Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Encoding subcomponents in cooperative co-evolutionary recurrent neural networks

Rohitash Chandra [a,b,*], Marcus Frean [a], Mengjie Zhang [a], Christian W. Omlin [c]

[a] School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand
[b] Department of Computing Science and Information Systems, Fiji National University, Suva, Fiji
[c] Department of Computer Engineering, Northern Cyprus Campus, Middle East Technical University, Cyprus

ARTICLE INFO

Article history:
Received 25 November 2009
Received in revised form
10 November 2010
Accepted 3 May 2011
Communicated by A. Abraham

Keywords:
Cooperative coevolution
Neuro-evolution
Recurrent neural networks
Grammatical inference
Genetic algorithms

ABSTRACT

Cooperative coevolution employs evolutionary algorithms to solve a high-dimensional search problem by decomposing it into low-dimensional subcomponents. Efficient problem decomposition methods or encoding schemes group interacting variables into separate subcomponents in order to solve them separately where possible. It is important to find out which encoding schemes efficiently group subcomponents and the nature of the neural network training problem in terms of the degree of non-separability. This paper introduces a novel encoding scheme in cooperative coevolution for training recurrent neural networks. The method is tested on grammatical inference problems. The results show that the proposed encoding scheme achieves better performance when compared to a previous encoding scheme.

1. Introduction

The ability of recurrent neural networks (RNNs) to model any open dynamical system has been praised [1–4]. They are difficult to train using gradient descent based methods which are unable to guarantee an acceptable solution in difficult problems and those involving long-term dependencies [5,6]. Neuro-evolution has been used for training neural networks as it does not rely on gradient information and can be easily deployed in any neural network optimization problem without being constrained to a particular architecture [7].

Cooperative coevolution (CC) is a biologically inspired evolutionary computation framework which divides a large problem into smaller subcomponents and solves them [8]. The subcomponents are represented using subpopulations that evolve independently. In the field of genetics, a genotype is an organisms heritable information and a phenotype is its observed behavior. In neuro-evolution, the genotype–phenotype mapping or encoding scheme is the way the genes are mapped into the neural network, which accepts the network's behavior [7]. Cooperative coevolution has been effective for training recurrent neural networks using different encoding schemes, including enforced subpopulations [9,10] and cooperatively co-evolved synapses [11]. Cooperative coevolution has also shown promising performance in training feedforward networks [8,12–14].

In the original cooperative co-evolutionary framework, the problem is decomposed by having a separate subcomponent for each variable [8]. It was later found that the strategy was only effective for problems which are *separable* [15], in the sense that there is no interdependency between the decision variables. Cooperative coevolution is a natural algorithm to use for separable problems as there is no interaction among the subcomponents during evolution [16]. The type of problem decomposition influences the performance of cooperative coevolution. A higher *degree of non-separability* indicates that more interacting variables exist than the problem with lower degree of non-separability.

The CC framework has been used effectively in training recurrent neural networks in order to preserve information associated with the recurrent or state neurons. In standard neuro-evolution, there is a tendency to lose internal state information as the crossover operator makes changes to weights of the entire network.

Pioneering work was done in the deployment CC as *enforced subpopulations* (ESP) for training recurrent neural networks in solving the double pole balancing problem [9,10]. A sophisticated version of ESP known as *Evolino* has been used to evolve the LSTM network. It was shown that the framework outperformed gradient based LSTM and learned tasks that were unlearn-able by Echo

[*] Corresponding author at: School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand.
E-mail address: rohitash_c@yahoo.com (R. Chandra).

0925-2312/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2011.05.003

State Networks [17]. Recently, the same framework with a different encoding scheme has been used for training RNNs for the double pole balancing problem without velocity information. The approach was called cooperatively co-evolved synapse neuro-evolution (CoSyNE) and has shown better performance than ESP and other neuro-evolution methods [11].

This work introduces an encoding scheme which is motivated from the architectural properties of a neuron and is specifically designed for training recurrent neural networks. The encoding scheme is called neuron-based subpopulation (NSP). The performance of NSP is compared with CoSyNE for grammatical inference problems. We investigate on the optimal depth of search required for the subcomponents in the respective encoding schemes. The depth of search also helps determine the degree of non-separability exhibited by the different encoding schemes. Note that the goal is to group interacting variables (weights) into separate subcomponents so that there is less interaction among subcomponents. In this way, the problem is most efficiently decomposed in order to take full advantage of cooperative coevolution which naturally appeals to separable problems.

We also observe the behavior of the respective methods for different numbers of hidden neurons which reflects on scalability. Our goal is to achieve faster convergence and better success rates when compared to CoSyNE. In our previous work, NSP has been implemented for feedforward neural networks [14]. The results show that NSP outperformed CoSyNE for pattern recognition problems. We also discuss the performance of NSP for recurrent neural networks in comparison to feedforward networks in pattern recognition problems and comment on the degree of non-separability and its relationship to the encoding scheme.

In order to demonstrate the effectiveness of NSP, specific grammatical inference problems are taken from [18] and the Tomita grammar [19,20]. The Elman style first-order recurrent neural network [21,22] is the designated network architecture in all experiments.

This paper applies the existing encoding schemes from literature, in particular, CoSyNE for training recurrent neural networks on grammatical inference problems. The main contribution of this paper is that it proposes a new encoding scheme for training recurrent neural networks and examines the degree of non-separability exhibited by the given encoding schemes.

The rest of the paper is organized as follows. Background on recurrent neural networks, grammatical inference and cooperative coevolution framework is presented in Section 2. Section 3 presents the encoding scheme in the proposed neuron-based subpopulation for training recurrent neural networks. Section 4 presents the results and discussion, and Section 5 concludes the paper with a discussion on future work.

2. Background

2.1. Recurrent neural networks

Recurrent neural networks are dynamical systems whose next state and output depend on the present network state and input; the networks are composed of an *input layer*, a *context layer* which provides state information, a *hidden layer* and an *output layer*. Each layer contains one or more neurons which propagate information from one layer to the another by computing a non-linear function of their weighted sum of inputs.

First-order recurrent neural networks employ context units to store the output of the state neurons from computation of previous time steps. The Elman architecture [21] employs the context layer which makes a copy of the hidden layer outputs in the previous time steps. The equation of the dynamics of the change of hidden state neuron activations in the context layer is given in as follows:

$$S_i(t)=g\left(\sum_{k=1}^{K} V_{ik}S_k(t-1)+\sum_{j=1}^{J} W_{ij}I_j(t-1)\right) \tag{1}$$

where $S_k\,(t)$ and $I_j(t)$ represent the output of the context state and input neurons, respectively, and V_{ik} and W_{ij} represent their corresponding weights. $g(.)$ is a sigmoid squashing function.

2.2. Grammatical inference

Grammatical inference problems have been used to study training algorithms for knowledge representation in recurrent neural networks. It has been demonstrated through knowledge extraction that RNNs can represent finite-state automata [23–26]. There is no feature extraction necessary in order for recurrent neural networks to learn these languages. Thus, grammatical inference is used as an appropriate test bed for the investigation of the performance and other issues of learning algorithms for recurrent neural networks. A formal definition on deterministic finite-state automata (DFA) and fuzzy finite-state automata (FFA) is given as follows.

Definition 1. A deterministic finite-state automata (DFA) 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 [27].

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. Fig. 1 shows the DFAs selected from the Tomita grammar which will be used for training the recurrent network in this study.

Definition 2. A fuzzy finite-state automaton M is a six-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 [27].

A restricted type of fuzzy automata is considered whose initial state is not fuzzy, and in which ω is a function from F to Z, where F is a non-fuzzy set of states, called finite states. Any fuzzy automaton as described in Definition 1 is equivalent to a restricted fuzzy automaton. The transformation of a fuzzy

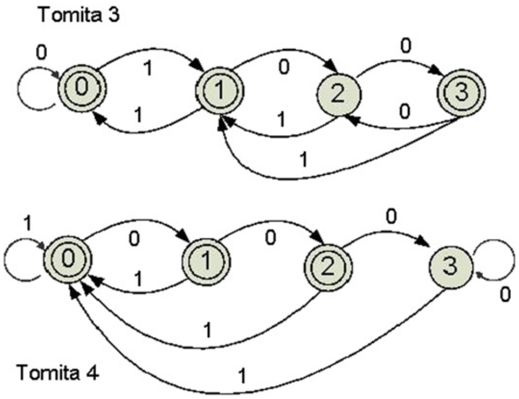

Fig. 1. Deterministic Finite-State Automata from the Tomita grammar: Double circles in the figure show accepting states while rejecting states are shown by single circles. State 1 is the automata's start state.

Please cite this article as: R. Chandra, et al., Encoding subcomponents in cooperative co-evolutionary recurrent neural networks, Neurocomputing (2011), doi:10.1016/j.neucom.2011.05.003

automaton to its corresponding deterministic acceptor is discussed in [1]. Fig. 2 shows an example of a FFA with its corresponding deterministic acceptor which is used for training recurrent neural networks. This FFA has been used in [18] to show that RNNs can be trained by evolutionary algorithms.

The Tomita grammars [19] have been used as a benchmark problem in order to evaluate RNN training algorithms and architectures [20]. The Tomita grammars consists of seven regular languages. An illustration of two selected grammars is shown in Fig. 1.

2.3. Cooperative coevolution

In the evolutionary process of nature, different species compete in order to survive with given resources. The individuals of a particular group of species mate among themselves. However, mating between different species does not give rise to viable offspring. The cooperative coevolution framework is nature inspired where species are represented as subcomponents, implemented as subpopulations.

The subpopulations in the cooperative coevolution framework are evolved in isolation and the cooperation only takes place for fitness evaluation for the respective individuals in each subpopulation. The subpopulations are evolved in a round-robin fashion for a given number of generations known as the *depth of search*. The depth of search has to be predetermined according to the nature of the problem. The depth of search can reflect whether the encoding schemes have been able to group the interacting variables into separate subcomponents. If the interacting variables have been grouped efficiently, then a deep greedy search for the subpopulation is possible, implying that the problem has been efficiently broken down into subcomponents which have fewer interactions among themselves.

2.4. The issue of separability in cooperative coevolution

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. A function of n variables is separable if it can be written as a sum of n functions with just one variable [28]. Non-separable problems have interdependencies between decision variables as opposed to separable ones. Several methods have been proposed which group interacting and non-interacting variables for global optimization problems. In order to solve large-scale optimization problems which fall between total separable and fully non-separable, it is important to efficiently decompose the high-dimensional problem into several low-dimensional subcomponents where the subcomponents have least interaction among themselves.

It has been outlined that in real-world application problems, most problems fall between total separable and fully non-separable. Cooperative coevolution has been effective for separable problems [8] as it has the property to decompose the problem into sub-problems and solve them in isolation. Conversely, evolutionary algorithms without any decomposition strategy appeal to fully non-separable problems. Note that the goal is to solve non-separable problems without any prior knowledge about the interacting variables in the problem. Therefore, the evolutionary algorithm has to identify the interdependencies itself.

Fast evolutionary programming in the cooperative co-evolutionary framework (FEPCC) has been the first attempt to tackle large-scale function optimization of up to 1000 dimensions [15]. FEPCC used the original CCEA framework by Potter and Jong [8]. 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.

Yang et al. [29] have presented a cooperative coevolution algorithm that employs 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 heuristically. Yang et al. also presented a multi-level cooperative coevolution framework (MLCC) [30] which adapts the size of the subcomponents in DECC-G in order to group interacting and non-interacting variables. The framework begins with small sized subcomponents and adapts to bigger subcomponents from a predefined set. MLCC showed better performance than DECC-G for non-separable problems up to 1000 dimensions. Omidvar et al. [31] made amendments to the random grouping approach. They presented a *more frequent random grouping* approach which outperformed its counterpart in several non-separable problems up to 1000 dimensions.

2.5. Network encoding schemes for cooperative coevolution

An important feature of cooperative coevolution for neuro-evolution is that it allows the neural network to be decomposed, enabling the subcomponents to be isolated. Problem decomposition determines the size of a subcomponent and the way it is encoded. In the case of neural networks, we refer to problem decomposition as an encoding scheme.

There have been two major encoding schemes based on the CC framework for training recurrent neural networks. The first scheme proposes a neuron level encoding where each neuron in the hidden layer is used as a major reference point for each subpopulation in the CC framework. Therefore, the number of hidden neurons is equal to the number of subpopulations.

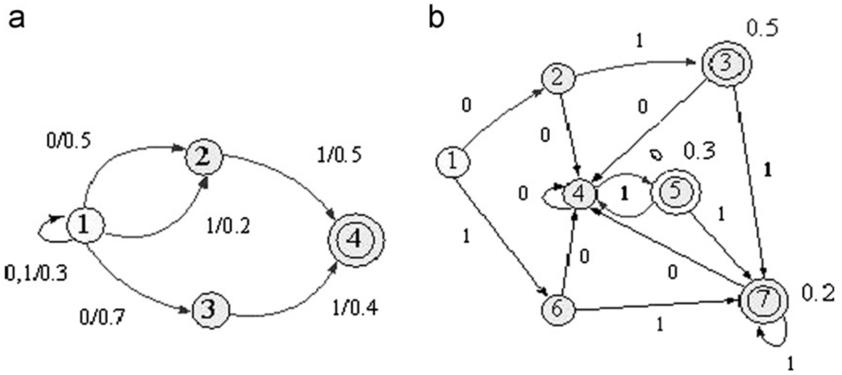

Fig. 2. The fuzzy finite-state automata (a) and its equivalent deterministic acceptor (b). The accepting states are labeled with a degree of membership. State 1 is the automata's start state; accepting states are drawn with double circles [18].

Please cite this article as: R. Chandra, et al., Encoding subcomponents in cooperative co-evolutionary recurrent neural networks, Neurocomputing (2011), doi:10.1016/j.neucom.2011.05.003

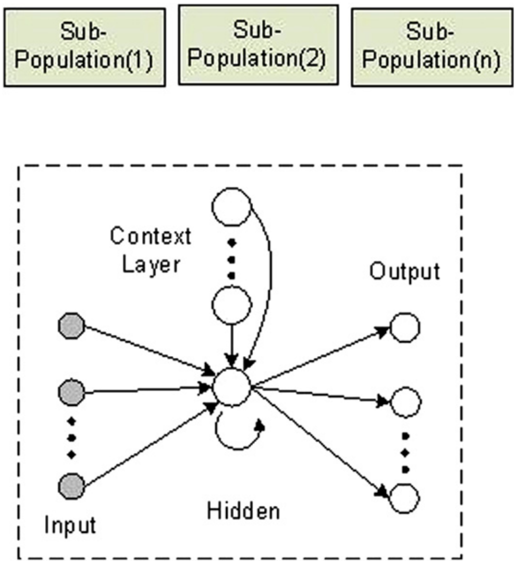

Fig. 3. The ESP encoding scheme taken from [10].

This encoding has been used in *enforced subpopulation* (ESP) [9,10] where a particular neuron h_i in the hidden layer encodes the following weight links in its subpopulation:

1. The weight 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; and
4. The bias associated with h_i.

In this encoding scheme, the sizes of all the subpopulations are the same for the entire framework. This encoding is shown in Fig. 3. Note that it is difficult to use this encoding scheme when more than one hidden layer is present.

The second encoding scheme has been presented in the cooperatively co-evolved synapse neuro-evolution (CoSyNE) algorithm. This encoding scheme decomposes the network into its lowest level, where each weight link (synapse) in the network is part of a single subpopulation. Therefore, the number of subpopulations depends on the number of weights and biases. Note that the CoSyNE demonstrated better performance than ESP on the double pole balancing problem [11].

3. Neuron-based subpopulation (NSP) for recurrent neural networks

The neuron-based subpopulation (NSP) is motivated by the architectural properties of a single neuron which computes the weighted sum of incoming weight links associated with it. Unlike ESP, NSP does not include the outgoing weight links in this computation. Each neuron in the hidden and output layer is a reference point for a subpopulation. Each hidden neuron also acts as a reference point for the recurrent (state or context) weight links connected to it. Therefore, each subpopulation for a RNN with a single hidden layer is composed of the following:

1. Hidden layer subpopulations: weight links from each neuron in the *hidden*(t) layer connected to all *input*(t) neurons and the bias of *hidden*(t), where t is time.
2. State (recurrent) neuron subpopulations: weight links from each neuron in the *hidden*(t) layer connected to all hidden neurons in previous time step *hidden*$(t-1)$.
3. Output layer subpopulations: weight links from each neuron in the *output*(t) layer connected to all *hidden*(t) neurons and the bias of *output*(t).

The general NSP CC framework for training RNN is given in Algorithm 1. Fig. 4 shows a detailed diagram of the NSP encoding scheme. Note that unlike ESP, NSP can be easily extended to a neural network with more than a single hidden layer.

Algorithm 1. The NSP CC framework for training RNN

```
Step 1: Decompose the problem into $k$ subcomponents
according to the number of Hidden, State, and Output neurons
Step 2: Encode each subcomponent in a subpopulation in the
following order:
i) Hidden layer subpopulations
ii) State (recurrent) neuron subpopulations
iii) Output layer subpopulations
Step 3: Initialize and cooperatively evaluate each
subpopulation
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 the new offspring to the subpopulation
    end for
   end for
end for
```

In Algorithm 1, the recurrent neural network is decomposed in k subcomponents, where k is equal to the number of hidden neurons, plus the number of context neurons, plus the number of output neurons. Each subpopulation contains all the weight links from the previous layer connecting to a particular neuron. A *cycle* is completed when all the subpopulations are evolved for a fixed number of generations. The algorithm halts if the termination condition is satisfied. The termination criteria can be a specified neural network error, classification performance on the training or validation data or when the maximum number of function evaluations has been reached.

A major concern in the NSP framework is the cooperative evaluation of each individual in every subpopulation. There are

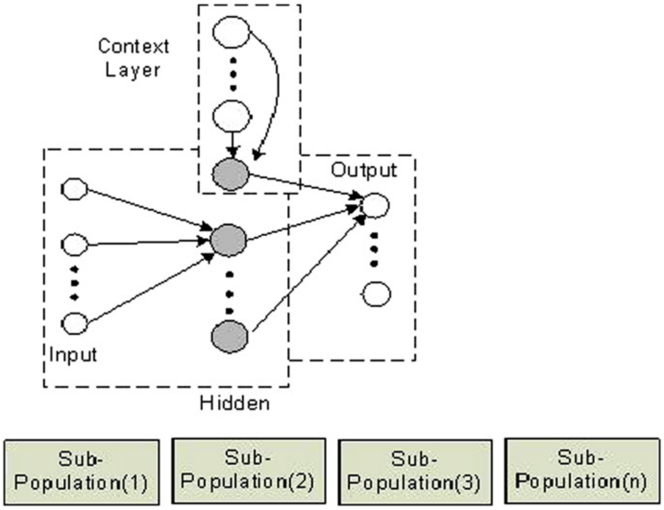

Fig. 4. The NSP encoding scheme. Each neuron in the hidden and output layer acts as a reference point to its subcomponents given as subpopulations. The subpopulation for the context weights is also shown. The same encoding scheme is used in the rest of the neurons in the hidden and output layer.

Please cite this article as: R. Chandra, et al., Encoding subcomponents in cooperative co-evolutionary recurrent neural networks, Neurocomputing (2011), doi:10.1016/j.neucom.2011.05.003

two main phases of evolution in the cooperative coevolution framework. The first is the *initialization phase* and second is the *evolution phase*.

Cooperative evaluation in the initialization phase is given in Step 3. In order to evaluate the *i*th individual of the *k*th subcomponent (chosen individual), arbitrary individuals from the rest of the subpopulations is selected and combined with the chosen individual and cooperatively evaluated. Arbitrary individuals are selected as in the initialization stage, the individuals from the rest of the subpopulations do not have a fitness. Once the fitness has been assigned to all the individuals of a particular subpopulation, then the best individual can be chosen. A similar approach is shown in [32]. In the evolution phase, cooperative evaluation shown in Step 3(ii) is done by combining or concatenating the chosen individual from a subpopulation *k* with the best individuals from the rest of the subpopulations. After the initialization phase, the best individuals from the rest of subpopulations can easily be found through ranking according to fitness. The concatenated individual is encoded into the recurrent neural network and the fitness is calculated. The fitness function is the sum-squared-error of the network. The fitness evaluation of individuals in each subpopulation is further shown in Fig. 5. The goal of the evolutionary process is to increase the fitness which tends to decrease the network error. In this way, the fitness of each subcomponent in the network is evaluated until the cycle is completed.

3.1. Performance evaluation

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. A run is considered successful if a desired solution is found before the maximum time is reached. The desired solution for neural network training is specified by a predefined minimum network error or minimum classification performance depending on the type of the problem.

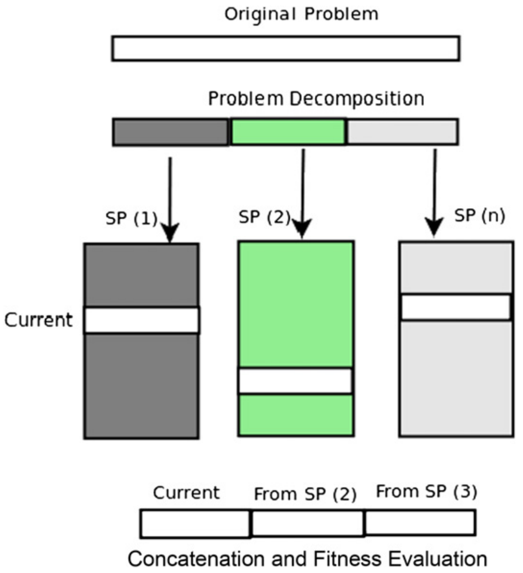

Fig. 5. The current individual whose fitness has to be evaluated is concatenated with arbitrary individuals from the rest of the subpopulations in the initialization phase. In the evolution phase, the best individuals from the rest of the subpopulations are chosen. The current individual is then concatenated with the chosen individuals. The fitness is then evaluated and assigned to the current individual.

4. Experimentation, results and analysis

This section presents an experimental study of NSP for training recurrent neural networks and compares it with CoSyNE. CoSyNE has shown better performance than ESP, therefore, we use it for comparison with NSP. In order to show a favorable comparison with NSP, CoSyNE will use the same method for subpopulation initialization and cooperative evaluation. The Elman recurrent network [21] with one hidden layer is used in all experiments.

The G3-PCX evolutionary algorithm [33] is employed in the respective CC frameworks with 100 individuals in all subpopulations. The G3-PCX algorithm employs the mating pool size of two offspring and two parents with the generation gap model for selection for NSP and CoSyNE. This set-up has been used in [33] for optimization problems and in [14] for training cooperative coevolutionary feedforward neural networks. The subpopulations are seeded with random real numbers in the range of $[-5, 5]$ in all the experiments.

Grammatical inference is used as a means to study the performance of the proposed NSP framework in recurrent neural networks. The FFA shown in Fig. 2(b) is used. The training dataset is generated by presenting strings of increasing lengths of 1–7 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 8–14 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 four fuzzy output states of the FFA.

Similarly, we generated the training and testing dataset form the Tomita language shown in Fig. 1. We used Tomita 1 to Tomita 4 (T1, T2, T3 and T4) for comparison [19,20]. In this case, the training and testing data is generated by presenting random strings of length 15–25 for each Tomita language. The training and testing dataset contains 250 (125 positive and 125 negative) string samples. Note that the string lengths considered here (15–25) cannot be trained using the classical backpropagation-through-time algorithm as outlined in [34].

In all problems, the RNN is trained until the mean-squared-error (MSE) reaches below 0.0005. The training is terminated if 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 a maximum of 10 000 function evaluations. These values were chosen in trial experiments.

We report the optimization time of the respective algorithms in terms of the number of function evaluations. The success rate is also used as a measure. The goal of each algorithm is to get convergence in the least optimization time with a high success rate.

4.1. Depth of search for NSP in recurrent networks

In the NSP framework for recurrent networks shown in Algorithm 1, each subpopulation 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 subpopulation which is considered as the *depth of search*. Note that all subpopulations are meant to evolve for the same number of *n* generations which must be fixed beforehand.

The FFA used in this experiment has seven states and in order to make the problem more difficult, only four neurons in the hidden layer of the RNN are used to represent seven states. In the T1 and T2 problem, two neurons in the hidden layer are used. In the T3 and T4 problem, three neurons in the hidden layer are used.

The results given in Figs. 6–10 report the optimization time with respect to the depth of search needed in the respective cooperative co-evolutionary framework (NSP and CoSyNE). These

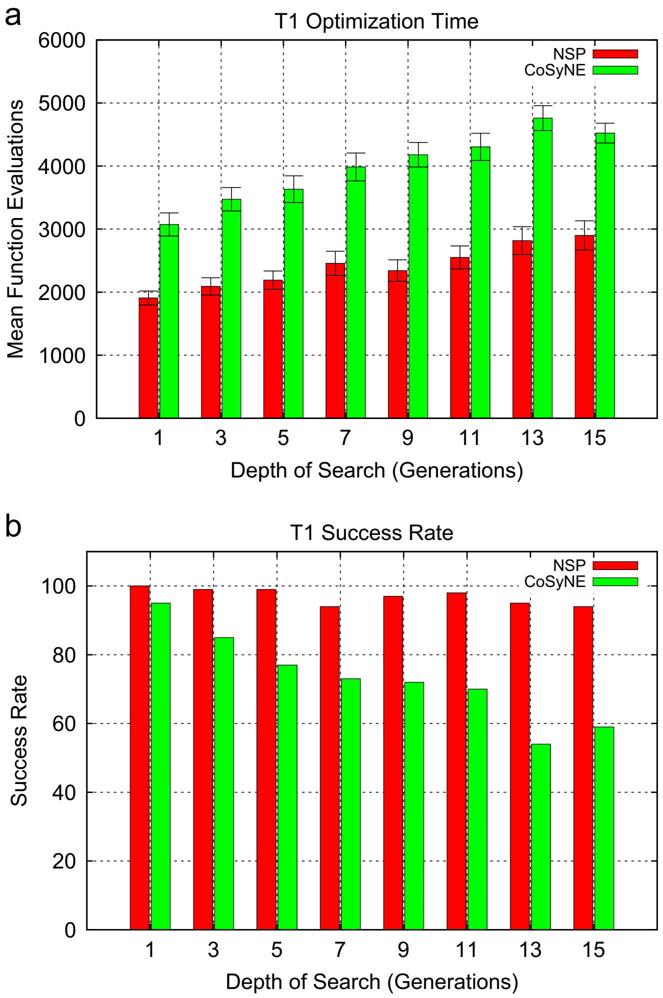

Fig. 6. The performance of NSP and CoSyNE on different depths of search in terms of the number of generations for the T1 problem. The optimization time in terms of the average number of function evaluations is shown in (a) and the success rate is shown in (b). A total of 100 independent experimental runs have been done.

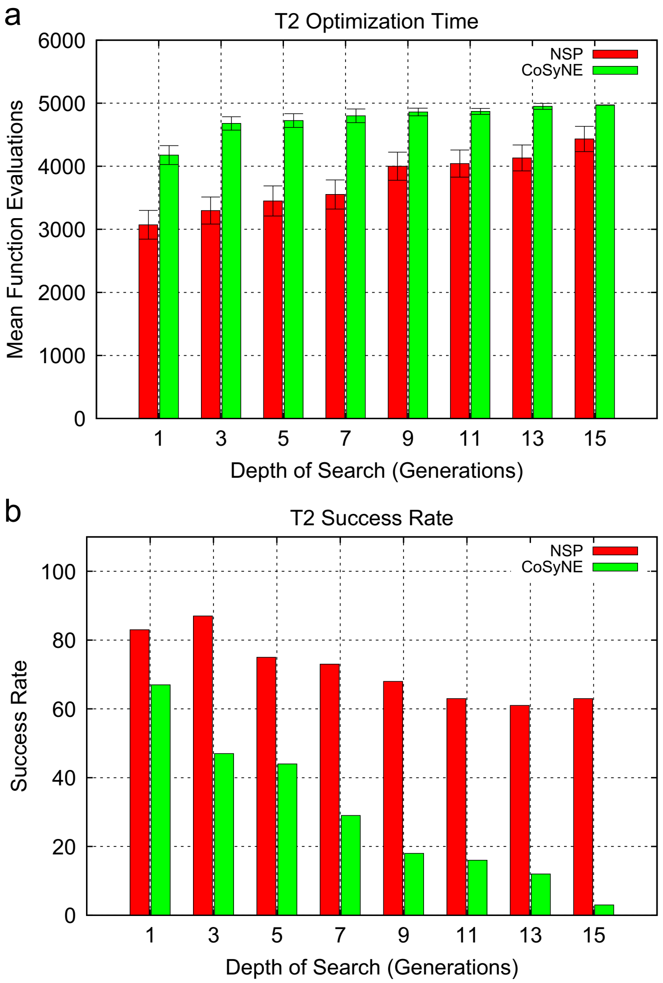

Fig. 7. The performance of NSP and CoSyNE on different depths of search in terms of the number of generations for the T2 problem. The optimization time in terms of the average number of function evaluations is shown in (a) and the success rate is shown in (b). A total of 100 independent experimental runs have been done.

results are for the evolution phase only. The error bars in the histograms of the respective figures show the confidence interval. The results do not include the time taken for the initialization phase as the goal is to observe the convergence of the respective methods during evolution.

In the T1 problem shown in Fig. 6, the depth of 1–5 generations in NSP gives similar performance, while in CoSyNE, the depth of 1 generation only 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 Fig. 7, the depth of 1–7 generations in NSP gives similar performance, while in CoSyNE, the depth of 1 generation gives the best result.

In the T3 problem shown in Fig. 8, the depth of 1–5 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 Fig. 9, the depth of 1–5 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 Fig. 10, 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 determine 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 to 5 generations give 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 show that NSP has been able to solve the problems in less optimization time with higher success rates 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 (1–5 generations) in most of the problems.

The generalization performance is given in Table 1 show that in all the problems, both methods have been able to achieve 100% generalization 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 performance for this value, i.e. a shallow depth.

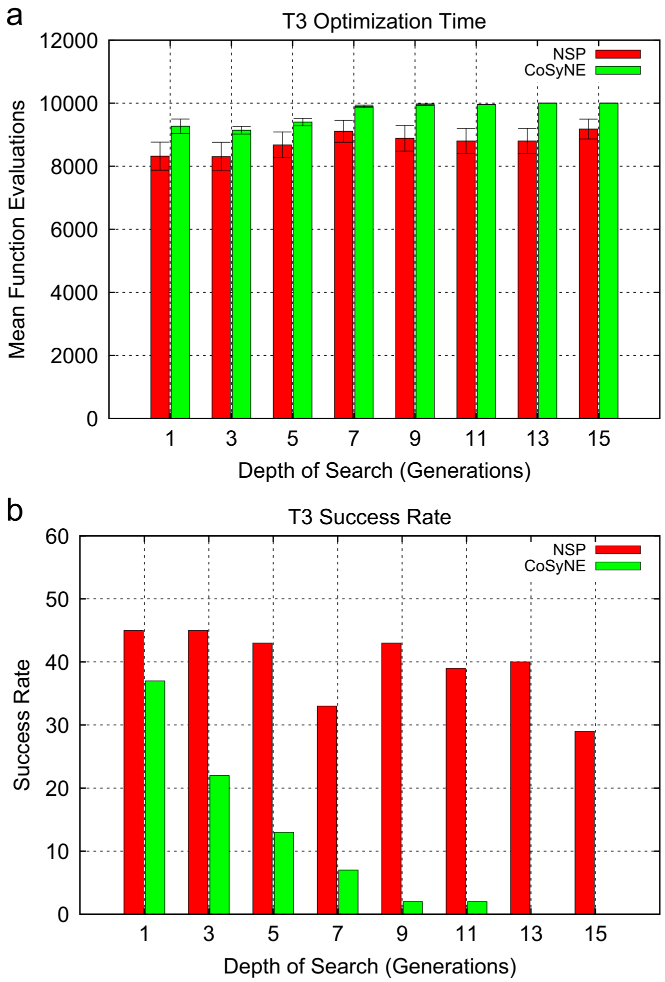

Fig. 8. The performance of NSP and CoSyNE on different depths of search in terms of the number of generations for the T3 problem. The optimization time in terms of the average number of function evaluations is shown in (a) and the success rate is shown in (b). A total of 100 independent experimental runs have been done.

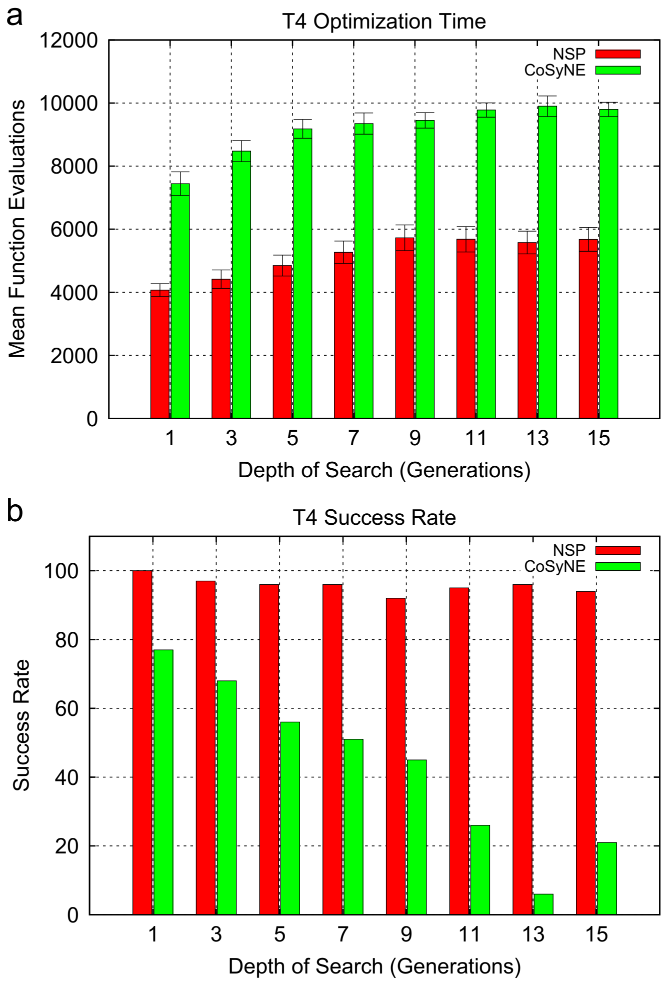

Fig. 9. The performance of NSP and CoSyNE on different depth of search in terms of the number of generations for the T4 problem. The optimization time in terms of the average number of function evaluations is shown in (a) and the success rate is shown in (b). A total of 100 independent experimental runs have been done.

4.2. 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 genetic algorithm in their subpopulation. The G3-PCX is used in both encoding schemes. The depth of 1 generation is used in NSP and CoSyNE.

The goal of this experiment is to observe the performance of the respective methods with a fixed recurrent network topology. Note that the number of hidden neurons directly influences the difficulty of the learning problem. It is more difficult to learn the problem if fewer neurons are present in the hidden layer.

Table 2 shows the relationship between the number of function evaluations and the number of hidden neurons used in the initialization phase of NSP and CoSyNE. The RNN topology has one input neuron and two 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 subpopulations. Note that NSP uses fewer number of function evaluations shown in Table 2 as it requires a smaller number of subcomponents when compared to CoSyNE. Therefore, the initialization phase of evaluating different subcomponent encoding schemes for the cooperative coevolution framework is an important measure.

In all problems, the RNN is trained until the mean-squared-error (MSE) is below 0.0005, or 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 a maximum of 10 000 function evaluations.

The comparative results during evolution are given in Figs. 11–15. 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 numbers of hidden neurons. A total of 100 experiments are performed for each case and the mean function evaluation is given for each problem. The total optimization time includes the initialization 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 are easier problems than the rest of the problems so they can be learned in a relatively shorter time. In the results for the T1 problem shown in Fig. 11, NSP shows better performance than CoSyNE in (a) the evolution phase and (b) the initialization plus the evolution phase with higher success rate shown in (c). The results are similar in the case of the T2 problem shown in Fig. 12, except that for four hidden neurons, the performance of NSP and CoSyNE is similar in the evolution phase (a). However, NSP shows better performance when the initialization phase is added to the evolution phase (total time) for four hidden neurons in (b).

Please cite this article as: R. Chandra, et al., Encoding subcomponents in cooperative co-evolutionary recurrent neural networks, Neurocomputing (2011), doi:10.1016/j.neucom.2011.05.003

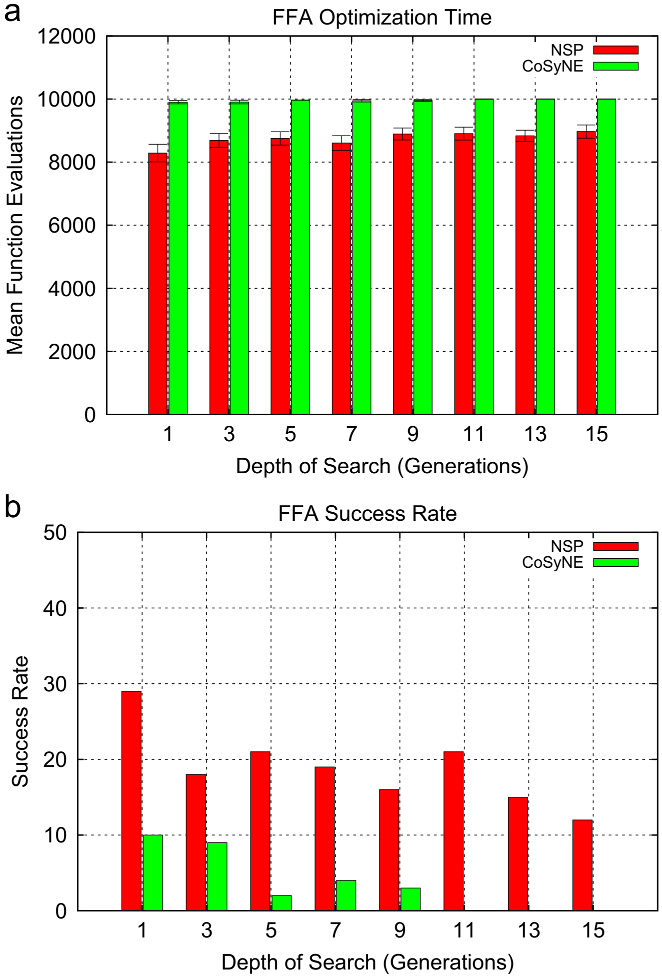

Fig. 10. The performance of NSP and CoSyNE on different depths of search in terms of the number of generations for the FFA problem. The optimization time in terms of the average number of function evaluations is shown in (a) and the success rate is shown in (b). A total of 100 independent experimental runs have been done.

Table 1
The generalization performance in percentage is given by NSP and CoSyNE for the depth of search of 1 generation. Note that the generalization performance does not include the performance of the unsuccessful runs in the mean. The success rate from 100 experiments is also given.

Problem	NSP (%)	Success rate	CoSyNE (%)	Success rate
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 2
A comparison of NSP and CoSyNE based on the number of function evaluation required during initialization. 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.

No. of hidden neurons	NSP	CoSyNE
3	8	23
4	10	34
5	12	47
6	14	62
7	16	79

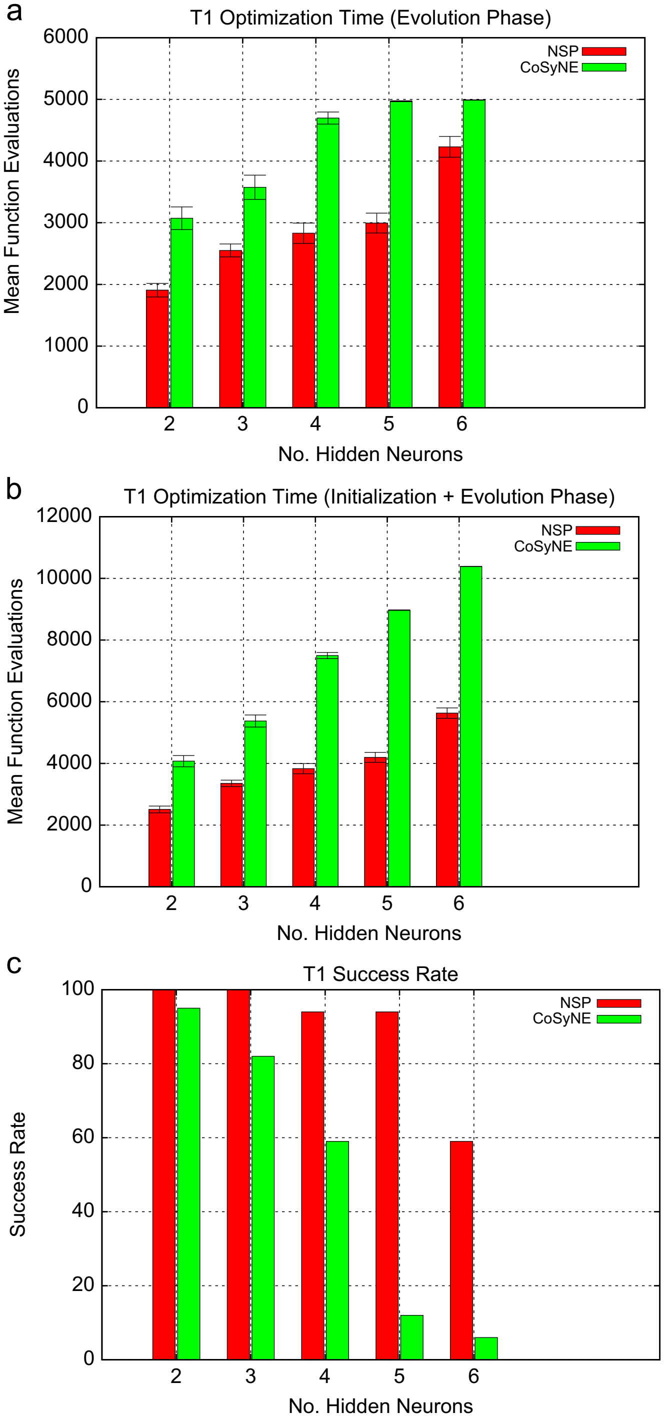

Fig. 11. The performance of NSP and CoSyNE on different numbers of hidden neurons for the T1 problem. The optimization time in terms of the average number of function evaluations (evolution phase) is shown in (a), the total optimization time which includes the initialization and evolution phase is shown in (b) and the success rate is shown in (c).

In the T3 problem shown in Fig. 13, the evolution phase in (a) shows that the performance of NSP is similar to CoSyNE for four and seven hidden neurons. In (b), NSP gives better performance for the total time. In the T4 problem shown in Fig. 14, NSP gives better performance than CoSyNE in general.

In the FFA problem shown in Fig. 15, the performance of NSP is weaker than CoSyNE for six hidden neurons and similar for seven and eight hidden neurons as shown in (a). NSP outperforms CoSyNE in all the cases for the total time in (b).

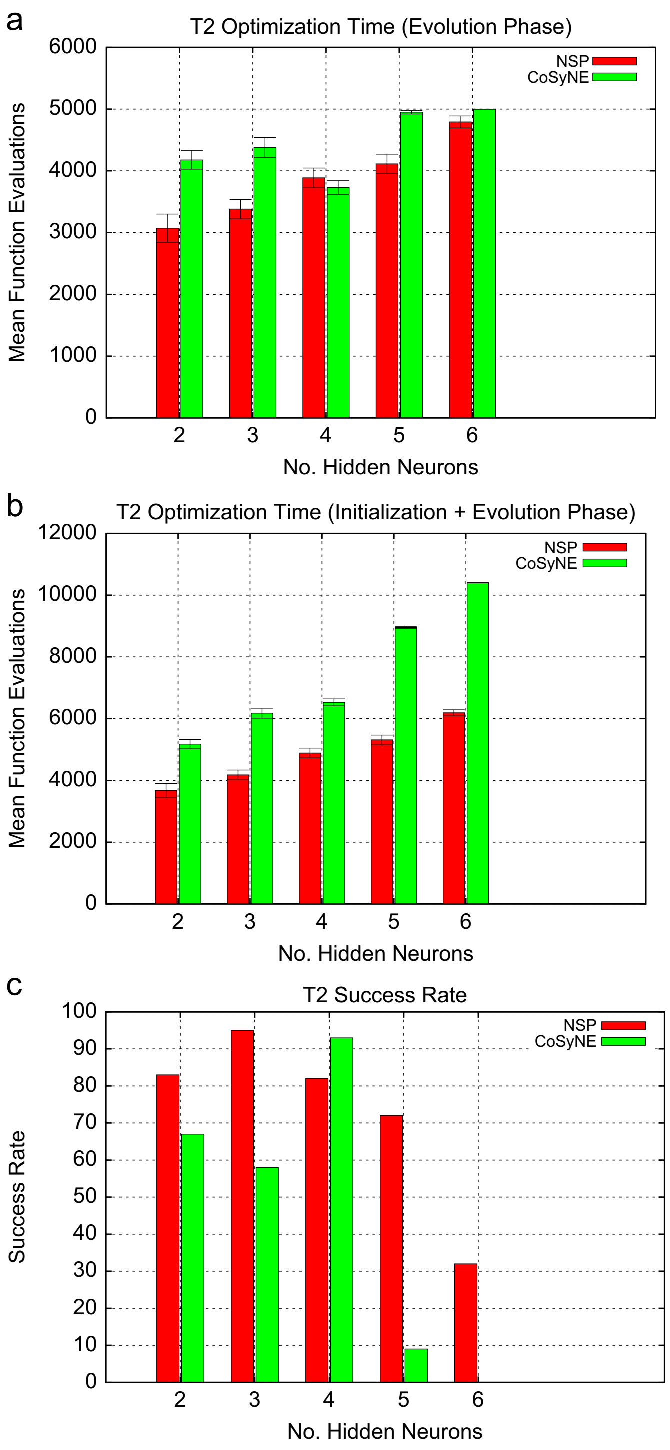

Fig. 12. The performance of NSP and CoSyNE on different numbers of hidden neurons for the T2 problem. The optimization time in terms of the average number of function evaluations (evolution phase) is shown in (a), the total optimization time which includes the initialization and evolution phase is shown in (b) and the success rate is shown in (c).

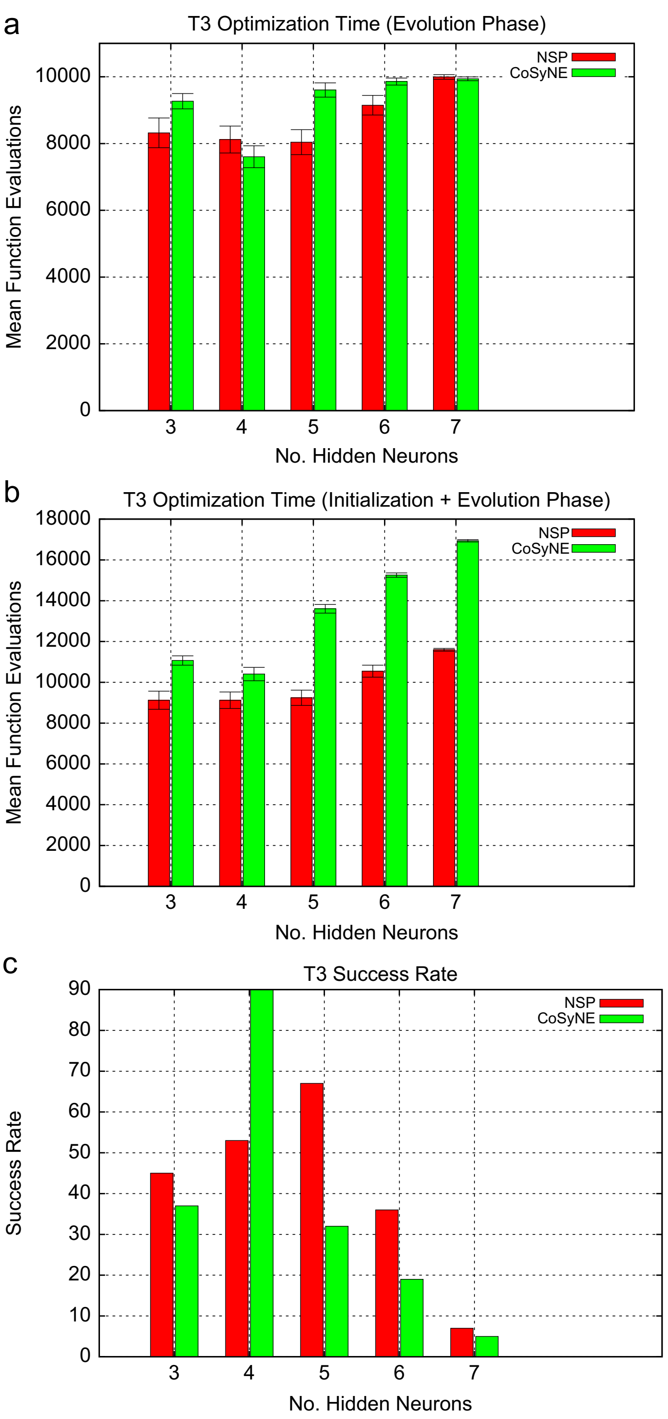

Fig. 13. The performance of NSP and CoSyNE on different numbers of hidden neurons for the T3 problem. The optimization time in terms of the average number of function evaluations (evolution phase) is shown in (a), the total optimization time which includes the initialization and evolution phase is shown in (b) and the success rate is shown in (c).

4.3. *Discussion*

Cooperative coevolution naturally 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 [35,36,29]. In the case of neuro-evolution, the architectural properties of the neural network have been used to group the subcomponents. In this paper, NSP encoding scheme has been introduced which groups subcomponents according to the synapse-links that are connected to a neuron.

The results reveal that good performance can be achieved when each subpopulation 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 lead to a non-separable problem. The level of non-separability is dependent on the particular neural

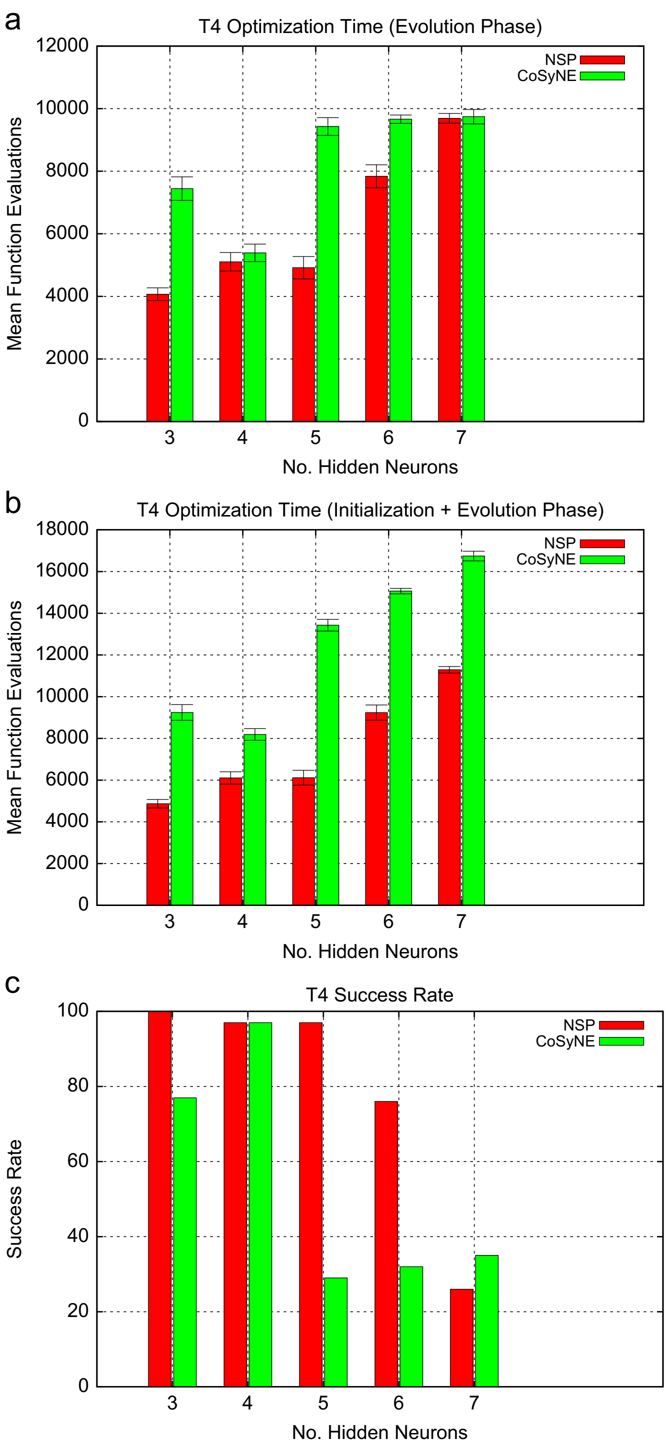

Fig. 14. The performance of NSP and CoSyNE on different numbers of hidden neurons for the T4 problem. The optimization time in terms of the average number of function evaluations (evolution phase) is shown in (a), the total optimization time which includes the initialization and evolution phase is shown in (b) and the success rate is shown in (c).

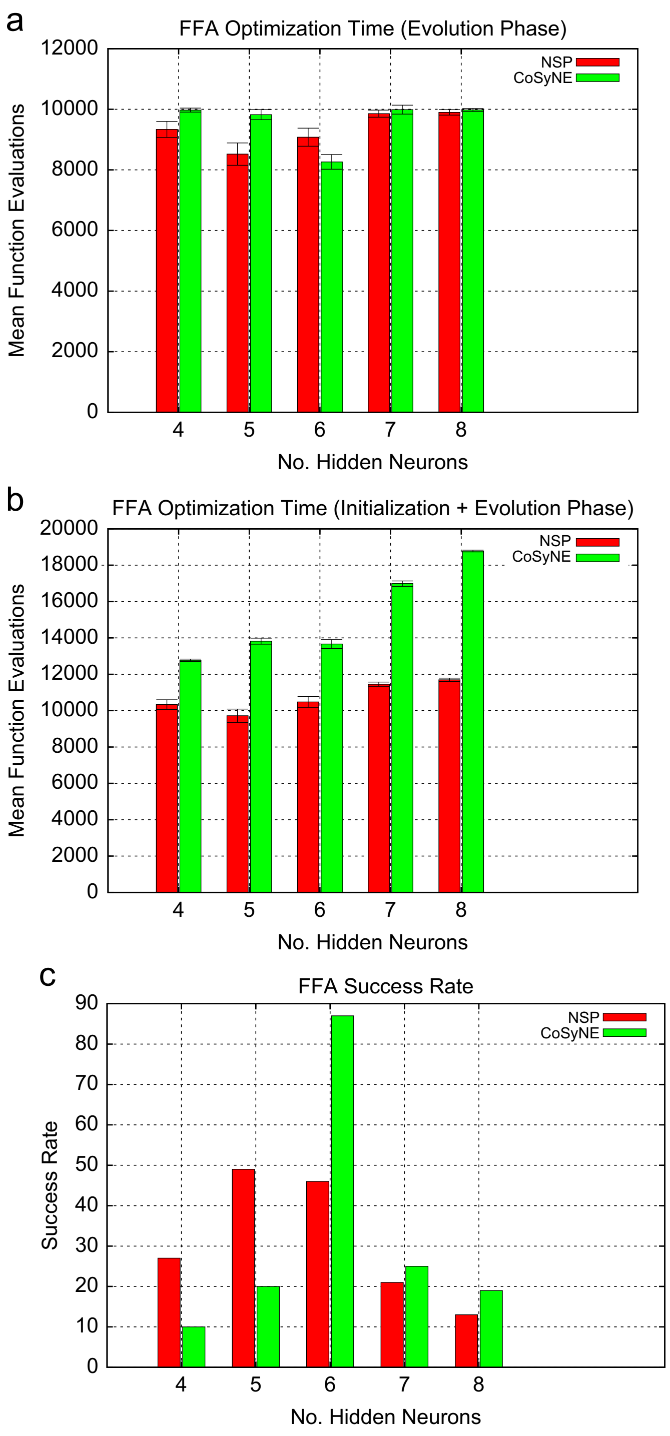

Fig. 15. The performance of NSP and CoSyNE on different numbers of hidden neurons for the FFA problem. The optimization time in terms of the average number of function evaluations (evolution phase) is shown in (a), the total optimization time which includes the initialization and evolution phase is shown in (b) and the success rate is shown in (c).

network architecture and application problem. The deep greedy search for a large number of generations for each subpopulation has not shown good performance and gives an indication that recurrent networks on the given grammatical inference problems are partially separable. The deep greedy search in NSP has shown similar performance for 1–5 generations. The performance deteriorates with deeper depth of search. This implies that the degree of non-separability is high in recurrent 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 might be the main reason that a deeper greedy search for the case of using NSP for evolving recurrent networks has not been beneficial when compared to NSP in feedforward networks [14].

The comparative performance shows that the proposed NSP framework shows a better performance than CoSyNE in most cases. Note that these results are specific for grammatical inference problems. NSP has been 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 is can represent the same problem in a smaller number of subcomponents than CoSyNE and at the same time it provides similar optimization performance. This advantage further enables NSP to have fewer function evaluations in the initialization phase as verified by the results in Table 2. The initialization 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 two 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 view the network as partially separable.

The generalization performance given in Table 1 shows that the performance of NSP has been similar to CoSyNE. This indicates that NSP has achieved the same solution quality with a lower optimization time and a better success rate.

NSP can be used for learning long-term dependency problems. This is due to the non-gradient requirement of neuro-evolution in optimizing 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 problems were successfully trained by NSP, which would have not been possible with backpropagation-through-time [34].

The performance of CoSyNE is also poor when compared to NSP for training feedforward neural networks in pattern recognition problems [14]. However, CoSyNE showed impressive results for pole balancing problems in [11]. Presumably, this is due to the nature of the problem.

5. Conclusions and future work

This paper introduced a novel problem decomposition method (NSP) for the cooperative coevolution of recurrent neural networks. The method has been evaluated on grammatical inference problems for recurrent neural networks.

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

It is important to evaluate the optimal depth of search in the subcomponents. The results show that the depth of search makes a higher impact on CoSyNE when compared to NSP. The depth of search for 1 generation only gives good results in the CoSyNE algorithm in all the given problems. The NSP encoding has shown better performance than CoSyNE for grammatical inference problems in general.

The NSP has shown to efficiently decompose the problem by grouping interacting variables into the separate subcomponents. This has been verified as the depth of search does not affect the performance of NSP as in the case of CoSyNE. This indicates that there are fewer interactions among the subcomponents during evolution in NSP. The synapse level encoding does not provide any grouping of interacting variables. Therefore, the depth of a single generation enables CoSyNE to deal with the higher degree of non-separability. A larger depth of search in CoSyNE would have given good performance if the problem was fully separable.

Future work can examine the implementation of the NSP cooperative coevolution framework for evolving both the weights and the network topology during training. The paradigm where the different encoding schemes can be combined during training can also be explored. This would be implemented by adapting the encoding scheme as the algorithm is progressing toward the final solution.

References

[1] C.L. Giles, C. Omlin, K.K. Thornber, Equivalence in knowledge representation: automata, recurrent neural networks, and dynamical fuzzy systems, Proc. IEEE 87 (9) (1999) 1623–1640.

[2] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (5) (1989) 359–366.

[3] F. Scarselli, A.C. Tsoi, Universal approximation using feedforward neural networks: a survey of some existing methods, and some new results, Neural Networks 11 (1) (1998) 15–37.

[4] A. Schaefer, H. Zimmermann, Recurrent neural networks are universal approximators, Int. J. Neural Syst. 17 (4) (2007) 253–263.

[5] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Networks 5 (2) (1994) 157–166.

[6] P. Frasconi, M. Gori, A. Tesi, Successes and failures of backpropagation: a theoretical investigation, in: Progress in Neural Networks, Ablex Publishing, 1993, pp. 205–242.

[7] X. Yao, Evolving artificial neural networks, Proc. IEEE 87 (9) (1999) 1423–1448.

[8] M.A. Potter, K.A. De Jong, 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, Springer-Verlag, London, 1994, pp. 249–257.

[9] F. Gomez, R. Mikkulainen, Incremental evolution of complex general behavior, Adapt. Behav. 5 (3–4) (1997) 317–342 〈http://dx.doi.org/10.1177/105971239700500305〉.

[10] F.J. Gomez, Robust Non-Linear Control through Neuroevolution, Technical Report AI-TR-03-303, Ph.D. Thesis, Department of Computer Science, The University of Texas at Austin, 2003.

[11] F. Gomez, J. Schmidhuber, R. Miikkulainen, Accelerated neural evolution through cooperatively coevolved synapses, J. Mach. Learn. Res. 9 (2008) 937–965.

[12] N. Garcia-Pedrajas, C. Hervas-Martinez, J. Munoz-Perez, COVNET: a cooperative coevolutionary model for evolving artificial neural networks, IEEE Trans. Neural Networks 14 (3) (2003) 575–596.

[13] N. Garcia-Pedrajas, C. Hervas-Martinez, D. Ortiz-Boyer, Cooperative coevolution of artificial neural network ensembles for pattern classification, IEEE Trans Evolut. Comput. 9 (3) (2005) 271–302.

[14] R. Chandra, M. Frean, M. Zhang, An encoding scheme for cooperative coevolutionary neural networks, in: Twenty-Third Australian Joint Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence, Springer-Verlag, Adelaide, Australia, 2010, pp. 253–262.

[15] Y. Liu, X. Yao, Q. Zhao, T. Higuchi, Scaling up fast evolutionary programming with cooperative coevolution, in: Proceedings of the 2001 Congress on Evolutionary Computation, 2001, pp. 1101–1108, doi:10.1109/CEC.2001.934314.

[16] R. Salomon, 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.

[17] J. Schmidhuber, D. Wierstra, M. Gagliolo, F. Gomez, Training recurrent networks by Evolino, Neural Comput. 19 (3) (2007) 757–779 〈http://dx.doi.org/10.1162/neco.2007.19.3.757〉.

[18] A. Blanco, M. Delgado, M.C. Pegalajar, A real-coded genetic algorithm for training recurrent neural networks, Neural Network 14 (1) (2001) 93–105 〈http://dx.doi.org/10.1016/S0893-6080(00)00081-2〉.

[19] M. Tomita, 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.

[20] M.A. Castaño, E. Vidal, F. Casacuberta, Finite state automata and connectionist machines: a survey, in: J. Mira, F.S. Hernández (Eds.), IWANN, Lecture Notes in Computer Science, vol. 930, Springer, 1995, pp. 433–440.

[21] J.L. Elman, Finding structure in time, Cognitive Sci. 14 (1990) 179–211.

[22] P. Manolios, R. Fanelli, First-order recurrent neural networks and deterministic finite state automata, Neural Comput. 6 (6) (1994) 1155–1173 〈http://dx.doi.org/10.1162/neco.1994.6.6.1155〉.

[23] C.W. Omlin, C.L. Giles, Constructing deterministic finite-state automata in recurrent neural networks, J. ACM 43 (6) (1996) 937–972 〈http://doi.acm.org/10.1145/235809.235811〉.

[24] C.W. Omlin, K.K. Thornber, C.L. Giles, Fuzzy finite state automata can be deterministically encoded into recurrent neural networks, IEEE Trans. Fuzzy Syst. 6 (1998) 76–89.

[25] R.L. Watrous, G.M. Kuhn, Induction of finite-state languages using second-order recurrent networks, Neural Comput. 4 (3) (1992) 406–414 〈http://dx.doi.org/10.1162/neco.1992.4.3.406〉.

[26] R. Chandra, C.W. Omlin, Training and extraction of fuzzy finite state automata in recurrent neural networks, in: Proceedings of International Conference on Computational Intelligence, 2006, pp. 274–279.

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

[28] D. Ortiz-Boyer, C. HerváMartínez, N. García-Pedrajas, Cixl2: a crossover operator for evolutionary algorithms based on population features, J. Artif. Intell. Res. 24 (2005) 1–48.

[29] Z. Yang, K. Tang, X. Yao, Large scale evolutionary optimization using cooperative coevolution, Inf. Sci. 178 (15) (2008) 2985–2999 〈http://dx.doi.org/10.1016/j.ins.2008.02.017〉.

[30] Z. Yang, K. Tang, X. Yao, Multilevel cooperative coevolution for large scale optimization, in: IEEE Congress on Evolutionary Computation, 2008, pp. 1663–1670.

[31] M. Omidvar, X. Li, X. Yao, Cooperative co-evolution for large scale optimization through more frequent random grouping, in: IEEE Congress on Evolutionary Computation (CEC), 2010, 2010, pp. 1754–1761.

[32] M.A. Potter, K.A. De Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evol. Comput. 8 (1) (2000) 1–29 〈http://dx.doi.org/10.1162/106365600568086〉.

[33] K. Deb, A. Anand, D. Joshi, A computationally efficient evolutionary algorithm for real-parameter optimization, Evol. Comput. 10 (4) (2002) 371–395.

[34] P.J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560.

[35] F. van den Bergh, A. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Trans. Evolut. Comput. 8 (3) (2004) 225–239.

[36] Y.-j. Shi, H.-f. Teng, Z.-q. Li, Cooperative co-evolutionary differential evolution for function optimization, in: L. Wang, K. Chen, Y.S. Ong (Eds.), Advances in Natural Computation, Lecture Notes in Computer Science, vol. 3611, Springer Berlin/Heidelberg, 2005, pp. 1080–1088.

Rohitash Chandra is a Ph.D. Candidate in Computer Science at Victoria University of Wellington. He has been a Lecturer in Computing Science at the Department of Computing Science and Information Systems, Fiji National University, Fiji. His research interests are in recurrent neural networks, cooperative coevolution, environmental decision support systems, neuro-evolution, time series prediction and robot kinematics.

Apart from his interest in science, he is actively involved in literature and has been the editor of Blue Fog Journal. He writes poetry and has published his second collection called “A Hot Pot of Roasted Poems”. His third collection, “Being at Home” is due to be launched by the end of this year.

Marcus Frean is a Senior Lecturer in Computing Science at the School of Engineering and Computer Science, Victoria University of Wellington. His research interests are in intelligent systems and machine learning.

Mengjie Zhang is an Associate Professor in Computing Science at the School of Engineering and Computer Science, Victoria University of Wellington. His research interests are in genetic programming and particle swarm optimization, intelligent and evolutionary computer vision, data mining and machine learning.

Christian W. Omlin is Professor in Computer Engineering in the Department of Computer Engineering at Middle East Technical University, Northern Cyprus Campus. His research interests are in recurrent neural networks, knowledge representation, and gesture recognition.