L., AND RIVEST, R. L. Training a 3-node neural network is np-complete. *Neural Networks 5*, 1 (1992), 117 – 127.

[18] BLUM, C., AND ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv. 35* (September 2003), 268–308.

[19] BLUM, C., AND ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv. 35* (2003), 268–308.

[20] BROOKSHEAR, J. G. *Theory of computation: formal languages, automata, and complexity*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1989.

[21] BUK, Z., KOUTNÍK, J., AND SNOREK, M. Neat in hyperneat substituted with genetic programming. In *ICANNGA* (2009), pp. 243–252.

[22] CAI, X., ZHANG, N., VENAYAGAMOORTHY, G. K., AND WUNSCH, II, D. C. Time series prediction with recurrent neural networks trained by a hybrid PSO-EA algorithm. *Neurocomput. 70*, 13-15 (2007), 2342–2353.

[23] CANT-PAZ, E., AND KAMATH, C. An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. *IEEE Transactions on systems, man and cybernetics-Part B: Cybernetics 35*, 5 (2005), 915–933.

[24] CAO, L. Practical method for determining the minimum embedding dimension of a scalar time series. *Physica D: Nonlinear Phenomena 110*, 1-2 (1997), 43 – 50.

[25] CAO, L. Support vector machines experts for time series forecasting. *Neurocomputing 51* (2003), 321 – 339.

[26] CASTAO, M. A., VIDAL, E., AND CASACUBERTA, F. Finite state automata and connectionist machines: A survey. In *IWANN* (1995), J. Mira and F. S. Hernndez, Eds., Lecture Notes in Computer Science, Springer, pp. 433–440.

[27] CHANDRA, R., FREAN, M., AND ZHANG, M. An encoding scheme for cooperative coevolutionary feedforward neural networks. In *AI 2010: Advances in Artificial Intelligence*, J. Li, Ed., vol. 6464 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2011, pp. 253–262.

[28] CHANDRA, R., FREAN, M., ZHANG, M., AND OMLIN, C. Building subcomponents in the cooperative coevolution framework for training recurrent neural networks. Technical Report ECSTR09-14, School of Engineering and Computer Science, Victoria University of Wellington, New Zealand, 2009.

[29] CHANDRA, R., FREAN, M., ZHANG, M., AND OMLIN, C. W. Encoding subcomponents in cooperative co-evolutionary recurrent neural networks. *Neurocomputing In Press* (2011).

[30] CHANDRA, R., AND OMLIN, C. W. Training and extraction of fuzzy finite state automata in recurrent neural networks. In *Proc. of International Conference on Computational Intelligence* (2006), pp. 274–279.

[31] CHANDRA, R., AND OMLIN, C. W. The comparison and combination of genetic and gradient descent learning in recurrent neural networks: An application to speech phoneme classification. In *Proc. of International Conference on Artificial Intelligence and Pattern Recognition* (2007), pp. 286–293.

[32] CHELOUAH, R., AND SIARRY, P. Genetic and nelder-mead algorithms hybridized for a more accurate global optimization of contin-

uous multiminima functions. *European Journal of Operational Research 148*, 2 (2003), 335 – 348. Sport and Computers.

[33] CHEN, W., WEISE, T., YANG, Z., AND TANG, K. Large-scale global optimization using cooperative coevolution with variable interaction learning. In *Proceedings of the 11th international conference on Parallel problem solving from nature: Part II* (Berlin, Heidelberg, 2010), Springer-Verlag, pp. 300–309.

[34] COBB, H. G., AND GREFENSTETTE, J. J. Genetic algorithms for tracking changing environments. In *Proceedings of the 5th International Conference on Genetic Algorithms* (San Francisco, CA, USA, 1993), Morgan Kaufmann Publishers Inc., pp. 523–530.

[35] COLSON, B., AND TOINT, P. L. Optimizing partially separable functions without derivatives. *Optimization Methods and Software 20*, 4 (2005), 493–508.

[36] CURRY, B., AND MORGAN, P. Neural networks: a need for caution. *Omega 25*, 1 (1997), 123 – 133.

[37] C.W.J., AND GRANGER. Long memory relationships and the aggregation of dynamic models. *Journal of Econometrics 14*, 2 (1980), 227 – 238.

[38] DAS, A., AND DAS, P. Chaotic analysis of the foreign exchange rates. *Applied Mathematics and Computation 185*, 1 (2007), 388 – 396.

[39] DAWKINS, R. Oxford University Press, 1976.

[40] DE LA HIGUERA, C. A bibliographical study of grammatical inference. *Pattern Recognition 38*, 9 (2005), 1332 – 1348.

[41] DEB, K., AND AGRAWAL, R. B. Simulated Binary Crossover for Continuous Search Space. Tech. rep., 1994.

[42] DEB, K., ANAND, A., AND JOSHI, D. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evol. Comput. 10*, 4 (2002), 371–395.

[43] DEEP, K., AND THAKUR, M. A new crossover operator for real coded genetic algorithms. *Applied Mathematics and Computation 188*, 1 (2007), 895 – 911.

[44] DEJONG, K. A. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, 1975.

[45] DELGADO, M., AND DEL CARMEN PEGALAJAR JIMNEZ, M. A multiobjective genetic algorithm for obtaining the optimal size of a recurrent neural network for grammatical inference. *Pattern Recognition 38*, 9 (2005), 1444–1456.

[46] DELGADO, M., AND PEGALAJAR, M. A multiobjective genetic algorithm for obtaining the optimal size of a recurrent neural network for grammatical inference. *Pattern Recognition 38*, 9 (2005), 1444 – 1456.

[47] DIACONESCU, E. The use of narx neural networks to predict chaotic time series. *WSEAS Trans. Comp. Res. 3* (March 2008), 182–191.

[48] ELMAN, J. L. Finding structure in time. *Cognitive Science 14* (1990), 179–211.

[49] ELMAN, J. L., AND ZIPSER, D. Learning the hidden structure of speech. *The Journal of the Acoustical Society of America 83*, 4 (1988), 1615–1626.

[50] ENGEL, A. Complexity of learning in artificial neural networks. *Theoretical Computer Science 265*, 1-2 (2001), 285 – 306.

[51] ESHELMAN, L. J., AND SCAHFFER, J. D. Real-coded genetic algorithms and interval-schemata. *Foundations of Genetic Algorithms 2* (1993), 187–202.

[52] FAN, H., LU, J. W., AND XU, Z. An empirical comparison of three novel genetic algorithms. *Engineering Computations 17* (2000), 981–1002.

[53] FISH, K. E., JOHNSON, J. D., DORSEY, R. E., AND BLODGETT, J. G. Using an artificial neural network trained with a genetic algorithm to model brand share. *Journal of Business Research 57*, 1 (2004), 79 – 85.

[54] FOGEL, L., OWENS, A., AND WALSH, M. John Wiley, 1966.

[55] FRASCONI, P., GORI, M., AND TESI, A. Successes and failures of backpropagation: a theoretical investigation. In *Progress in Neural Networks. Ablex Publishing* (1993), Ablex Publishing, pp. 205–242.

[56] FRASER, A. Simulation of genetic systems by automatic digital computers. i. introduction. *Australian Journal of Biological Sciences 10* (1957), 484–491.

[57] FRAZIER, C., AND KOCKELMAN, K. Chaos theory and transportation systems: Instructive example. *Transportation Research Record: Journal of the Transportation Research Board 20* (2004), 9–17.

[58] GABRIJEL, I., AND DOBNIKAR, A. On-line identification and reconstruction of finite automata with generalized recurrent neural networks. *Neural Netw. 16*, 1 (2003), 101–120.

[59] GANG, P., IIMURA, I., AND NAKAYAMA, S. Application of genetic recombination to genetic local search in tsp, 2007.

[60] GARCÍA-MARTÍNEZ, C., AND LOZANO, M. *Advances in Metaheuristics for Hard Optimization*. Springer, 2008, ch. Local search based on genetic algorithms.

[61] GARCIA-PEDRAJAS, N., HERVAS-MARTINEZ, C., AND MUNOZ-PEREZ, J. Multi-objective cooperative coevolution of artificial neural networks (multi-objective cooperative networks). *Neural Netw. 15*, 10 (2002), 1259–1278.

[62] GARCIA-PEDRAJAS, N., HERVAS-MARTINEZ, C., AND MUNOZ-PEREZ, J. COVNET: a cooperative coevolutionary model for evolving artificial neural networks. *IEEE Transactions on Neural Networks 14*, 3 (2003), 575–596.

[63] GARCIA-PEDRAJAS, N., HERVAS-MARTINEZ, C., AND ORTIZ-BOYER, D. Cooperative coevolution of artificial neural network ensembles for pattern classification. *IEEE Transactions on Evolutionary Computation 9*, 3 (2005), 271–302.

[64] GARCÍA-PEDRAJAS, N., AND ORTIZ-BOYER, D. A cooperative constructive method for neural networks for pattern recognition. *Pattern Recogn. 40*, 1 (2007), 80–98.

[65] GARCÍA-PEDRAJAS, N., AND ORTIZ-BOYER, D. A cooperative constructive method for neural networks for pattern recognition. *Pattern Recogn. 40*, 1 (2007), 80–98.

[66] GARCA-MARTNEZ, C., LOZANO, M., HERRERA, F., MOLINA, D., AND SNCHEZ, A. Global and local real-coded genetic algorithms based on parent-centric crossover operators. *European Journal of Operational Research 185*, 3 (2008), 1088 – 1113.

[67] GHOLIPOUR, A., ARAABI, B. N., AND LUCAS, C. Predicting chaotic time series using neural and neurofuzzy models: A comparative study. *Neural Process. Lett. 24* (2006), 217–239.

[68] GILES, C. L., HORNE, B. G., AND LIN, T. Learning a class of large finite state machines with a recurrent neural network. *Neural Networks 8*, 9 (1995), 1359 – 1365.

[69] GILES, C. L., MILLER, C. B., CHEN, D., CHEN, H. H., SUN, G. Z., AND LEE, Y. C. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Comput. 4* (May 1992), 393–405.

[70] GILES, C. L., OMLIN, C., AND THORNBER, K. K. Equivalence in knowledge representation: Automata, recurrent neural networks, and dynamical fuzzy systems. *Proceedings of the IEEE 87*, 9 (1999), 1623–1640.

[71] GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res. 13* (1986), 533–549.

[72] GLOVER, F. W., AND KOCHENBERGER, G. A. *Handbook of Metaheuristics* . Springer, 2003.

[73] GOLDBERG, D. E. Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems 5* (1991), 139–167.

[74] GOLDBERG, D. E., AND RICHARDSON, J. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application* (Hillsdale, NJ, USA, 1987), pp. 41–49.

[75] GOMEZ, F., AND MIKKULAINEN, R. Incremental evolution of complex general behavior. *Adapt. Behav. 5*, 3-4 (1997), 317–342.

[76] GOMEZ, F., AND MIKKULAINEN, R. Incremental evolution of complex general behavior. *Adapt. Behav. 5*, 3-4 (1997), 317–342.

[77] GOMEZ, F., SCHMIDHUBER, J., AND MIIKKULAINEN, R. Accelerated neural evolution through cooperatively coevolved synapses. *J. Mach. Learn. Res. 9* (2008), 937–965.

[78] GOMEZ, F. J. Robust non-linear control through neuroevolution. Technical Report AI-TR-03-303, PhD Thesis, Department of Computer Science, The University of Texas at Austin, 2003.

[79] GOMEZ, F. J., AND SCHMIDHUBER, J. Co-evolving recurrent neurons learn deep memory pomdps. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation* (New York, NY, USA, 2005), ACM, pp. 491–498.

[80] GOUDREAU, M., GILES, C., CHAKRADHAR, S., AND CHEN, D. First-order versus second-order single-layer recurrent neural networks. *Neural Networks, IEEE Transactions on 5*, 3 (May 1994), 511 –513.

[81] GREEWOOD, G., FOGEL, G., AND CIOBANU, M. Emphasizing extinction in evolutionary programming. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on* (1999), vol. 1, pp. 666–671.

[82] GRIEWANK, A., AND TOINT, P. L. Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik 39* (1982), 119–137. 10.1007/BF01399316.

[83] HANSEN, N., MÜLLER, S. D., AND KOUMOUTSAKOS, P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evol. Comput. 11* (March 2003), 1–18.

[84] HANSEN, N., AND OSTERMEIER, A. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput. 9*, 2 (2001), 159–195.

[85] HANSEN, N., AND OSTERMEIER, A. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput. 9*, 2 (2001), 159–195.

[86] HAYKIN, S. *Neural Networks and Learning Machines*. Prentice Hall, 2008.

[87] HEIDRICH-MEISNER, V., AND IGEL, C. Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms 64*, 4 (2009), 152 – 168. Special Issue: Reinforcement Learning.

[88] HERRERA, F., AND LOZANO, M. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation 4* (2000), 43–63.

[89] HERRERA, F., AND LOZANO, M. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation 4* (2000), 43–63.

[90] HERRERA, F., LOZANO, M., AND VERDEGAY, J. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review 12* (1998), 265–319. 10.1023/A:1006504901164.

[91] HERTZ, J., KROGH, A., AND PALMER, R. G. *Introduction to the theory of neural computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1991.

[92] HOCHREITER, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 6*, 2 (1998), 107–116.

[93] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural Comput. 9*, 8 (1997), 1735–1780.

[94] HOHIL, M. E., LIU, D., AND SMITH, S. H. Solving the n-bit parity problem using neural networks. *Neural Networks 12*, 9 (1999), 1321 – 1323.

[95] HOLLAND, J. H. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[96] HORNIK, K., STINCHCOMBE, M., AND WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks 2* (July 1989).

[97] HORNIK, K., STINCHCOMBE, M., AND WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 5 (1989), 359–366.

[98] HUNG, S. L., AND ADELI, H. Parallel genetic/neural network learning algorithm for mimd shared memory machines. *IEEE Trans. Neural Networks 5* (1994), 900–909.

[99] IGEL, C. Neuroevolution for reinforcement learning using evolution strategies. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on* (8-12 2003), vol. 4, pp. 2588 – 2595 Vol.4.

[100] ISHU, K., VAN DER ZANT, T., BECANOVIC, V., AND PLOGER, P. Identification of motion with echo state network. In *OCEANS '04. MTTS/IEEE TECHNO-OCEAN '04* (2004), vol. 3, pp. 1205 – 1210.

[101] JAEGER, H. The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science, 2001.

[102] JAIN, A. K., MAO, J., AND MOHIUDDIN, K. M. Artificial neural networks: A tutorial. *Computer 29* (March 1996), 31–44.

[103] JANG, J.-S. Anfis: adaptive-network-based fuzzy inference system. *Systems, Man and Cybernetics, IEEE Transactions on 23*, 3 (may/jun 1993), 665 –685.

[104] JORDAN, M. I. Attractor dynamics and parallelism in a connectionist sequential machine. 112–127.

[105] JR., M. M. G., AND ARAJO, A. F. A population dynamics model to describe gene frequencies in evolutionary algorithms. *Applied Soft Computing* (2012), –.

[106] K. TANG, XIAODONG LI, P. N. S. Z. Y., AND WEISE, T. Benchmark functions for the CEC'2010 special session and competition on large scale global optimization. Tech. rep., Nature Inspired Computation and Applications Laboratory, USTC, China, Tech. Rep., 2009.

[107] KALINLI, A., AND KARABOGA, D. Training recurrent neural networks by using parallel tabu search algorithm based on crossover operation. *Engineering Applications of Artificial Intelligence 17*, 5 (2004), 529 – 542.

[108] KAWAUCHI, S., SUGIHARA, H., AND SASAKI, H. Development of very-short-term load forecasting based on chaos theory. *Electrical Engineering in Japan 148*, 2 (2004).

[109] KAZARLIS, S. A., PAPADAKIS, S. E., THEOCHARIS, I. B., AND PETRIDIS, V. Microgenetic algorithms as generalized hill-climbing operators for ga optimization. *IEEE Trans. Evolutionary Computation 5*, 3 (2001), 204–217.

[110] KENNEL, M. B., AND ISABELLE, S. Method to distinguish possible chaos from colored noise and to determine embedding parameters. *Phys. Rev. A 46*, 6 (1992), 3111–3118.

[111] KIM, D., AND KIM, C. Forecasting time series with genetic fuzzy predictor ensemble. *Fuzzy Systems, IEEE Transactions on 5*, 4 (nov 1997), 523 –535.

[112] KINNEBROCK, W. Accelerating the standard backpropagation algorithm using genetic approach. *Neurocomput. 6*, 5-6 (1994), 583–588.

[113] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Optimization by simulated annealing. *Science 220*, 4598 (May 1983), 671–680.

[114] KOK, S., AND SANDROCK, C. Locating and characterizing the stationary points of the extended rosenbrock function. *Evol. Comput. 17* (September 2009), 437–453.

[115] KOLMAN, E., AND MARGALIOT, M. Extracting symbolic knowledge from recurrent neural networks–a fuzzy logic approach. *Fuzzy Sets and Systems 160*, 2 (2009), 145 – 161.

[116] KORDK, P., KOUTNK, J., DRCHAL, J., KOVRK, O., CEPEK, M., AND SNOREK, M. Meta-learning approach to neural network optimization. *Neural Networks 23*, 4 (2010), 568 – 582. The 18th International Conference on Artificial Neural Networks, ICANN 2008.

[117] KOSKELA, T., LEHTOKANGAS, M., SAARINEN, J., AND KASKI, K. Time series prediction with multilayer perceptron, fir and elman neural networks. In *In Proceedings of the World Congress on Neural Networks* (1996), pp. 491–496.

[118] KRASNOGOR, N., AND GUSTAFSON, S. Toward truly "memetic" memetic algorithms: discussion and proofs of concept. In *Advances in Nature-Inspired Computation: The PPSN VII Workshops* (2002).

[119] KRASNOGOR, N., AND SMITH, J. A memetic algorithm with self-adaptive local search: Tsp as a case study. In *Proceedings of the international conference on genetic and evolutionary computation* (2000), Morgan Kaufmann, pp. 987–994.

[120] KREMER, S. On the computational power of elman-style recurrent networks. *Neural Networks, IEEE Transactions on 6*, 4 (1995), 1000 – 1004.

[121] KU, K. W. C., MAK, M.-W., AND SIU, W.-C. A study of the lamarckian evolution of recurrent neural networks. *IEEE Trans. Evolutionary Computation 4*, 1 (2000), 31–42.

[122] LAWRENCE, S., AND GILES, C. Overfitting and neural networks: conjugate gradient and backpropagation. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on* (2000), vol. 1, pp. 114 –119 vol.1.

[123] LI, M., TIAN, J., AND CHEN, F. Improving multiclass pattern recognition with a co-evolutionary rbfnn. *Pattern Recogn. Lett. 29*, 4 (2008), 392–406.

[124] LI, X., AND YAO, X. Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation* (Piscataway, NJ, USA, 2009), CEC'09, IEEE Press, pp. 1546–1553.

[125] LIN, C.-J., CHEN, C.-H., AND LIN, C.-T. A hybrid of cooperative particle swarm optimization and cultural algorithm for neural fuzzy networks and its prediction applications. *Trans. Sys. Man Cyber Part C 39* (January 2009), 55–68.

[126] LIN, T., HORNE, B., TINO, P., AND GILES, C. Learning long-term dependencies in narx recurrent neural networks. *IEEE transactions on neural networks 7*, 6 (1996), 1329–1338.

[127] LIU, Y., YAO, X., ZHAO, Q., AND HIGUCHI, T. Scaling up fast evolutionary programming with cooperative coevolution. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on* (2001).

[128] LORENZ, E. Deterministic non-periodic flows. *Journal of Atmospheric Science 20* (1963), 267 – 285.

[129] LORENZ, E. *The Essence of Chaos*. University of Washington Press, 1993.

[130] LOZANO, M., AND GARCA-MARTNEZ, C. Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report. *Computers and Operations Research In Press*.

[131] LOZANO, M., HERRERA, F., KRASNOGOR, N., AND MOLINA, D. Real-coded memetic algorithms with crossover hill-climbing. *Evol. Comput. 12*, 3 (2004), 273–302.

[132] LOZANO, M., HERRERA, F., KRASNOGOR, N., AND MOLINA, D. Real-coded memetic algorithms with crossover hill-climbing. *Evolutionary Computation 12*, 3 (2004), 273–302.

[133] LOZANO, M., HERRERA, F., KRASNOGOR, N., AND MOLINA, D. Real-coded memetic algorithms with crossover hill-climbing. *Evol. Comput. 12* (September 2004), 273–302.

[134] LUKOSEVICIUS, M., AND JAEGER, H. Reservoir computing approaches to recurrent neural network training. *Computer Science Review 3*, 3 (2009), 127 – 149.

[135] LUKOSEVICIUTE, K., AND RAGULSKIS, M. Evolutionary algorithms for the selection of time lags for time series forecasting by fuzzy inference systems. *Neurocomput. 73* (2010), 2077–2088.

[136] MA, Q.-L., ZHENG, Q.-L., PENG, H., ZHONG, T.-W., AND XU, L.-Q. Chaotic time series prediction based on evolving recurrent neural networks. In *Machine Learning and Cybernetics, 2007 International Conference on* (aug. 2007), vol. 6, pp. 3496 –3500.

[137] MAASS, W., NATSCHLGER, T., AND MARKRAM, H. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation 14*, 11 (2002), 2531–2560.

[138] MACKEY, M., AND GLASS, L. Oscillation and chaos in physiological control systems. *Science 197*, 4300 (1977), 287–289.

[139] MANOLIOS, P., AND FANELLI, R. First-order recurrent neural networks and deterministic finite state automata. *Neural Comput. 6*, 6 (1994), 1155–1173.

[140] MCCULLOCH, W., AND PITTS, W. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5* (1943), 127–147.

[141] MENEZES, JR., J. M. P., AND BARRETO, G. A. Long-term time series prediction with the narx network: An empirical evaluation. *Neurocomput. 71* (2008), 3335–3343.

[142] MICHALEWICZ, Z. *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, London, UK, 1996.

[143] MICHALEWICZ, Z. *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, London, UK, 1996.

[144] MIRIKITANI, D., AND NIKOLAEV, N. Recursive bayesian recurrent neural networks for time-series modeling. *Neural Networks, IEEE Transactions on 21*, 2 (feb. 2010), 262 –274.

[145] MITCHELL, T. M. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.

[146] MOLINA, D., LOZANO, M., GARCÍA-MARTÍNEZ, C., AND HERRERA, F. Memetic algorithms for continuous optimisation based on local search chains. *Evol. Comput. 18* (March 2010), 27–63.

[147] MOLINA, D., LOZANO, M., GARCA-MARTNEZ, C., AND HERRERA, F. Memetic algorithms for continuous optimisation based on local search chains. *Evol. Comput. 18*, 1 (2010), 27–63.

[148] MORIARTY, D. E., AND MIIKKULAINEN, R. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation 5*, 4 (1997), 373–399.

[149] MOSCATO, P. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Tech. rep., 1989.

[150] MOSCATO, P. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report 826, Caltech Concurrent Computation Program, 1989.

[151] MOSCATO, P. A gentle introduction to memetic algorithms. In *Handbook of Metaheuristics* (2003), Kluwer Academic Publishers, pp. 105–144.

[152] MÜHLENBEIN, H., AND SCHLIERKAMP-VOOSEN, D. Predictive models for the breeder genetic algorithm i. continuous parameter optimization. *Evol. Comput. 1*, 1 (1993), 25–49.

[153] MUTOH, A., KATO, S., AND ITOH, I. Efficient real-coded genetic algorithms with flexible-step crossover. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on* (2005), pp. 1470 – 1476 Vol. 2.

[154] NGUYEN, Q. H., ONG, Y.-S., AND LIM, M. H. A probabilistic memetic framework. *Evolutionary Computation, IEEE Transactions on 13*, 3 (june 2009), 604 –623.

[155] NOMAN, N., AND IBA, H. Accelerating differential evolution using an adaptive local search. *Evolutionary Computation, IEEE Transactions on 12*, 1 (2008), 107 –125.

[156] OMIDVAR, M., LI, X., AND YAO, X. Cooperative co-evolution for large scale optimization through more frequent random grouping. In *Evolutionary Computation (CEC), 2010 IEEE Congress on* (2010), pp. 1754–1761.

[157] OMIDVAR, M., LI, X., AND YAO, X. Cooperative co-evolution with delta grouping for large scale non-separable function optimization. In *Evolutionary Computation (CEC), 2010 IEEE Congress on* (2010), pp. 1762 –1779.

[158] OMLIN, C. W., AND GILES, C. L. Constructing deterministic finite-state automata in recurrent neural networks. *J. ACM 43*, 6 (1996), 937–972.

[159] OMLIN, C. W., AND GILES, C. L. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks 9*, 1 (1996), 41 – 52.

[160] OMLIN, C. W., THORNBER, K. K., AND GILES, C. L. Fuzzy finite state automata can be deterministically encoded into recurrent neural networks. *IEEE Trans. Fuzzy Syst. 6* (1998), 76–89.

[161] ONG, Y. S., AND KEANE, A. Meta-lamarckian learning in memetic algorithms. *Evolutionary Computation, IEEE Transactions on 8*, 2 (april 2004), 99 – 110.

[162] ONG, Y. S., LIM, M. H., AND CHEN, X. S. Memetic computation - past, present and future. *IEEE Computational Intelligence Magazine* (2010), In Press.

[163] ONG, Y. S., LIM, M. H., ZHU, N., AND WONG, K. W. Classification of adaptive memetic algorithms: A comparative study. *IEEE Transactions on Systems Man and Cybernetics – Part B 36*, 1 (2006), 141–152.

[164] ONG, Y.-S., LIM, M.-H., ZHU, N., AND WONG, K.-W. Classification of adaptive memetic algorithms: a comparative study. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 36*, 1 (feb. 2006), 141 –152.

[165] ORTIZ-BOYER, D., HERVS-MARTNEZ, C., AND GARCA-PEDRAJAS, N. CIXL2 - a crossover operator for evolutionary algorithms based on population features. *Journal of Artificial Intelligence Research 24* (2005), 2005.

[166] PALIWAL, M., AND KUMAR, U. A. Neural networks and statistical techniques: A review of applications. *Expert Systems with Applications 36*, 1 (2009), 2 – 17.

[167] PHAM, D. T., AND KARABOGA, D. Training elman and jordan networks for system identification using genetic algorithms. *Artificial Intelligence in Engineering 13*, 2 (1999), 107 – 117.

[168] POTTER, M. A., AND DE JONG, K. A. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evol. Comput. 8*, 1 (2000), 1–29.

[169] POTTER, M. A., AND JONG, K. A. D. A cooperative coevolutionary approach to function optimization. In *PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature* (London, UK, 1994), Springer-Verlag, pp. 249–257.

[170] POTTS, J. C., GIDDENS, T. D., AND YADAV, S. B. The development and evaluation of an improved genetic algorithm based on migration and artificial selection. *Systems, Man and Cybernetics, IEEE Transactions on 24*, 1 (1994), 73–86.

[171] POŠIK, P. Bbob-benchmarking the generalized generation gap model with parent centric crossover. In *Proceedings of the 11th An-*

nual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers (New York, NY, USA, 2009), GECCO '09, ACM, pp. 2321–2328.

[172] RACHENBERG, I. *(In German) Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Reprinted by Fromman-Holzboog, 1973.

[173] RADCLIFFE, N. J. Equivalence class analysis of genetic algorithms. *Complex Systems 5* (1991), 183–205.

[174] RAIDL, G. A unified view on hybrid metaheuristics. In *Hybrid Metaheuristics*, F. Almeida, M. Blesa Aguilera, C. Blum, J. Moreno Vega, M. Prez Prez, A. Roli, and M. Sampels, Eds., vol. 4030 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 1–12.

[175] RAY, T., AND YAO, X. A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation* (Piscataway, NJ, USA, 2009), CEC'09, IEEE Press, pp. 983–989.

[176] RAY, T. V., N. KOK, S. W., AND KIAN, P. C. Study on the behaviour and implementation of parent centric crossover within the generalized generation gap model. In *IEEE Congress on Evolutionary Computation* (2004), IEEE, pp. 1996–2003.

[177] ROBINSON, A. J., AND FALLSIDE, F. The utility driven dynamic error propagation network. Tech. rep., Cambridge University Engineering Department, Technical Report CUED/F-INFENG/TR.1, 1987.

[178] ROBINSON, T. An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks 5* (1994), 298–305.

[179] ROJAS, I., POMARES, H., BERNIER, J. L., ORTEGA, J., PINO, B., PELAYO, F. J., AND PRIETO, A. Time series analysis using normalized pg-rbf network with regression weights. *Neurocomputing 42*, 1-4 (2002), 267 – 285.

[180] ROJAS, I., VALENZUELA, O., ROJAS, F., GUILLEN, A., HERRERA, L., POMARES, H., MARQUEZ, L., AND PASADAS, M. Soft-computing techniques and arma model for time series prediction. *Neurocomputing 71*, 4-6 (2008), 519 – 537.

[181] RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J. Learning internal representations by error propagation. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1* (Cambridge, MA, USA, 1986), MIT Press, pp. 318–362.

[182] S., S. Solar cycle forecasting: A nonlinear dynamics approach. *Astronomy and Astrophysics 377* (2001), 312–320.

[183] SALOMON, R. Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. a survey of some theoretical and practical aspects of genetic algorithms. *Biosystems 39*, 3 (1996), 263 – 278.

[184] SCHAEFER, A., AND ZIMMERMANN, H. Recurrent neural networks are universal approximators. *Int. J. Neural Systems 17*, 4 (2007), 253–263.

[185] SCHAEFER, A. M., UDLUFT, S., AND ZIMMERMANN, H.-G. Learning long-term dependencies with recurrent neural networks. *Neurocomput. 71* (August 2008), 2481–2488.

[186] SCHLIERKAMP-VOOSEN, D. Strategy adaptation by competition. In *Proceedings of the Second European Congress on Intelligent Techniques and Soft Computing* (1994), pp. 1270–1274.

[187] SCHMIDHUBER, J., WIERSTRA, D., GAGLIOLO, M., AND GOMEZ, F. Training recurrent networks by evolino. *Neural Comput. 19*, 3 (2007), 757–779.

[188] SERONT, G., AND BERSINI, H. A new ga-local search hybrid for continuous optimization based on multi-level single linkage clustering. In *GECCO* (2000), pp. 90–95.

[189] SEXTON, R. S., AND DORSEY, R. E. Reliable classification using neural networks: a genetic algorithm and backpropagation comparison. *Decision Support Systems 30*, 1 (2000), 11–22.

[190] SEXTON, R. S., AND GUPTA, J. N. D. Comparative evaluation of genetic algorithm and backpropagation for training neural networks. *Information Sciences 129*, 1-4 (2000), 45 – 59.

[191] SEYAB, R. A., AND CAO, Y. Nonlinear system identification for predictive control using continuous time recurrent neural networks and automatic differentiation. *Journal of Process Control 18*, 6 (2008), 568 – 581.

[192] SHANG, Y.-W., AND QIU, Y.-H. A note on the extended rosenbrock function. *Evol. Comput. 14* (March 2006), 119–126.

[193] SHI, Y.-J., TENG, H.-F., AND LI, Z.-Q. Cooperative co-evolutionary differential evolution for function optimization. In *Advances in Natural Computation*, L. Wang, K. Chen, and Y. S. Ong, Eds., vol. 3611 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, pp. 1080–1088.

[194] SHIMODAIRA, H. A diversity control oriented genetic algorithm (DCGA): Development and experimental results. In *Proceedings of the Genetic and Evolutionary Computation Conference* (1999), W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., Morgan Kaufmann, pp. 603–611.

[195] SIDC. World data center for the sunspot index, montly smoothed sunspot data.

[196] SIEGELMANN, H., HORNE, B., AND GILES, C. Computational capabilities of recurrent NARX neural networks. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 27*, 2 (1997), 208 –215.

[197] SIMA, J. Back-propagation is not efficient. *Neural Networks 9*, 6 (1996), 1017 – 1023.

[198] SINHA, A., TIWARI, S., AND K., D. A population based steady state procedure for real parameter optimization. Tech. rep., KANGAL Technical Report no. 2005004, 2005.

[199] SMITH, J. Modelling gas with self adapting mutation rates. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2001).

[200] SMITH, J. Coevolving memetic algorithms: A review and progress report. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 37*, 1 (feb. 2007), 6 –17.

[201] SOAK, S.-M., LEE, S.-W., MAHALIK, N., AND AHN, B.-H. A new memetic algorithm using particle swarm optimization and genetic algorithm. In *Knowledge-Based Intelligent Information and Engineering Systems*, B. Gabrys, R. Howlett, and L. Jain, Eds., vol. 4251 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 122–129.

[202] STANLEY, K. O. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines 8*, 2 (2007), 131–162.

[203] STANLEY, K. O., D'AMBROSIO, D. B., AND GAUCI, J. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life 15*, 2 (2009), 185–212. PMID: 19199382.

[204] STANLEY, K. O., AND MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evolutionary Computation 10*, 2 (2002), 99–127.

[205] STEPHEN, H. K. *In the Wake of Chaos: Unpredictable Order in Dynamical Systems*. University of Chicago Press, 1993.

[206] STORN, R., AND PRICE, K. Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization 11* (1997), 341–359.

[207] TAKENS, F. Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence, Warwick 1980*, Lecture Notes in Mathematics. 1981, pp. 366–381.

[208] TALBI, E.-G., AND BACHELET, V. Cosearch: A parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms 5*, 1 (2006), 5–22.

[209] TEO, J., HIJAZI, H. A., OMAR, Z. A., MOHAMAD, N. R., AND HAMID, Y. Harnessing mutational diversity at multiple levels for improving optimization accuracy in g3-pcx. In *IEEE Congress on Evolutionary Computation* (2007), IEEE, pp. 4502–4507.

[210] TEO, K. K., WANG, L., AND LIN, Z. Wavelet packet multi-layer perceptron for chaotic time series prediction: Effects of weight initialization. In *Proceedings of the International Conference on Computational Science-Part II* (2001), ICCS '01, pp. 310–317.

[211] THOMSEN, R., RICKERS, P., AND KRINK, T. A religion-based spatial model for evolutionary algorithms. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature* (London, UK, 2000), PPSN VI, Springer-Verlag, pp. 817–826.

[212] TOMASSINI, M. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation 6* (2002), 443–462.

[213] TOMITA, M. Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Cognitive Science Conference* (Ann Arbor, MI., 1982), pp. 105–108.

[214] TSOI, A. C., AND BACK, A. Locally recurrent globally feedforward networks: a critical review of architectures. *Neural Networks, IEEE Transactions on 5*, 2 (1994).

[215] TSOI, A. C., AND BACK, A. Discrete time recurrent neural network architectures: A unifying review. *Neurocomputing 15*, 3-4 (1997), 183 – 223. Recurrent Neural Networks.

[216] TSOULOS, I. G. Modifications of real code genetic algorithm for global optimization. *Applied Mathematics and Computation 203*, 2 (2008), 598 – 607.

[217] TSUTSUI, S., GHOSH, A., CORNE, D., AND FUJIMOTO, Y. A Real Coded Genetic Algorithm with an Explorer and an Exploiter Populations. In *Proceedings of the 7th ICGA* (1997), T. Bäck, Ed., Morgan Kaufmann, pp. 238–245.

[218] TSUTSUI, S., YAMAMURA, M., AND HIGUCHI, T. Multi-parent recombination with simplex crossover in real coded genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference* (13-17 July 1999), vol. 1, pp. 657–664.

[219] URSEM, R. Multinational evolutionary algorithms. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on* (1999), vol. 3, pp. 1633–1640.

[220] URSEM, R. K. Diversity-guided evolutionary algorithms. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature* (London, UK, UK, 2002), PPSN VII, Springer-Verlag, pp. 462–474.

[221] VAN DEN BERGH, F., AND ENGELBRECHT, A. A cooperative approach to particle swarm optimization. *Evolutionary Computation, IEEE Transactions on 8*, 3 (2004), 225 – 239.

[222] VAN KEMENADE, C. Cluster evolution strategies. enhancing the sampling density function using representatives. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on* (May 1996), pp. 637 –642.

[223] ČERNÝ, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications 45*, 1 (January 1985), 41–51.

[224] WATROUS, R. L., AND KUHN, G. M. Induction of finite-state languages using second-order recurrent networks. *Neural Comput. 4*, 3 (1992), 406–414.

[225] WATTS, D. J. *Small worlds: The dynamics of networks between order and randomness*. Princeton University Press, Princeton, NJ, 1999.

[226] WERBOS, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE 78*, 10 (1990), 1550–1560.

[227] WHITLEY, D. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms* (1989), Morgan Kaufmann, pp. 116–121.

[228] WHITLEY, D., DOMINIC, S., DAS, R., AND ANDERSON, C. W. Genetic reinforcement learning for neurocontrol problems. *Machine Learning 13*, 2 (1993), 259–284.

[229] WILLIAMS, R., AND ZIPSER, D. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science 1*, 1 (1989), 87–111.

[230] WILLIAMS, R. J., AND ZIPSER, D. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput. 1*, 2 (1989), 270–280.

[231] WONG, M. L., LEE, S. Y., AND LEUNG, K. S. Data mining of bayesian networks using cooperative coevolution. *Decis. Support Syst. 38*, 3 (2004), 451–472.

[232] WRIGHT, A. H. Genetic algorithms for real parameter optimization. In *Foundations of Genetic Algorithms* (1991), Morgan Kaufmann, pp. 205–218.

[233] YANG, Z., TANG, K., AND YAO, X. Large scale evolutionary optimization using cooperative coevolution. *Inf. Sci. 178*, 15 (2008), 2985–2999.

[234] YANG, Z., TANG, K., AND YAO, X. Multilevel cooperative coevolution for large scale optimization. In *IEEE Congress on Evolutionary Computation* (2008), pp. 1663–1670.

[235] YAO, X. Evolving artificial neural networks. *Proceedings of the IEEE 87*, 9 (1999), 1423–1448.

Appendix A

Problem	Features	Classes	Min. Train (%)	No. Instances
4-Bit	4	1	–	16
Wine	13	3	95	178
Iris	4	3	95	150
Heart-Disease	13	1	88	303
Breast-Cancer	9	1	95	699
Zoo	16	7	95	102

Table A.1: The dataset information and neural network configuration for the given problems. Note that the Breast-Cancer dataset contains 16 missing values and is class imbalanced (65.5 % Benign and 34.5 % Malignant). In the 4-Bit problem, the network is trained until the mean-squared-error goes below 1E-3. 70 % of the data is used for training while the remaining 30 % is used for testing in all the other problems. The classification targets given by "Min. Train (%)" were determined in trial experiments. The number of input and output neurons in the network depends on the number of features and classes in the dataset.