B. Cooperative Coevolution for Neuroevolution

CC divides a large problem into smaller subcomponents, which are implemented as subpopulations that are evolved in isolation and cooperation takes place for fitness evaluation [1]. The subcomponents are also referred as modules. Problem decomposition determines how the problem is broken down into subcomponents. The size of a subcomponent and the way it is encoded depends on the problem. The original CC framework has been used for general function optimization and the problems were decomposed to its lowest level, where a separate subcomponent was used to represent each dimension of the problem [1]. It was later found that this strategy is effective only for problems that are fully separable [2]. Much work has been done in the use of CC in large-scale function optimization, and the focus has been on nonseparable problems [2], [45]–[47].

A function of n variables is separable if it can be written as a sum of n functions with just one variable [48]. Nonseparable problems have interdependencies between variables as opposed to separable ones. Real-world problems mostly fall between fully separable and fully nonseparable. CC has been effective for separable problems. Evolutionary algorithms without any decomposition strategy appeal to fully nonseparable problems [4].

The subpopulations in CC 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 problem decomposition method has been able to group the interacting variables into separate subcomponents [7]. 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 that have fewer interactions among themselves [4].

CC methods have been used for neuroevolution of recurrent neural networks for time-series problems [11], [17], and it has been shown that they perform better when compared with several methods from literature.

C. Diversity in Cooperative Coevolution

Population diversity is a key issue in the performance of evolutionary algorithms. The diversity of a population affects the convergence of a evolutionary algorithm. A population, which consists of similar candidate solutions in the initial stages of the search, is prone to convergence in a local minimum. The selection pressure and recombination operations mainly affect the diversity of the population. Evolutionary operators, such as crossover and mutation, must ensure that the population is diverse enough in order to avoid local convergence. Diverse candidate solutions can ensure the algorithm to escape a local minimum. In evolutionary algorithms, diversity has been improved using techniques, such as: 1) complex population structures [49], [50]; 2) the use of specialized operators to control and assist the selection pressure [51]; 3) reintroduction of genetic materials in the population [52], [53]; and 4) diversity measures, such as the hamming distance [54], gene frequencies [70], and diversity measures to explore and exploit search [55].

CC naturally retains diversity through the use of subpopulations, where mating is restricted to the subpopulations and cooperation, is mainly by collaborative fitness evaluation [1], [56]. Since selection and recombination are restricted to a subpopulation, the new solution will not have features from the rest of the subpopulations; therefore, CC produces more diverse population when compared with a standard evolutionary algorithm with a single population.

D. Recurrent Neural Networks for Time-Series Prediction

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. First-order recurrent neural networks use context units to store the output of the state neurons from computation of the previous time steps. The context layer is used for computation of present states as they contain information about the previous states. The Elman architecture [18] employs a context layer, which makes a copy of the hidden layer outputs in the previous time steps. The dynamics of the change of hidden state neuron activation's in Elman style recurrent networks are given by

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

where $y_k(t)$ and $x_j(t)$ represent the output of the context state neuron and input neurons, respectively. v_{ik} and w_{ij} represent their corresponding weights. $f(.)$ is a sigmoid transfer function.

In order to use neural networks for time-series prediction, the time-series data need to be preprocessed and reconstructed into a state space vector [57]. Given an observed time series $x(t)$, an embedded phase space $Y(t) = [(x(t), x(t-T), \ldots, x(t(D-1)T)]$ can be generated, where T is the time delay, D is the embedding dimension, $t = 0, 1, 2, \ldots, N - DT - 1$, and N is the length of the original time series [57]. Taken's theorem expresses that the vector series reproduces many important characteristics of the original time series. The right values for D and T must be chosen in order to efficiently apply Taken's theorem [58]. Taken's proved that if the original attractor is of dimension d, then $D = 2d + 1$ will be sufficient to reconstruct the attractor [57].

The reconstructed vector is used to train the recurrent network for one-step-ahead prediction where one neuron is used in the input and the output layer. The recurrent network unfolds k steps in time, which is equal to the embedding dimension D [11], [28], [30].

Either the root-mean-squared error (RMSE) or the normalized mean-squared error (NMSE) can be used to measure the prediction performance of the given recurrent neural network.

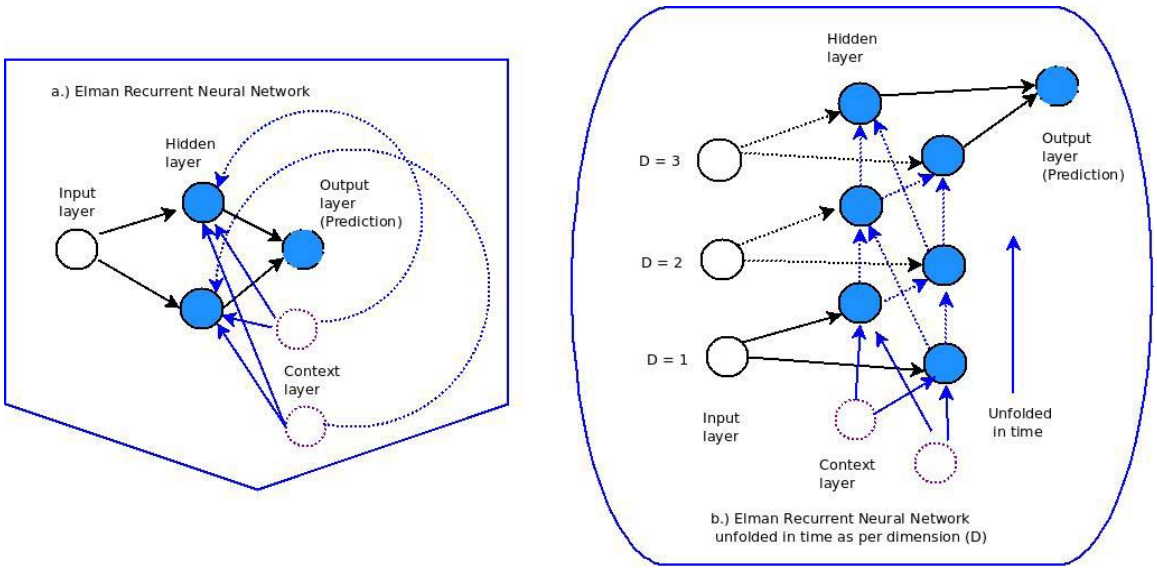

Fig. 1. Elman recurrent neural network used for time-series prediction. Note that only one neuron is used in the input and output layer. The number of hidden neurons varies as per application. The network unfolds in time according to the size of the dimension (D) from using Taken's theorem in obtaining state space vector from the time series. Solid lines (synapses): trainable weights that are evolved using the proposed competitive coevolution algorithm.

These are given in

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

$$\text{NMSE} = \left(\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}\right) \tag{3}$$

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

Elman recurrent neural networks given in (1) are shown in Fig. 1, where the network is given along with how it is unfolded through time according to the dimension (D). This approach has been used in our previous work [11]. We use a fixed dimension size; however, the architecture can also use the dimension size that varies for different points in the time series.

E. Problem Decomposition for Recurrent Networks

Problem decomposition is an important procedure in using CC for neuroevolution. The problem decomposition method will determine which set of weights from the neural network will be encoded into a particular subpopulation of CC. In the case of recurrent neural networks, special consideration needs to be made for the weights that are associated with the feedback connections.

There are two major problem decomposition methods for neuroevolution that decompose the network on the NL and SL. In SL problem decomposition, the neural network is decomposed to its lowest level, where each weight connection (synapse) forms a subcomponent. Examples include cooperatively coevolved synapse neuroevolution [5] and neural fuzzy network with cultural cooperative particle swarm optimization (CPSO) [10]. In NL problem decomposition, the neurons in the network act as the reference point for the decomposition. Examples include enforced subpopulations [8], [9] and neuron-based subpopulation [6], [7].

III. Competition and Collaboration in Cooperative Coevolution

Collaboration in an environment of limited resources is an important feature used for survival in nature. Collaboration helps in the sharing of resources between the different species that have different characteristics for adaption when given with environmental changes and other challenges. In CC, the species are implemented as subpopulations that do not exchange genetic material with other subpopulations. Collaborations and exchange of genetic material or information between the subpopulations can be helpful in the evolutionary process. Competition and collaboration are vital component of evolution where different groups of species compete for resources in the same environment. Different types of problem decomposition methods in CC represent different groups of species (neural and SL [5]–[7]) in an environment that features collaboration through fitness evaluation during evolution.

In this section, we propose a CC method that incorporates competition and collaboration with species that is motivated by evolution in nature. The proposed method employs the strength of a different problem decomposition method that reflects on the different degree of nonseparability (interaction of variables) and diversity (number of subpopulations) during evolution [4].

The proposed method is called competitive island-based CC (CICC), which employs different problem decomposition methods that compete with different features they have in terms of diversity and degree of nonseparability. In the rest