Soft Comput (2012) 16:1009–1020
DOI 10.1007/s00500-011-0798-9

ORIGINAL PAPER

Adapting modularity during learning in cooperative co-evolutionary recurrent neural networks

Rohitash Chandra · Marcus Frean · Mengjie Zhang

Published online: 24 December 2011

Abstract 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 organizing the training problem into modules and independently solving them. The number of modules required to represent a neural network is critical to the success of evolution. This paper proposes a framework for the adaptation of the number of modules during evolution. The framework is called adaptive modularity cooperative coevolution. It is used for training recurrent neural networks on grammatical inference problems. The results shows that the proposed approach performs better than its counterparts as the dimensionality of the problem increases.

Keywords Cooperative coevolution · Neuro-evolution · Recurrent neural networks · Grammatical inference · Evolutionary algorithms

1 Introduction

Cooperative coevolution (CC) is a biologically inspired evolutionary computation framework that divides a problem into subcomponents which are implemented as sub-populations. The sub-populations are genetically isolated and cooperation takes place during fitness evaluation. In the original cooperative co-evolutionary framework, the problem is decomposed by having a separate subcomponent for each variable (Potter and Jong 1994). It was later found that the strategy was only effective for problems which are separable (Liu et al. 2001). In separable problems, there is no interdependency between the decision variables whereas in non-separable problems, interdependencies exist. Cooperative coevolution naturally appeals to separable problems as there is no interaction among the subcomponents during evolution (Salomon 1996). In most problems, groups of interacting and non-interacting variables exist which determine the *degree of non-separability*.

In order to take full advantage of cooperative coevolution, it is important to group interacting variables in a single subcomponent and non-interacting variables into separate subcomponents. However, it is difficult to identify interacting variables. van den Bergh and Engelbrecht (2004) used a de-compositional strategy which decomposed the n dimensional problem into m s-dimensional sub-problems, which showed better performance than the original CC framework. 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.

The CC framework has been used for training feedforward (Potter and De Jong 2000; García-Pedrajas and Ortiz-Boyer 2007) and recurrent neural networks (Gomez and Mikkulainen 1997; Schmidhuber et al. 2007); however, the attention has not been on the issue of separability and interacting variables. It is important to use the right problem decomposition method to group interacting variables that relate to the neural network training problem and architecture.

In training recurrent neural networks, cooperative co-evolution has been deployed as *enforced sub-populations* (ESP) (Gomez and Mikkulainen 1997; Gomez 2003) where a subcomponent consists of the input, output and recurrent

R. Chandra (✉) · M. Frean · M. Zhang
School of Engineering and Computer Science,
Victoria University of Wellington, P.O. Box 600,
Wellington 6140, New Zealand
e-mail: c.rohitash@gmail.com

connections to a hidden neuron. A sophisticated version of ESP known as *Evolino* has been used to evolve long-short term memory networks (Schmidhuber et al. 2007). The cooperatively coevolved synapse (CoSyNE) decomposes the training problem to the synapse level where a subcomponent is a single weight connection. It has shown better performance than ESP and standard neuro-evolutionary methods for pole-balancing problems studied in (Gomez et al. 2008). A neuron level decomposition has been proposed in neuron-based sub-population (NSP) method which has shown slightly better performance than CoSyNE for grammatical inference problems (Chandra et al. 2011b).

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 recognition problems (Chandra et al. 2010). The decomposition by ESP and NSP exhibits different degrees of non-separability. NSP showed better performance than ESP and CoSyNE in pattern recognition problems (Chandra et al. 2010).

It is reasonable to adapt the encoding scheme (modularity) to implement different levels of search in terms of the degree of non-separability needed at different stages of evolution. This work proposes an adaptive modularity cooperative coevolution (AMCC) framework for training recurrent neural networks in grammatical inference problems. Instead of using a fixed level of modularity during the entire evolution, the AMCC framework adapts the number of modules at different stages during evolution. The performance of the AMCC is compared with standard neuro-evolution and the NSP CC framework as they have been efficient for grammatical inference problems. Our recent study has shown promising results in the application of AMCC for training feedforward networks on pattern recognition problems (Chandra et al. 2011a).

The G3-PCX evolutionary algorithm (Deb et al. 2002) is employed as the designated evolutionary algorithm in all of the sub-populations. The Elman recurrent neural network (Manolios and Fanelli 1994) is used in all experiments. Specific grammatical inference problems are taken from Blanco et al. (2001) and from the Tomita grammar (Tomita 1982) as it has been a major benchmark for training recurrent networks (Castao et al. 1995).

The rest of the paper is organised as follows. In Sect. 2, the background on recurrent neural networks and grammatical inference problems is presented. The cooperative coevolution framework is presented in Sect. 3 while Sect. 4 gives details of the proposed AMCC framework. Section 5 presents the results and Sect. 6 concludes the work with a discussion on future work.

2 Recurrent neural networks

Recurrent neural networks have been an important focus of research as they can be applied to difficult problems involving time-varying patterns. They are suitable for modeling temporal sequences. They have been applied to difficult real world problems such as speech recognition (Robinson 1994), financial prediction (Giles et al. 1997) and gesture recognition (Marakami and Taguchi 1991). A detailed study on the theoretical foundation, design and application of recurrent neural networks is done in (Haykin et al. 2006; Kolen and Kremer 2001; Medsker and Jain 1999).

Recurrent neural networks are dynamical systems whose next state and output(s) depend on the present network state and input(s); therefore, they are particularly useful for modelling dynamical systems. A detailed study on different recurrent network architectures is beyond this work; however, we will present the details of first-order recurrent networks as they are used for the remainder of this discussion.

First-order recurrent neural networks use context units to store the output of the state neurons from computation of previous time steps. The Elman architecture (Elman 1990) employs the context layer which makes a copy of the hidden layer outputs in the previous time steps as shown in Fig. 1. They are composed of an *input layer*, a *context layer* which provides state information, a *hidden layer* and an *output layer* as shown in Fig. 1. Each layer contains one or more neurons which propagate information from one layer to the next by computing a non-linear function of their weighted sum of inputs. The equation of the dynamics of the change of hidden state neuron activations in the context layer is given by Eq. 1.

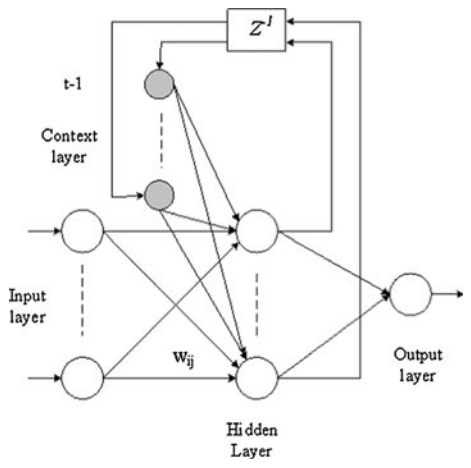


Fig. 1 Elman style first-order RNN architecture (Elman 1990)

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

where S_k (t) and I_j (t) represent the output of the context or recurrent state and input neurons, respectively and V_ik and W_ij represent their corresponding weights. t is time and I_j is the input to the jth neuron. S_k is the kth neuron of the recurrent state.

2.1 Grammatical inference

Grammatical inference problems have been used to study training algorithms and knowledge representation in recurrent neural networks. It has been demonstrated through knowledge extraction that recurrent networks can represent finite-state automata (Omlin and Giles 1998; Watrous and Kuhn 1992; Chandra and Omlin 2006). Grammatical inference has been used as an appropriate test bed for the investigation of the performance of learning algorithms. The Tomita grammar (Tomita 1982) has been used as a benchmark problem to evaluate RNN training algorithms and architectures (Castao et al. 1995; Gabrijel and Dobnikar 2003). A formal definition on deterministic and fuzzy finite-state automata 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 $\delta : Q \times \Sigma \rightarrow Q$ defines where the state $q' = \delta(q, \sigma)$ has reached 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 (Brookshear 1989).

The language $L(M)$ accepted by the automaton contains all the strings which take the automaton from its initial to an accepting state. The languages accepted by DFAs are called regular languages. Figure 2 shows the DFAs

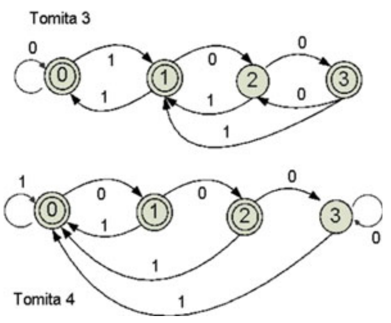


Fig. 2 Deterministic finite-state automata from the Tomita grammar (Gabrijel and Dobnikar 2003). *Double circles* show accepting states while rejecting states are shown by *single circles*. State 1 is the automatons start state

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

Definition 2 A fuzzy finite-state automaton (FFA) 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 (Brookshear 1989).

A restricted type of fuzzy automata 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. Any fuzzy automaton as described in *Definition 1* is equivalent to a restricted fuzzy automaton. The transformation of a fuzzy automaton to its corresponding deterministic acceptor is discussed in Giles et al. (1999). Figure 3 shows an example of a FFA with its corresponding deterministic acceptor. This FFA has been used to show that recurrent networks can be trained by evolutionary algorithms (Blanco et al. 2001).

3 Cooperative coevolution and neuro-evolution

The cooperative coevolution framework divides the problem into subcomponents that are implemented as sub-populations (Potter and Jong 1994). 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 sub-populations are evolved in a *round-robin* fashion for a given number of generations. The original CC framework has been used for general function optimisation and the problems were decomposed to its lowest level where a separate subcomponent was used to represent each dimension of the problem (Potter and Jong 1994). It was later found that this strategy is only effective for problems which are fully separable (Liu et al. 2001). Much work has been done in the use of cooperative coevolution in large-scale function optimization and the focus has been on non-separable problems (Liu et al. 2001; van den Bergh and Engelbrecht 2004; Shi et al. 2005; Yang et al. 2008; Chen et al. 2010; Omidvar et al. 2011; Li and Yao 2011a).

A function of n variables is separable if it can be written as a sum of n functions with just one variable (Ortiz-Boyer et al. 2005). The parameters of a separable problem are called independent variables. 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. Yang et al. (2008a) presented the cooperative coevolution framework that employs a *random grouping* and *adaptive weighting* strategy with differential

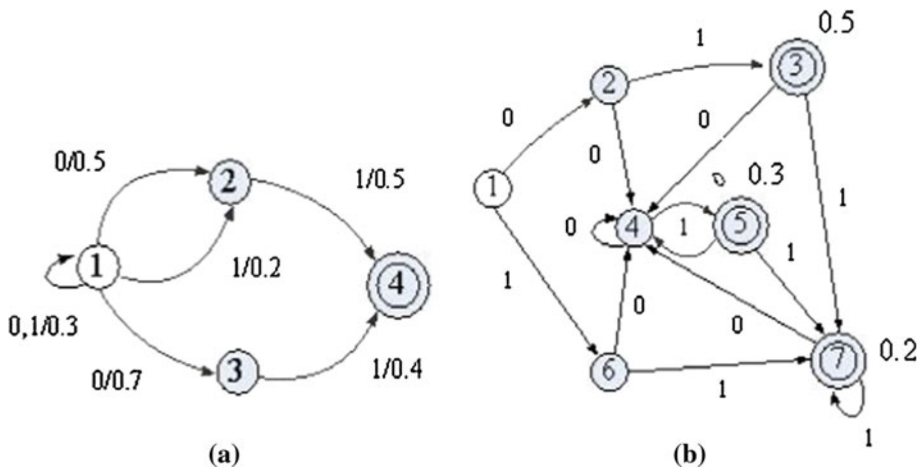


Fig. 3 The fuzzy finite-state automata (**a**) and its equivalent deterministic acceptor (**b**) taken from Blanco et al. (2001). The accepting states are labelled with a degree of membership. State 1 is the automatons start state; accepting states are drawn with *double circles*

evolution (DECC-G) for its subcomponents. The method groups interacting and non-interacting variables into separate subcomponents heuristically. Yang et al. (2008b) also presented a multi-level cooperative coevolution framework (MLCC) which adapts the size of the subcomponents in DECC-G to group interacting and non-interacting variables. The framework starts with small subcomponents and adapts to bigger subcomponents whose size are taken from a predefined set. MLCC showed better performance than DECC-G for non-separable problems of upto 1,000 dimensions. Omidvar et al. (2010) made amendments to the random grouping approach. They presented a *more frequent random grouping* approach which turned to outperform its counterpart in several non-separable problems of up to 1,000 dimensions. Chen et al. (2010) presented cooperative coevolution with variable interaction learning that treats all variables independent initially and groups them according to the interactions. Rather than evolving the subcomponents in a round-robin fashion, Omidvar et al. (2011) presented a contribution-based cooperative coevolution approach that selects and evolves the subcomponents based on their contributions to the global fitness. Li and Yao (2011) have recently employed particle swarm optimisation-based cooperative coevolution for large-scale problems of up to 2,000 dimensions.

Real-world applications mostly fall between fully separable and non-separable problems. Cooperative coevolution has been effective for separable problems. Evolutionary algorithms without any decomposition strategy appeal to fully non-separable problems.

The *depth of search* determines the number of generations each sub-population is evolved in a *round-robin* fashion. The depth of search has to be predetermined according to the nature of the problem. The depth of search can reflect whether the encoding scheme has been able to group the interacting variables into separate subcomponents (Chandra et al. 2011b). If the interacting variables have been grouped efficiently, then a deep greedy search for the sub-population is possible, implying that the problem has been efficiently broken down into subcomponents which have fewer interactions among themselves.

3.1 Encoding schemes for recurrent networks

In standard neuro-evolution, where the entire network is encoded in a single population, information associated with state and hidden neurons can be lost due to reproduction. This is due to the nature of the crossover and mutation operators which can affect all the variables in the population. Cooperative coevolution has been used in training recurrent neural networks to preserve information associated with the recurrent state neurons. This is done by having separate sub-populations for each hidden neuron.

There are 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 module in the CC framework. Therefore, the number of hidden neurons is equal to the number of subcomponents. In ESP (Gomez and Mikkulainen 1997; Gomez 2003), a particular neuron h_i in the hidden layer encodes all the incoming, outgoing and recurrent weight connections.

Another neuron level encoding scheme is known as the NSP (Chandra et al. 2011b). Each subcomponent in the NSP consists of incoming weight links associated with a neuron in the hidden, state (recurrent), and output later. Therefore, each sub-population for a layer is composed of the following:

1. *Hidden layer sub-populations* Weight-links from each neuron in the $hidden_j(t)$ layer connected to all $input_i(t)$ neurons and the bias of $hidden_j(t)$, where t is the time. $input_i(t)$ is the input layer and $hidden_j(t)$ is the hidden layer.

2. *State (recurrent) neuron sub-populations* Weight-links from each neuron in the $hidden_j(t)$ layer connected to all hidden neurons in previous time step $hidden_j(t-1)$.
3. *Output layer sub-populations* Weight-links from each neuron in the $output_k(t)$ layer connected to all $hidden_j(t)$ neurons and the bias of $output_k(t)$, where $output_k(t)$ is the output layer.

NSP has performed better than CoSyNE and ESP for pattern recognition problems in (Chandra et al. 2010). CoSyNE views the recurrent network as a fully separable problem and has been successful for pole balancing (Gomez et al. 2008); however, it performed poorly for pattern recognition problems (Chandra et al. 2010).

The second encoding scheme has been presented in the CoSyNE which decomposes the network into synapse level. Therefore, the number of modules depends on the number of weights and biases. The CoSyNE demonstrated better performance than ESP on the double pole balancing problem (Gomez et al. 2008). Note that NSP and CoSyNE will be used in the AMCC framework.

4 The adaptive modularity cooperative coevolution framework

The general idea behind the AMCC framework is to use the strength of a particular encoding scheme which reflects on the degree of non-separability when needed during evolution. The adaptive framework 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 framework employs different levels of modularity which is implemented by the different encoding schemes. It employs 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 framework for training recurrent neural networks transforms 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 modules using all the three level of encoding forms the general AMCC framework. The levels of encoding that exhibit modularity are further described as follows:

1. *Synapse level encoding* Decomposes the network into its lowest level to form a single module (Gomez et al. 2008). 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 (Chandra et al. 2011b).
3. *Network level encoding* The standard neuro-evolutionary encoding scheme where only one population represents the entire network. There is no decomposition present in this level of encoding.

The AMCC framework is described in Algorithm 1 which follows three main stages of evolution. In the beginning, all the sub-populations of the Synapse level, neuron level and network level encoding are randomly 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.

Algorithm. 1 Adaptive Modularity in Cooperative Coevolution

```
Initialise Synapse, Neuron and Network level

Stage 1: Synapse level encoding
Cooperatively evaluate Synapse level only

while (FuncEval ≤ 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 ≤ 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 ≤ MaxGlobal do
    Network level Genetic Operators

    if (FuncEval ≤ MinGlobal) then
        Go back to Neuron level encoding
    end
end
```

Stage 1 employs 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 in any of the stages. However, if the desired solution has not been reached, the change of modularity is done and the framework proceeds to Neuron level encoding in Stage 2. The best individuals with their fitness are transferred to the

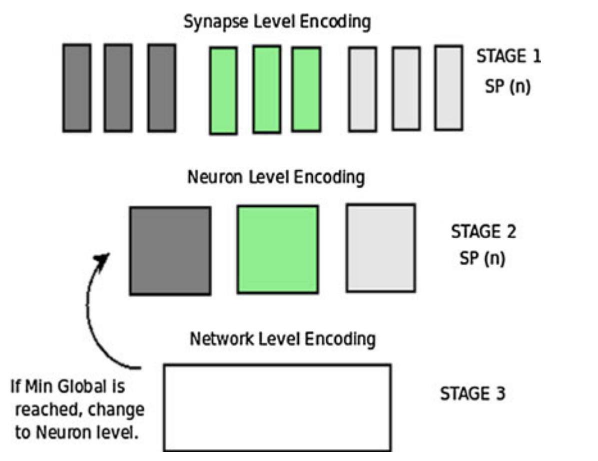


Fig. 4 The 3 Stage AMCC framework. The figure shows how the framework transforms the evolutionary process with different levels of encoding. The sub-populations "SP (n)" at Synapse level and Neuron level are shown. Note that in Stage 3, if the problem is not solved before the global minimum time in terms of function evaluations is reached, then the modularity is changed from Network to Neuron level

sub-populations of the Neuron level at Stage 2. All the sub-populations are cooperatively evaluated. During Neuron level evolution, the evolution proceeds and the change of modularity is done to proceed to Network level encoding at Stage 3. The best individuals at Neuron level from each sub-population are concatenated and added to the population at 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 them and then the evolution proceeds.

Note that in Stage 3, if the minimum number of function evaluations (MinGlobal) is reached, then the framework assumes that there is a convergence in local minimum. Therefore, the modularity is changed to Neuron level to employ 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 framework in Fig. 4 which gives a visualisation of how the framework transforms the evolutionary process with different levels of encoding.

4.1 Cooperative evaluation of subcomponents

A major concern in the general AMCC framework is the cooperative evaluation of each subcomponent in every sub-population. There are two main phases of evolution in the cooperative coevolution framework. The first is the *initialisation phase* and the second is the *evolution phase*.

In the initialization stage, the individuals in the sub-populations do not have a fitness. In order to cooperatively evaluate the ith individual of the kth subcomponent (chosen individual) in the initialization phase, arbitrary individuals from the rest of the sub-populations are selected and combined with the chosen individual and cooperatively evaluated. Once the fitness has been assigned to all the individuals of a particular sub-population, then the best individual can be chosen. A similar approach is used in (Potter and De Jong 2000; Chandra et al. 2011b).

We evaluate the cost in terms of the number of function evaluations for each level of encoding during initialization as follows.

1. *Synapse level encoding* Synapse $= ((I * H) + (H * O) + (H * H) + H + O) * P$ 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 framework will include the cost of functional evaluations during initialization in all three stages.

AMCC $=$ Synapse $+$ Neuron $+$ Network

In the evolution phase, cooperative evaluation is done by combining or concatenating the chosen individual from a sub-population with the best individuals from the rest of the sub-populations. After the initialization phase, 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 recurrent neural network and the fitness is calculated through the training error. The fitness evaluation of individuals in each sub-population 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.

4.2 Transfer of valuable information

The transition from one level of modularity to another should ensure that the information gained using the existing modularity is transferred to the next level of encoding. In the AMCC framework, 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 Synapse and Neuron level to be concatenated into a single chromosome.

Note that the number of sub-populations in each level are different. The Synapse level encoding has more sub-populations than the Neuron level, however, information

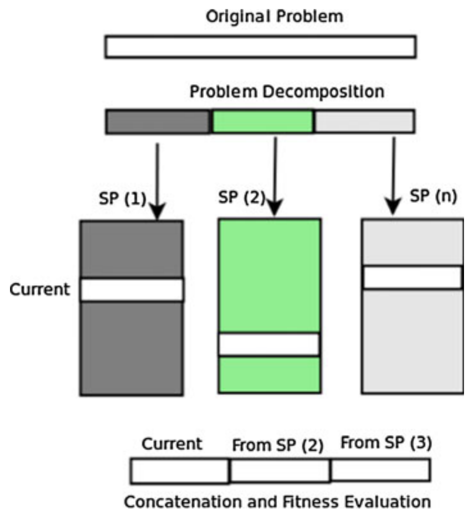


Fig. 5 The current individual whose fitness has to be evaluated is joined with arbitrary individuals from the rest of the sub-populations during cooperative evaluation in the initialization phase. In the evolution phase, the best individuals from the rest of the sub-populations are chosen. The current individual is then concatenated with the chosen individuals. The concatenated individual is encoded into the recurrent neural network by the given cooperative co-evolutionary encoding scheme. The fitness is then evaluated and assigned to the current individual (Chandra et al. 2011)

from the best individuals at 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
   In the transformation from 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 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 Network level. Consequently, the best solution obtained from the Neuron level encoding is transferred and retained in the Network level encoding using elitism. Note that it is important to retain the neural network encoding positions of all the synapses during the transfer. Information will be lost if the synapses are placed in the wrong position.

4.3 A heuristic to change modularity

The general idea in changing the 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 low improvement is made, then a change in modularity will be done.

α and β are two small constants which act as a threshold and determine how much of difference is acceptable, and if these thresholds are passed, then change in modularity is enforced. α and β can be chosen empirically with trial experiments. α will be usually bigger than β as there are greater differences in change of error during the earlier stages of evolution.

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

```
    $ChangeError = Error[current - n] - Error[current]$
If($ChangeError < Alpha$)
Set $ChangeMod$ to TRUE
```

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 in previous time steps the error has to be captured and deducted from the current error.

This heuristic is used in the AMCC framework shown in Algorithm 1. Once ChangeMod is TRUE, the modularity is changed from Synapse level to Neuron level. Similar approach is used for Neuron level to Network level where β is used as the threshold. Note that the AMCC framework 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 can deteriorate if more levels are used than required.

4.4 Evaluation of AMCC

The ability of an evolutionary algorithm to reach the desired solution within a specified time is an important measure of its performance. 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 number of function evaluations with the number of successful runs can be used for comparison. The least number of function evaluations with maximum success is desired.

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. In this work, we will also check the performance of AMCC 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 and adds up to the overall problem size.

4.5 G3-PCX evolutionary algorithm

The AMCC framework employs the generalised generation gap with parent centric crossover operator (G3-PCX) evolutionary algorithm (Deb et al. 2002). Note that any evolutionary algorithm could be used; however, we choose the G3-PCX because it requires fewer parameters to be optimised. The G3-PCX algorithm does not use a mutation operator and its crossover operator is based on the parent-centric mechanism which has mutation like features.

In G3-PCX, the whole population is randomly initialised and evaluated similarly to the standard genetic algorithm. The difference lies in the optimisation phase where a small sub-population is made of few chosen parents and children. At each generation, only the sub-population is evaluated rather than the whole population as in a standard genetic algorithm. The children with their new fitness become part of the bigger population. The best individual in the population is retained at each generation. The parent centric crossover operator is used for creating an offspring based on orthogonal distance between the parents. The parents are made of female and male components. 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 a 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 an offspring near the female than anywhere else in the space.

An 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. They have been used as a hill-climbing local search procedure in a memetic-based algorithm (Lozano et al. 2004). There is no mutation operator in the original G3-PCX algorithm. Amendments to the original algorithm have been done by proposing a mutation operator (Teo et al. 2007) which has shown good performance in multi-modal problems.

5 Simulation and analysis

This section presents an empirical study of the proposed AMCC framework and compares it with Neuron level (NL) (Chandra et al. 2011b) and Synapse Level (SL) (Gomez et al. 2008) encodings for training recurrent neural networks on grammatical inference problems.

The same evolutionary algorithm (Deb et al. 2002) is used in SL, NL and AMCC framework for comparison. A population size of 100, a pool size of 2 offspring and a family size of 2 parents are used. This set-up has been used in (Deb et al. 2002) and (Chandra et al. 2010, 2011b).

The individuals in the respective populations are initialised with random real numbers in the range of $[-5, 5]$ in all experiments. The Elman recurrent network (Elman 1990) is used in all experiments.

The data for the grammatical inference problems are generated as follows. We generate a training dataset by presenting strings of increasing lengths of 1–7 to the FFA shown in Fig. 3 and record the corresponding output for each string. Note that for every string length, all the possible bits are generated. Therefore, we have a training set of 255 samples.

The recurrent neural 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. Note that due to the nature of the FFA, this problem is class unbalanced.

The Tomita grammar is also used where the training data are generated by presenting random strings of length 15–25. Tomita 1–Tomita 4 (T1, T2, T3 and T4) languages from the Tomita grammar shown in Fig. 2 are used. The training set contains 125 positive and 125 negative strings. The RNN topology for the Tomita grammar has one neuron in the input layer and two neurons in the output layer representing the accepting and rejecting states.

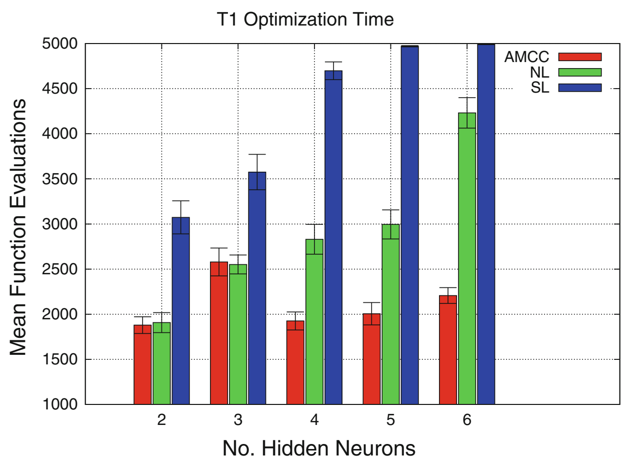
In all problems, the RNN is trained until the mean-squared-error (MSE) has reached below 0.0005. The training terminates if the number of function evaluation exceeds the maximum. The maximum number of function evaluations for T1 and T2 is 5,000. T3, T4 and FFA problem use a maximum of 10,000 function evaluations. These values have been chosen in trial experiments.

The goal of this paper is to evaluate AMCC on the training performance and hence, generalisation performance is not given.

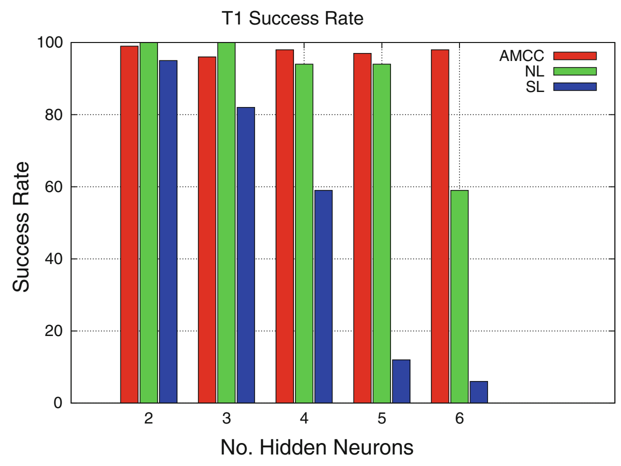
5.1 Comparison of AMCC with Synapse and Neuron level

This section compares the performance of the AMCC framework with Neuron and Synapse level encoding. The AMCC employs the heuristic to change of modularity where $\alpha = 0.05$ and $\beta = 0.01$. These are checked for previous $n = 5$ time steps.

The goal is to observe the performance of the respective algorithms in different network topologies given by the 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 enough

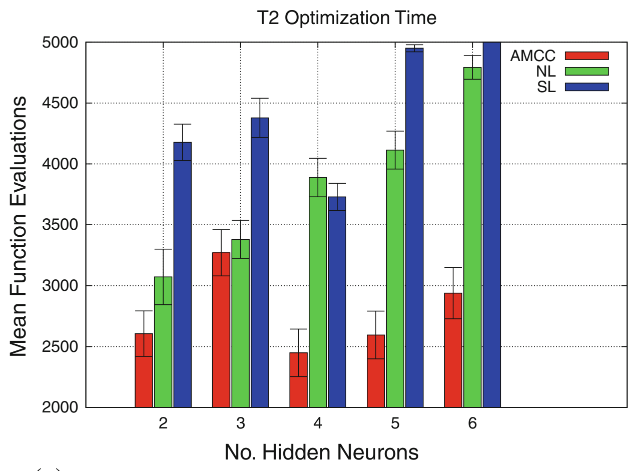


(a) Optimization time in terms of the number of function evaluations

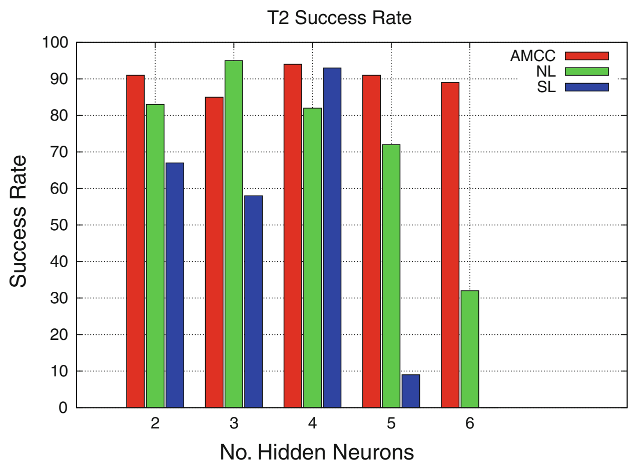


(b) Success rate in the T1 problem

Fig. 6 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

Fig. 7 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**

neurons are not present in the hidden layer. It is also difficult for evolutionary algorithms to optimise a problem when their chromosome size increases according to the number of hidden neurons in the hidden layer.

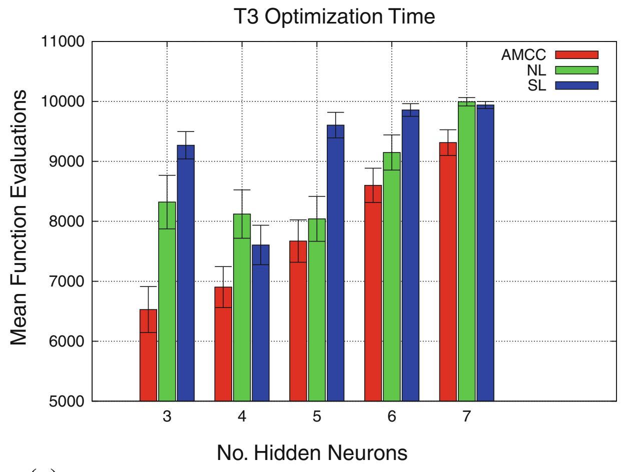
Figures 6, 7, 8, 9 and 10 show the performance of each algorithm in relation to the number of hidden neurons. The number of successful runs out of 100 experiments is shown in part (a) while the number of average function evaluations is shown in part (b) of the respective figures. The goal of AMCC is to obtain maximum success with the minimum number of average function evaluations. Note that the average function evaluations consists of both successful and unsuccessful runs.

In Figs. 6 and 7, for the T1 and T2 problems, the performance of AMCC is consistent as the number of hidden neurons changes in terms of least number of function evaluations (a) and best success rates (b). The performance of SL and NL is good only for a certain number of hidden neurons and deteriorates otherwise.

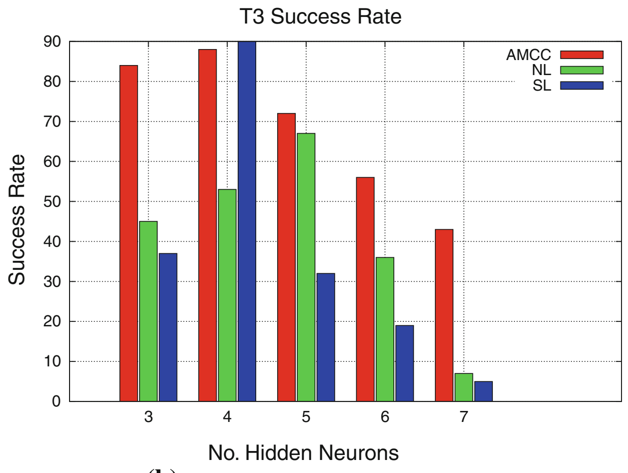
In Fig. 8, for the T3 problem, AMCC outperforms SL and NL for all the different number of hidden neurons in terms of least function evaluations (a) and best success rates (b). In the T4 problem given in Fig. 9, the performance of NL is slightly better than AMCC for 3, 4 and 5 neurons. AMCC outperforms NL for 5 and 6 neurons.

In the FFA problem given in Fig. 10, AMCC outperforms NL in all 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 a certain number of hidden neurons. The results have shown that the AMCC framework scales better than NL and SL when the size of the problem increases.

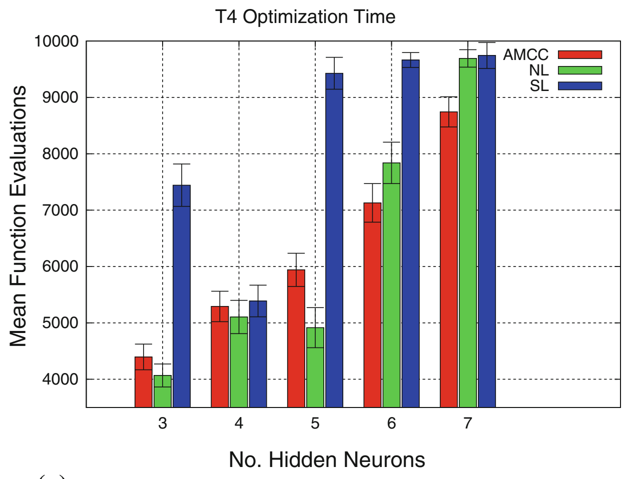


(a) Optimization time in terms of the number of function evaluations

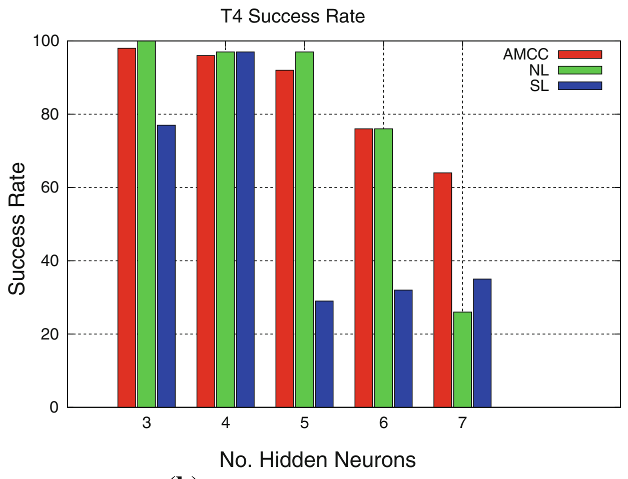


(b) Success rate in the T3 problem

Fig. 8 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



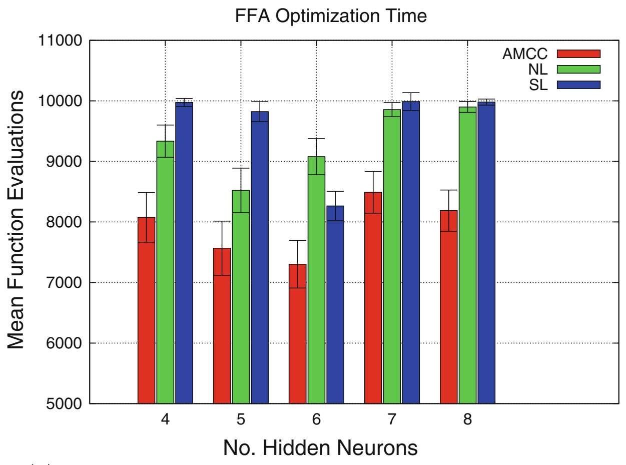
(b) Success rate in the T4 problem

Fig. 9 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**

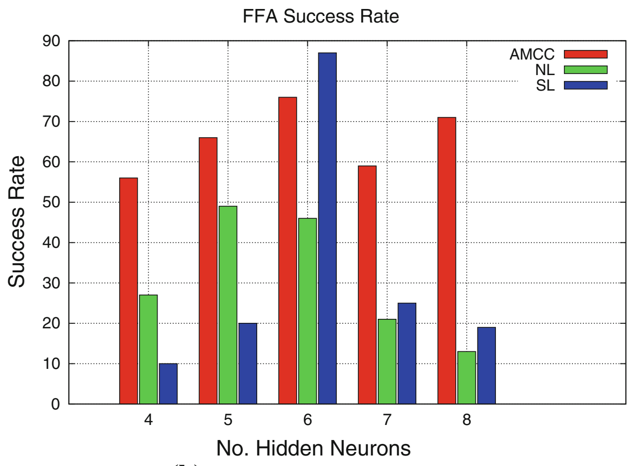
5.1.1 Discussion

The results have shown that the nature of the problem changes while training recurrent neural network, i.e., the degree of non-separability increases. The problem has shown to be separable initially (where Synapse level encoding helps) and later, the degree of non-separability increases where network level encoding with one population is more effective. The improved performance of the AMCC framework in comparison with Neuron and Synapse level gives justification that the degree of non-separability changes during evolution. Neuron and Synapse level encoding failed to deliver the desired solution 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 Neuron and Synapse level encoding do not scale well as AMCC when the size of the problem increases due to the number of hidden neurons.

In neural network training, the synapses in cooperation contribute to the overall error of the entire network. It 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 two main issues of concern are (1) global and local search and (2) degree of non-separability. In most problems, a global search is needed in the beginning and refinement using local search is done during the final stages. Different evolutionary algorithms balance the intensity of the global-local search using different forms of crossover and mutation operators. In memetic evolutionary approaches, additional local search techniques are used to balance with global exploration (Smith 2007; Ong et al. 2006).



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



(b) Success rate in the FFA problem

Fig. 10 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**

The second issue is the balance in the degree of the non-separability.

The AMCC framework utilizes the strengths of both synapse, neuron, and network level encoding. This gives it the flexibility to apply a particular level of encoding 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 and deliver evolutionary search which relates to the different degrees of non-separability. In the initial stages, the neural network training problem has 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 neuron level and network level encoding perform well when needed.

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

6 Conclusions and future work

This paper introduced a novel cooperative co-evolution framework for adapting modularity during evolution. The framework provided a better understanding on the degree of non-separability for training recurrent neural networks.

In evolution of recurrent neural networks, the nature of the problem changes as the problem is being learnt. The nature of the problem changes in terms of the degree of non-separability which reflects on the interacting variables. During the later stages of evolution, the interaction between variables increases and the problem becomes more non-separable which requires bigger sub-populations. Towards the end of evolution, a single population is needed. This is the main reason that standard cooperative co-evolution problem decomposition methods like Neuron level and Synapse level encodings failed to perform better as they had a fixed problem decomposition method through the entire evolution process. The AMCC framework can adapt according to the different degrees of non-separability which delivered improved performance when compared to Neuron and Synapse level encoding as the size of the network increased.

In future, it would be interesting to apply AMCC for training recurrent networks for real-world applications such and control and time series prediction.

Acknowledgments The author expresses his sincere gratitude towards Mohammad Omidvar for an earlier discussion on the subject of this paper.

References

Blanco A, Delgado M, Pegalajar MC (2001) A real-coded genetic algorithm for training recurrent neural networks. Neural Netw 14(1):93–105

Brookshear JG (1989) Theory of computation: formal languages, automata, and complexity. Benjamin-Cummings Publishing Co., Inc., Redwood City

Castao MA, Vidal E, Casacuberta F (1995) Finite state automata and connectionist machines: a survey. In: Mira J, Hernndez FS (eds) IWANN. Lecture Notes in Computer Science. Springer, New York, pp 433–440

Chandra R, Frean M, Zhang M (2010) An encoding scheme for cooperative coevolutionary feedforward neural networks. In: Li J (ed) AI 2010: advances in artificial intelligence. Lecture Notes in Computer Science, vol 6464. Springer, Berlin, pp 253–262

Chandra R, Frean M, Zhang M (2011a) Modularity adaptation in cooperative coevolutionary feedforward neural networks. In: International joint conference on neural networks, pp 681–688

Chandra R, Frean M, Zhang M, Omlin CW (2011b) Encoding subcomponents in cooperative co-evolutionary recurrent neural networks. Neurocomputing 74(17):3223–3234

Chandra R, Omlin CW (2006) Training and extraction of fuzzy finite state automata in recurrent neural networks. In: Proceedings of international conference on computational intelligence, pp 274–279

Chen W, Weise T, Yang Z, Tang K (2010) 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. PPSN'10, pp 300–309

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

Elman JL (1990) Finding structure in time. Cogn Sci 14:179–211

Gabrijel I, Dobnikar A (2003) On-line identification and reconstruction of finite automata with generalized recurrent neural networks. Neural Netw 16(1):101–120

García-Pedrajas N, Ortiz-Boyer D (2007) A cooperative constructive method for neural networks for pattern recognition. Pattern Recogn 40(1):80–98

Giles CL, Lawrence S, Tsoi A (1997) Rule inference for financial prediction using recurrent neural networks. In: Proceedings of the conference on computational intelligence for financial engineering, pp 253–259

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

Gomez F, Mikkulainen R (1997) Incremental evolution of complex general behavior. Adapt Behav 5(3–4):317–342

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

Gomez FJ (2003) Robust non-linear control through neuroevolution. Technical Report AI-TR-03-303, PhD thesis, Department of Computer Science, The University of Texas at Austin

Haykin S, Principe J, Sejnowski T, McWhirter J (2006) New directions in statistical signal processing: from systems to brain. MIT Press, Cambridge

Kolen J, Kremer S (2001) A field guide to dynamical recurrent networks. IEEE Press, New Jersey

Li X, Yao X (2011) Cooperatively coevolving particle swarms for large scale optimization. IEEE Trans Evol Comput (in press)

Liu Y, Yao X, Zhao Q, Higuchi T (2001) Scaling up fast evolutionary programming with cooperative coevolution. In: Proceedings of the 2001 Congress on Evolutionary Computation, pp 1101–1108

Lozano M, Herrera F, Krasnogor N, Molina D (2004) Real-coded memetic algorithms with crossover hill-climbing. Evol Comput 12(3):273–302

Manolios P, Fanelli R (1994) First-order recurrent neural networks and deterministic finite state automata. Neural Comput 6(6): 1155–1173

Marakami K, Taguchi H (1991) Gesture recognition using recurrent neural networks. In: Proceedings of the SIGCHI conference on Human factors in computing systems: reaching through technology, pp 237–242

Medsker L, Jain L (1999) Recurrent neural networks: design and application, Computer Intelligence. CRC Press international

Omidvar M, Li X, Yao X (2010) Cooperative co-evolution for large scale optimization through more frequent random grouping. In: 2010 IEEE Congress on evolutionary computation (CEC), pp 1754–1761

Omidvar MN, Li X, Yao X (2011) Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms. In: Proceedings of the 13th annual conference on genetic and evolutionary computation. GECCO '11, pp 1115–1122

Omlin CW, Thornber KK, Giles CL (1998) Fuzzy finite state automata can be deterministically encoded into recurrent neural networks. IEEE Trans Fuzzy Syst 6:76–89

Ong Y-S, Lim M-H, Zhu N, Wong K-W (2006) Classification of adaptive memetic algorithms: a comparative study. IEEE Trans Syst ManCybern Part B Cybern 36(1):141–152

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

Potter MA, De Jong KA (2000) Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evol Comput 8(1):1–29

Potter MA, Jong KAD (1994) 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, London, UK, pp 249–257

Robinson T (1994) An application of recurrent nets to phone probability estimation. IEEE Trans Neural Netw 5:298–305

Salomon R (1996) 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):263–278

Schmidhuber J, Wierstra D, Gagliolo M, Gomez F (2007) Training recurrent networks by evolino. Neural Comput 19(3):757–779

Shi Y-j, Teng H-f, Li Z-Q (2005) Cooperative co-evolutionary differential evolution for function optimization. In: Wang L, Chen K, Ong YS (eds) Advances in natural computation. Lecture Notes in Computer Science, vol 3611. Springer, Berlin, pp 1080–1088

Smith J (2007) Coevolving memetic algorithms: a review and progress report. IEEE Trans Syst Man Cybern Part B Cybern 37(1):6–17

Teo J, Hijazi HA, Omar ZA, Mohamad NR, Hamid Y (2007) Harnessing mutational diversity at multiple levels for improving optimization accuracy in g3-pcx. In: IEEE Congress on Evolutionary Computation. IEEE, New Jersey, pp 4502–4507. http://dblp.uni-trier.de/db/conf/cec/cec2007.html#TeoHOMH07

Tomita M (1982) Dynamic construction of finite automata from examples using hill-climbing. In: Proceedings of the fourth annual cognitive science conference. Ann Arbor Michigan, pp 105–108

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

Watrous RL, Kuhn GM (1992) Induction of finite-state languages using second-order recurrent networks. Neural Comput 4(3): 406–414

Yang Z, Tang K, Yao X (2008a) Large scale evolutionary optimization using cooperative coevolution. Inf Sci 178(15):2985–2999

Yang Z, Tang K, Yao X (2008b) Multilevel cooperative coevolution for large scale optimization. In: IEEE congress on evolutionary computation, pp 1663–1670