DE GRUYTER OPEN

JAISCR, 2018, Vol. 8, No. 2, pp. 133 – 154
10.1515/jaiscr-2018-0010

LEARNING STRUCTURES OF CONCEPTUAL MODELS FROM OBSERVED DYNAMICS USING EVOLUTIONARY ECHO STATE NETWORKS

Hassan Abdelbari and Kamran Shafi

[1] *School of Engineering and Information Technology,*
University of New South Wales at Canberra,
Northcott Drive, Campbell, ACT, 2600, Australia,
Email: hassan.abdelbari@student.adfa.edu.au, k.shafi@adfa.edu.au

Submitted: 4th March 2017; Accepted: 29th March 2017

Abstract

Conceptual or explanatory models are a key element in the process of complex system modelling. They not only provide an intuitive way for modellers to comprehend and scope the complex phenomena under investigation through an abstract representation but also pave the way for the later development of detailed and higher-resolution simulation models. An evolutionary echo state network-based method for supporting the development of such models, which can help to expedite the generation of alternative models for explaining the underlying phenomena and potentially reduce the manual effort required, is proposed. It relies on a customised echo state neural network for learning sparse conceptual model representations from the observed data. In this paper, three evolutionary algorithms, a genetic algorithm, differential evolution and particle swarm optimisation are applied to optimize the network design in order to improve model learning. The proposed methodology is tested on four examples of problems that represent complex system models in the economic, ecological and physical domains. The empirical analysis shows that the proposed technique can learn models which are both sparse and effective for generating the output that matches the observed behaviour.

Keywords: Complex systems modelling, Conceptual models, Causal loop diagrams, Computational intelligence, Echo state networks, Evolutionary algorithms

1 Introduction

Many approaches for modelling a complex system involve developing high-level conceptual models, also referred to as mental or explanatory, as an initial step in the modelling process in order to map the interdependencies among the system's components. Such models often work as intermediate representations which lead to the development of final detailed models that, in turn, can be used to develop simulations that mimic the system's behaviour; for instance, in system dynamics (SD) modelling [1], the conceptual modelling step is represented using causal loop diagrams (CLDs) which are later converted into stock and flow diagrams (SFDs) to build a SD simulation. Similarly, in agent-based modelling (ABM), a unified modelling language (UML) is commonly used as an initial step toward implementing the behaviours of the agents [2].

There are several representations for developing conceptual models, including CLDs [1], influence diagrams [3], the business process modelling notation (BPMN) [4], event graphs [5], process flow diagrams [6], simulation activity diagrams [7] and

UML, to name a few. Choosing an appropriate one depends mainly on the modelling approach used. In particular, CLDs are considered powerful tools for developing such models due to their capability to represent visual hypotheses about the dynamics of a systems using feedback loops and time delays.

However, as the process for building these conceptual models is generally very manual, it involves intensive amount of time, effort and human resources. Although there is no easy alternative to process, recent advances in computational intelligence and machine learning techniques provide promising means of ameliorating some of the tedious tasks, including data analysis, knowledge extraction, model generation, and the testing of a large number of models in a short time [8–10]. In addition to saving on resources, integrating such approaches for modelling support can help to complement human cognitive abilities that have proven to have limitations when dealing with large amounts of data and exploring large model spaces [11].

In a recent work, the authors investigated using of echo state networks (ESNs) [12] to learn CLD-like models [13] by training the network on observational data. Our basic motivation for exploring the use of ESNs for this purpose emanated from observing a number of similarities between the architectures of an ESN reservoir (see Section 2.2 for details) and a CLD, such as the use of nodes, directed connections with polarities and feedback loops. Extensive experimentation using a number of variants of ESNs for several case studies showed that an ESN can be adopted to learn matching CLD-like structures from the given data representing a system's output behaviour. In a sense, this enables simulations of learned causal models and the generation of system behaviour. This additional feature is quite useful as it allows the accuracy of a learned conceptual model to be tested at an early stage without the need to convert it into a full-scale simulation. To achieve the above, several modifications to the standard ESN architecture, including removing a number of links between different layers that only contribute to network complexity rather than improve the learning performance, restricting the number of ESN reservoir neurons used according to the number of system variables and reducing the output nodes to the number of output variables, have been proposed.

The work presented in this paper builds on the above studies and explores the use of different evolutionary algorithms (EAs) to further optimize the design of an ESN in order to improve its learning performance in terms of matching both the structure and output of the target model. Three different evolutionary optimisation methods, a genetic algorithm (GA) [14], differential evolution (DE) [15] and particle swarm optimisation (PSO) [16], are applied to optimize the design of an ESN's parameters and weights. A fitness function that takes into account both the model's complexity and its output error as well as a varying penalty term for handling infeasible solutions are designed (see Section 3.2.2 for details). The evolutionary ESN methodology for learning sparse and interpretable models is tested on four problems that represent examples of complex system models selected from different business, ecological and physical domains. They have varying degrees of structural and output complexity to demonstrate the domain-independence of the applied approach.

A thorough analysis of the experimental results highlights the trade-off between the output and structural errors of the learned models as well as convergence properties of the evolutionary process. This empirical analysis demonstrates that the evolved ESNs perform better than conventional non-evolutionary ones in terms of matching both the target model's structure and observed behaviour.

Overall, this work relates to the broader field of model learning and identification for complex systems. It includes a large body of research focusing mainly on representing complex system dynamics as mathematical relationships, such as ordinary differential equations [17], partial differential equations [18] and difference equations [19]. These approaches are most commonly applied in the engineering and physical science domains [20]. Generally, mathematical modelling approaches rely on making a number of assumptions and simplifications that can significantly affect a model's fidelity and adopting complex functional forms that limit their capbility to provide a logical explanation of a system's behaviour. Computational or machine learning-based approaches, such as artificial neural networks [21], evolutionary computation [22] and fuzzy systems [23], rely on direct model learning from system observations. Generally, they are

quite powerful in terms of their capability to model complex and non-linear behaviours. However, their learned models are often black box and have low interpretability. Other more process-based or systematic approaches, such as SD and ABM, involve several modelling steps and rely on subject matter experts being heavily involved in studying the underlying system, collecting relevant data, identifying the system variables and so on. Such approaches often produce highly interpretable and logical models but can be quite resource intensive in terms of both time and effort. They are also more prone to subjective bias and judgmental error due to their nature. Our proposed approach is essentially computational but could be positioned to complement both human input into the development process for models and black box models.

The rest of this paper is organised as follows: Section 2 provides the necessary background information on conceptual models and ESNs. Section 3 describes the proposed methodology applied to learn conceptual models. Section 4 discusses the experimental settings and examples of the complex system models used to evaluate the proposed methodology. Section 5 presents and analyses the empirical results, and Section 6 concludes the paper.

2 Background

2.1 Conceptual Models

Conceptual models are defined as high-level abstract representations of a real-world system or phenomenon developed by domain experts that capture system components and their interrelationships [24]. A typical one consists of two main components: system variables; and links between those that represent a certain type of relationship. Conceptual models can be represented in different ways (e.g., CLDs, UMLs, influence diagrams). Specifically, a CLD [25] is a formal method for representing conceptual models and is commonly used in SD modelling approach. It describes the relationships among system variables as directed links with negative or positive polarities that indicate direct or relative causal changes between two variables, respectively. A feedback loop in a CLD refers to a closed chain of causal relationships and delays. In general, conceptual models and, in particular, CLDs, cannot be simulated directly but need to be converted to a formal model to run a simulation and generate system behaviour.

An example of a conceptual model represented as a CLD is shown in Figure 1, where *skills level* and *earned income* are examples of systems variables, the connection between *transportation network* and *drivers of economy* a causal link and the connection between *educational level* and *skills level* a feedback loop. This model describes a regional development system in the Atlantic region in Canada where the *education level* and *diversification of economy* are the main driving forces affecting its system [26]. It shows that the relationship between *diversification of economy* and *economic prosperity*, indicating that the system should reject a mono-industrial economy and encourage other sectors, such as education, and innovations in technology. On the other hand, the relationship between *educational level* and *population retention* shows that an increase in the former decreases the later as young workers tend to immigrate to main cities.

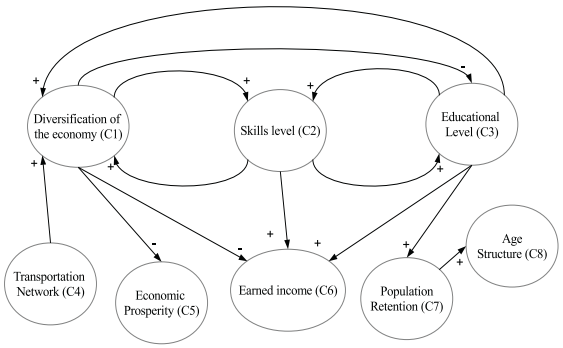


Figure 1. Conceptual model represented as CLD

2.2 Echo State Networks (ESNs)

An ESN is a special type of recurrent neural network (RNN) with cycles and feedback loops in the connections among its different network layers in contrast to the feedforward neural network that has direct connections from each layer to the next without any cycle or feedback. These cycles enable RNNs to more naturally model temporal data and non-linear dynamical systems [27].

A typical ESN consists of three layers: input; hidden (dynamic reservoir); and output. An example of an ESN standard architecture with input

layer of size K, dynamic reservoir of size N and output layer of size L is shown in Figure 2. The input layer is connected to the dynamic reservoir, and could be connected to the output layer, through the input weight matrix $W_{in} \in \mathbb{R}^{N\times K}$. The dynamic reservoir's neurons are connected with each other through the reservoir weight matrix $W \in \mathbb{R}^{N\times N}$ and also to the output layer through the output weight matrix $W_{out} \in \mathbb{R}^{L\times(N+K)}$. Finally, the output layer could be connected to the reservoir layer through the feedback weight matrix $W_{back} \in \mathbb{R}^{N\times L}$ [12].

The main difference between ESNs and RNNs resides in their design and training procedures. In a typical RNN, although all the weights are adjusted during training, due to its feedback loops and cycles, the training suffers from problems such as poor convergence and computationally expensive parameters updates [28].

The power of an ESN lies in its straightforward design and training procedure for network weights all of which, except W_{out}, are fixed with only W_{out} trained using a relatively simple procedure. Both W_{in} and W_{back} are initialised randomly with a scale value of δ and W designed using a few steps to ensure that the network maintains an echo state property. This property ensures that, if the network runs for a very long time, its state will be uniquely identified by its historical input/output signals and, if a new input signal is presented to it, it will be capable of generating a suitable corresponding output one [29]. Mathematically, the echo state property is connected to the algebraic properties of W, the spectral radius of which should not exceed unity [30], and can be defined as follows:

Definition 2.1. Echo State Property. An ESN network $N : X\times U \longrightarrow X$ has an echo state property with respect to the input signals set U if, for any left infinite input sequence $u^{-\infty} \in U^{-\infty}$ and any two state vector sequences belonging to network state's set X $x^{-\infty}, y^{-\infty} \in X^{-\infty}$ and compatible with $u^{-\infty}$, it holds that $x_0 = y_0$.

In order to design W to have this property, a weight matrix W_0 is generated randomly $\in [-1,1]$ with a given connectivity probability parameter ρ. Then, the maximum eigenvalue of W_0 is calculated and W_0 divided by this value to build a weight matrix W_1 which, finally, is multiplied by a chosen value called spectral radius $\alpha \in [0,1[$. After initialising all network weights, except the output ones, the network reservoir states are updated according to the following equation

$$x(t+1) = (1-\gamma)x(t) + \gamma f(W_{in}u(t+1) + Wx(t) + W_{back}y(t)), \quad (1)$$

where $x(t+1)$ and $x(t)$ refer to the activation values of the reservoir's neurons at times $t+1$ and t, respectively, where all initial values of the reservoir's neurons are set to zero, $u(t+1)$ the external input signal at time $t+1$, $y(t)$ the network output at time t, $f(.)$ the activation function which could be *sigmoid* or $tanh$ and γ the leaking rate parameter which controls the speed of the reservoir's dynamics [31]. This equation is applied for the number of steps equal to the size of the training data set and the reservoir's neuron states are collected in a matrix M, before which their first initial states are ignored. The number of these discarded steps is called the washout size.

After collecting the reservoir states in matrix M, the output weights are trained by a straightforward step as

$$W_{out} = M^{-1}Y^{target}, \quad (2)$$

where Y^{target} is the desired output from the training data set and M^{-1} the morse-pseudo inverse for states matrix M.

After determining the output weights, the network's output is calculated as

$$y(t) = f(x(t)W_{out}), \quad (3)$$

where $y(t)$ is the output at time t and $f()$ the activation function that could be *sigmoid*, $tanh$ or *linear*.

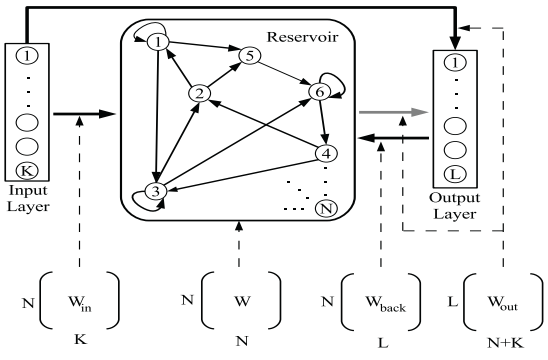


Figure 2. Standard ESN architecture

3 Learning Conceptual Models Using Evolutionary Echo State Networks

The proposed methodology for learning conceptual models from observable outputs involves three main steps: customising an ESN so that its reservoir can be used to represent conceptual models; optimising as ESN's design parameters using the given EAs; and evaluating the learned models. Figure 3 depicts the overall process for model learning. The data collection step involves identifying the key system variables from the system of interest and is the only, and critical, input required to encode the ESN. A customised ESN is set up using the given number of system variables with an EA employed to search for the design parameters of this ESN that learn by minimising the error with the target outputs collected from system observations. Finally, the structural error of the best model learned is computed by comparing it with a known or expert-driven one. Note that a known model structure is not used to train the network but only to evaluate the goodness of the model learned by the ESN. The process of customising the ESN and evolving its design using EAs is further elaborated below.

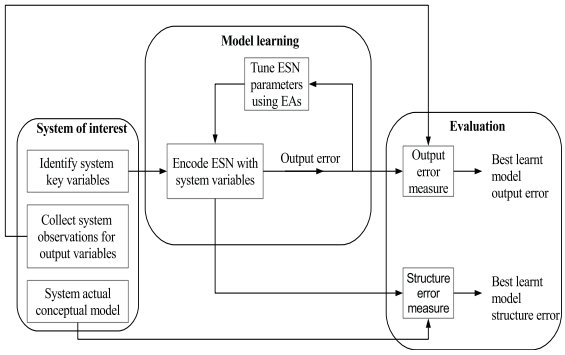


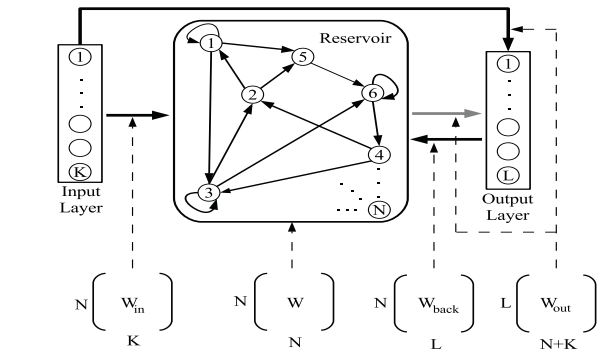
Figure 3. Overview of conceptual model learning process using evolutionary ESNs

3.1 ESN Modifications

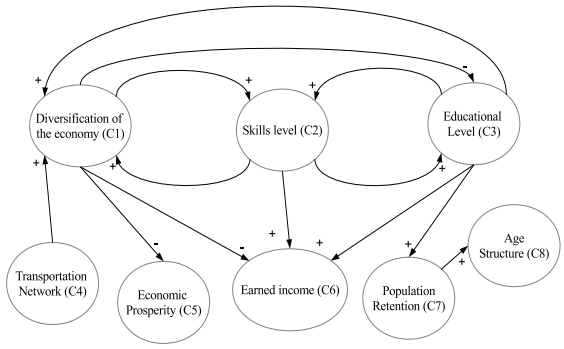
Several ESN variants based on different network architectures were recently investigated [13] and their performances in terms of learning target causal models and outputs were compared with those of the standard ESN. In summary, the extensive empirical analysis conducted in the above work showed that, under given assumptions (as discussed below), an ESN encoded with the same number of reservoir nodes as the number of system variables and same number of I/O nodes as the number of observable output system variables is not only able to produce the target output's behaviour but also learn sparse causal models closely match the target models structurally. The dynamic reservoir neurons are also labelled implicitly with the names of the system variables using an adjacency matrix convention. While ESN learning does not take into account any semantic information between variables, using such labelling allows us to impose this association indirectly and also use it to evaluate the closeness of the resultant models to the target CLD structure. This relates to our main assumption in adopting ESNs to represent causal structures, that is, the inclusiveness of system variables provided by the modeller are the only, and complete, set of variables that define the observed behaviour. This assumption is important for preserving the implicit mapping imposed in our setup between system variables and network nodes, as explained above. It could be seen as forcing the network to overfit the data using the given number of variables. However, it should note that our goal is to learn the most plausible structures and not predict future values. Nonetheless, we use the standard cross-validation process to train the network which shows that the generalisation is maintained and the error is minimised over the test data.

Other modifications include the removal of direct links between the input and output neurons, output neurons to reservoir and self connections of the reservoir neurons. Most of these extra connections are considered optional and only marginally improve the network's performance, if at all. On the other hand, they considerably complicate the architecture and hinder the learning of a sparse model. Figure 4 shows the standard ESN and a customised one that represents the conceptual model example shown in Figure 4(b).

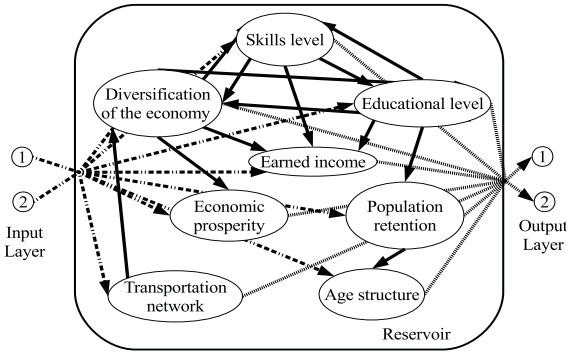
We build on these modifications and employ EAs to further tune the design of our customised ESN in order to improve its performance.



(a) Standard ESN architecture



(b) Example of concaptual model



(c) Customised ESN

Figure 4. Illustrative example of standard and customised ESN structures

3.2 Optimising ESN Design Parameters and Weights

ESNs have two sets of parameters that need to be identified, a weights set S_w and design parameters set S_d as

$$\begin{aligned} S_w &= \{W_{in}, W, W_{back}\}, \\ S_d &= \{\alpha, \delta, \gamma, \rho, N\}, \end{aligned} \tag{4}$$

where W_{in} is input weights matrix, W the reservoir weights matrix, W_{back} the feedback weights matrix, α the spectral radius of W, ρ the connectivity probability of W, δ the scale of weights for both W_{in} and W_{back}, γ the leaking rate used in the update equation for the network states and N the size of the dynamic reservoir neurons.

Since the training of network W_{out} is straightforward and fast, its design parameters can be found by trying different parameter combinations in S_d and training the network for each given large N value that can range from hundreds to thousands of neurons [31]. The method for doing this could be challenging and requires experience in tuning despite the guidance provided by researchers [28, 31]. Identifying ESN design parameters could be formulated as a search problem whereby an ESN model that minimises the error between the outputs from the target system and model is sought. Intelligent search methods, such as EAs can be used for this task.

EAs are population-based algorithms developed according to the concepts of natural selection and fitness-based survival. They are commonly used for difficult optimisation problems whereby a set, or population, of solutions is evolved iteratively by applying selection and recombination operators and generating new solutions in each step.

In recent years, several works in the literature proposed optimising an ESN design using EAs. They focused on optimising different features of an ESN architecture, including its weights [32], topology [33], design parameters [34] or combinations of them [35]. Applying EAs to optimize ESN designs have shown to produce better network performances than those with recommended settings [36].

In this work, three well-known EAs, a GA, DE and PSO are employed to optimize an ESN's design parameters (as discussed above), input weights W_{in} and reservoir weights W. The following Sections provide details for the encoding scheme and fitness evaluation mechanism used in our implementation of these three algorithms to tune an ESN.

3.2.1 Encoding

Consider a customised ESN with N reservoir neurons, K neurons for the input layer, and L neurons for the output layer where N is equal to the

number of system variables and K and L to the system's observable output variables (Figure 4). The numbers of input weights in W_{in} matrix, reservoir weights in W matrix and design parameters are $N\times K$, $N\times K$ and 4, respectively. A solution is described by an M real-coded values vector (flat) representation, where $M = 4+N\times K+N\times N$. The first four genes represents the design parameters α, δ, γ and ρ, respectively which are initialised uniformly randomly from $[0,1]$. The next $N\times K$ and $N\times N$ genes represent the input and reservoir weights, respectively which are initialised uniformly randomly from $[-1,1]$; for example, for a network with 2 input neurons and 10 reservoir ones, the total length M is 124 (sum of 4, 20 and 100 real-coded values for the design parameters, input weights and reservoir weights, respectively).

3.2.2 Fitness Evaluation

The objective function is designed to minimise two terms: the error (ε) between the observed system output and trained ESN output measured as the normalised root mean square error (NRMSE) [29]; and the density of the learned model determined in this work by the ESN's reservoir connectivity probability (ρ). ρ is used as a surrogate measure of structural similarity of the learned and target models as the structure of the later is considered unknown in the learning process. The motivation for using ρ comes from the observation that most expert-developed conceptual models are quite sparse. On the other hand, final trained ESNs tend to be very dense. Introducing the minimisation of ρ in the fitness function enables the selection pressure to be increased to learn sparse yet accurate models. A simple mean weighted sum method is used to combine these two terms into the objective function as

$$O = \frac{w_1\varepsilon + w_2\rho}{2}, \tag{5}$$

where w_1 and $w_2 \in [0,1]$ refer to the weights assigned to each of the two error terms, respectively. Later, in the experimental Section, we conduct a sensitivity analysis to determine the best combination of weights. The output error (ε), measured as the NRMSE, is computed as

$$\varepsilon(y_s,y_e) = \sqrt{\frac{\sum_{n=1}^{T}(y_s(n)-y_e(n))^2}{T\sigma_{y_s}^2}}, \tag{6}$$

where $y_s(n)$ refers to the target system's output, $y_e(n)$ the output generated from the learned model, T the length of the target's output test sequence and $\sigma_{y_s}^2$ the variance of the target's output.

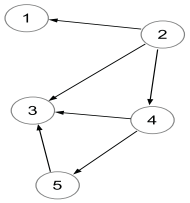
As previously mentioned, all the ESN design parameters are generated randomly from a uniform distribution in the $[0,1]$ range. Therefore, ρ is already bounded while the ε value is unbounded $[0,\infty]$. In order to bound the ε value to be within $[0,1]$ and avoid any bias in the fitness caused by this unbounded value, we apply the following rule

$$\varepsilon(y_s,y_e) = \begin{cases} 1 & \text{if } \varepsilon(y_s,y_e) > 1 \\ \varepsilon(y_s,y_e) & \text{otherwise.} \end{cases} \tag{7}$$

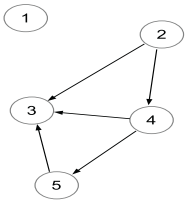
3.2.3 Handling of Infeasible Solutions

Since the reservoir connections are randomly initialised based on a given ρ value, there is a possibility that some of the solutions in the population result in disconnected networks. A solution is considered feasible if the learned structure is at least weakly connected.

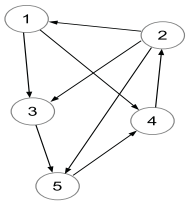
In order to define the connectivity of the learned model, we use the definition of connectivity for directed graphs form graph theory [37] since the learned models could be considered directed graphs. An undirected graph $G = (V,E)$ consists of two sets, vertices or nodes V and edges E. G is considered directed if all its edges are directed and is called G_d (Figure 5(a)). A directed edge is an edge with one of its endpoints designed as a head and the other as a tail. G_d could be disconnected (Figure 5(b)), strongly connected (Figure 5(c)) or weakly connected (Figure 5(d)). It is strongly connected if there is a path between every pair of vertices in V. On the other hand, it is weakly connected if the underlying undirected graph (G) is connected, that is, if it can be traversed using a search algorithm, such as the depth first search [38], every vertex in it is visited. Since we search for models with small connectivity parameter ρ values, weak connectivity will be sufficient to ensure they are connected.



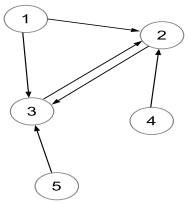
(a) Directed graph



(b) Disconnected



(c) Strongly connected



(d) Weakly connected

Figure 5. Types of connectivity for directed graph.

Several approaches for handling infeasible solutions generated during an evolutionary search are proposed [39]. We adopt the method of adding a varying penalty term to the fitness function. As this method does not discard every infeasible solution in earlier generations of the evolution, it allows potentially useful genetic material into better solutions. However, the penalty term increases as the search progresses (as the number of generations increase) to prevent infeasible solutions taking over the space for feasible solutions in the population [40].

The varying penalty term for each solution is calculated as

$$P_v = [(\frac{g}{G}A).d + \frac{g}{G}B].v, \tag{8}$$

where P_v refers to the varying penalty function, g the current generation, G the maximum number of generations, A the severity factor, B the penalty threshold factor, d the level or severity of a violation and v a binary value that indicates if the solution is infeasible or not and is calculated for a solution with structure S and output error $\varepsilon(S)$ as

$$v = \begin{cases} 1 & \text{if S is disconnected or } \varepsilon(S) > 1, \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

The number of violations (d) for each solution with structure S and output error $\varepsilon(S)$ is calculated as

$$d = \begin{cases} 2 & \text{if S is disconnected and } \varepsilon(S) > 1, \\ 1 & \text{if S is disconnected or } \varepsilon(S) > 1, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

By combining terms in both equations 5 and 8, the fitness function is calculated as

$$F = O + P_v. \tag{11}$$

3.3 Measuring Model Similarity

A model similarity metric is used to measure the structure resemblance between the learned and target models. It is not used in the learning process since the target system's conceptual model is considered unknown and the goal is to learn it using only the system's observable output. There are several approaches proposed in the literature for comparing conceptual conceptual models based on some features, such as number of variables, causal links and feedback loops. In this paper, we adopt a measure called the distance ratio (DR) [41] which measures the distance between two conceptual models. It uses the information of the number of nodes and causal links, and their directions represented in an associated adjacency matrix. Since both target and learned models share the same information, we can apply it to measure model similarity. The structure error (ψ), measured using DR, is calculated as

$$\psi = \frac{\sum_{i=1}^{p}\sum_{j=1}^{p}|a_{ij} - b_{ij}|}{2(p^2 - p)}, \tag{12}$$

where a_{ij} and b_{ij} are the cell indices in the adjacency matrices representing the target and learned models, respectively. Both a_{ij} and b_{ij} can have a value of one or zero which indicates if there is a link, or not, between nodes i and j, respectively, in their corresponding adjacency matrices. The total number of system variables, (p), is the same for both models. Simply, $\psi \in [0,1]$ provides the average number of differences between the two structures normalised over all possible links and indicates their closeness, with lower values indicating greater similarity.

4 Experimental Setup

The effectiveness of our methodology was evaluated using four examples of complex systems taken from different domains, one from each of economics and ecology and two from physical systems. These examples are considered because they all have well-developed conceptual models related to their different system variables and how they behave. The target outputs for all four examples are generated through simulating their SD simulation models [1]. For all examples, we implement our own simulation that involved solving model equations using the Euler method. Each of these systems and their conceptual model structures are explained in the Sections below. All the methods adopted, including ESNs, EAs and measures for computing the structural similarity between models, are implemented using Matlab 8.5. The experiments are run on an Intel core i7 CPU 3.4 GHz, with 16 GB of RAM and Windows 7 (64 bit) machine.

Table 1. Different weight settings of fitness function

Weights combination identifier	**Weights' values**
WC_1	$(w_1 = 0.9, w_2 = 0.1)$
WC_2	$(w_1 = 0.8, w_2 = 0.2)$
WC_3	$(w_1 = 0.7, w_2 = 0.3)$
WC_4	$(w_1 = 0.6, w_2 = 0.4)$
WC_5	$(w_1 = 0.5, w_2 = 0.5)$
WC_6	$(w_1 = 0.4, w_2 = 0.6)$
WC_7	$(w_1 = 0.3, w_2 = 0.7)$
WC_8	$(w_1 = 0.2, w_2 = 0.8)$
WC_9	$(w_1 = 0.1, w_2 = 0.9)$

In the experiments, a cross-validation method is used to train our ESN models. The training data is collected using the outputs generated through the simulation models for each system, as described above. Each data set is normalised to have a zero mean and unit variance, and is split into two sets, training and test, using 80%/20% splits. For each example, GA, DE and PSO, are used to evolve the customised ESN parameters, as explained in Section 3.2. During an experiment, the EA is initialised by generating random solutions (an initial population) following the encoding schema (Section 3.2.1). A network model is created from each solution and the fitness value calculated using Equation (11). This initial population is evolved over a number of generations until the maximum number of generations is reached, a process repeated for 20 independent runs with different seeds.

Each of the above experiments is repeated with 9 different weight settings (Table 1) to test the sensitivity of the objective weights given in Equation (5). This provides a total of 27 experiments (9 weight combinations $\times$ 3 EAs) for each example.

For the EAs, a real coded GA [42] is used with a tournament selection mechanism of size 3, an arithmetic crossover with a range factor of 0.4 and a random uniform mutation with a rate of 0.1. A generational scheme is used during the evolution whereby a new population of solutions is generated in every iteration with a crossover pecentage of 0.7 and mutation percentage of 0.3. For DE, we use random uniform mutation, bin schema crossover and greedy selection mechanism proposed in [15]. The lower and upper bounds of the scaling factor are 0.2 and 0.8, respectively, and the crossover probability 0.2. For PSO, a static topology [43] and inertia weight method to update particles' positions and velocities [44] are used. The inertia weight is 1, inertia weight damping ration 0.99, personal learning coefficient 1.5 and global learning coefficient 2.0. For all algorithms, the population size is 100, maximum number of generations 100, number of runs 20, severity factor (A) 1000 and penalty threshold factor (B) zero.

In order to compare the effectiveness of the evolutionary search, an additional experiment is also repeated for a customised ESN without applying evolutionary tuning to its design (ESN_{base}).

For each example, the results (Section 5) show the best learned models based on the output error (lowest ε) and structural error (lowest ψ). These models and their outputs and compared with the target outputs and structures, and the best models learned by the ESN_{base}. We also present an analysis of the trade-offs between the learned models' output and structural errors as well as convergence analysis. All these results are averaged over 20 independent runs, with the best solution selected based on the minimum fitness value obtained from each run over all generations.

4.1 Complex Systems Examples

4.1.1 Workforce Inventory (WI) System

This model shows the interactions between the inventory management sector and employment and labour sectors in a supply chain management system [1]. The conceptual model consists of 13 variables, 17 links and 3 feedback loops, as shown in Figure 6(a). Each node (Ci) in the model represents one of the system variables as: $C1$ (production); $C2$ (inventory); $C3$ (productivity); $C4$ (sales); $C5$ (inventory coverage); $C6$ (target production); $C7$ (target inventory); $C8$ (inventory correction); $C9$ (workforce); $C10$ (net hire rate); $C11$ (time to adjust workforce); $C12$ (target workforce); and $C13$ (time to correct inventory). In Figure 6(e), the simulated behaviour at the two system output variables $C2$ and $C9$ show their growths reaching a steady state.

4.1.2 Lotka-Volterra (LV) System

This is an ecological model that shows natural interactions between two populations (e.g., preys and predators) which mutually affect each each other [45]. The conceptual model shown in Figure 6(b) has 10 variables, 17 links and 8 feedback loops. Each node (Ci) representing one system variables as: $C1$ (prey births); $C2$ (prey population); $C3$ (prey deaths); $C4$ (predator births); $C5$ (predator population); $C6$ (predator deaths); $C7$ (capacity); $C8$ (prey crowds); $C9$ (predator consumption); and $C10$ (food availability for predators). The simulated output from this model at variables $C2$ and $C5$ shown in Figure 6(f) indicate oscillatory behaviour.

4.1.3 Van der Pol Oscillator (VdPO) System

This model describes the non-linear dynamics in a physical system consisting of vacuum tube circuits designed as part of the development process for electronics technologies [46]. The conceptual model shown in Figure 6(c) has 8 variables, 10 links and 3 feedback loops. Each node (Ci) representing one of the system variables as: $C1$ (X); $C2$ ($\frac{dX}{dt}$); $C3$ (Y); $C4$ ($\frac{dY}{dt}$); $C5$ (effect of X on Y); $C6$ (w^2); $C7$ (u); and $C8$ (k). Its behaviours generated at the output variables $C1$ and $C3$ are chaotic and oscillatory, as shown in Figure 6(g).

4.1.4 Lorenz Equations (LE) System

This model is developed for atmospheric convections as part of a weather forecasting simulation [47]. Its structure, as shown in Figure 6(d) has 9 variables, 14 links and 5 feedback loops. Each node (Ci) representing one of the system variables as: $C1$ (X); $C2$ ($\frac{dX}{dt}$;, $C3$ (σ); $C4$ (r); $C5$ (Y); $C6$ ($\frac{dY}{dt}$); $C7$ (Z); $C8$ ($\frac{dZ}{dt}$); and $C9$ (b). Its behaviours generated at the output variables $C1$, $C5$ and $C7$ are chaotic and oscillatory, as shown in Figure 6(h).

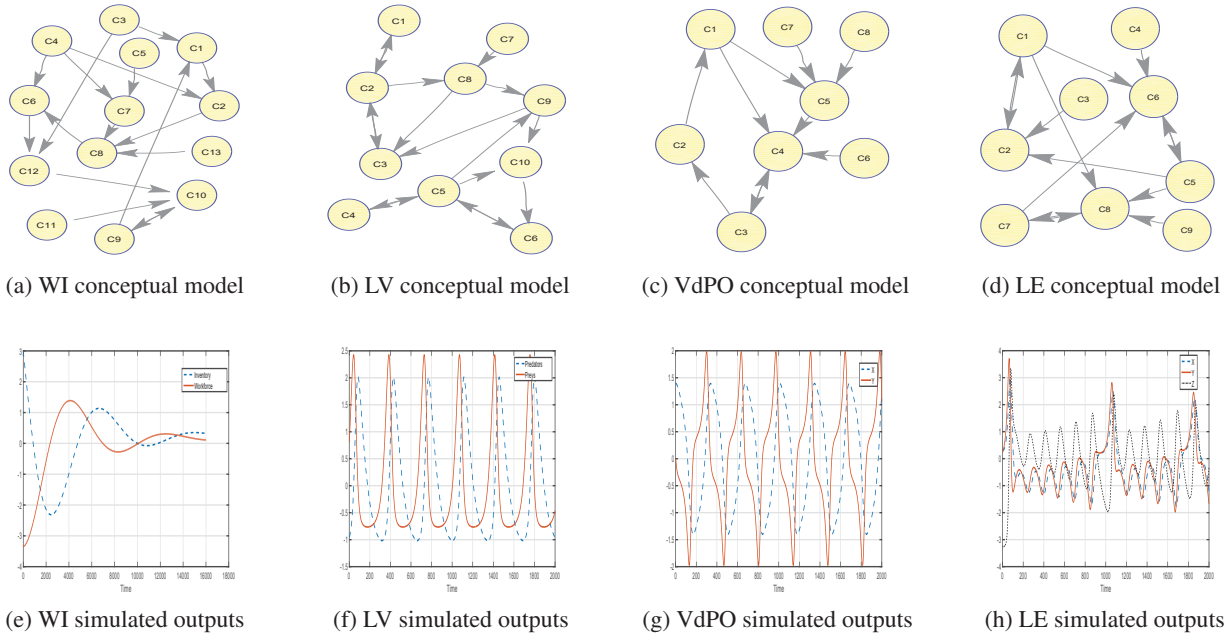
5 Results and Analysis

5.1 Learning Performance

To analyse the effectiveness of the proposed evolutionary ESN for learning target model structures, we observe the best learned ones based on both their output errors (ε) and structural errors (ψ).

5.1.1 Best Models Learned Based on Output Error (ε)

The outputs from the best learned models for each of the four examples are provided in Figures 8(a) to 8(d), respectively. These models are selected based on their minimum output errors (ε) across all 27 setups (9 weight combinations $\times$ 3 EAs) as follows. Firstly, for every independent run in each experiment carried out with a specific weight combination (WC_x) and specific EA, the best learned model is selected based on its minimum ε over all 100 generations, a process repeated for all 20 independent runs. Next, the Friedman test [48] of significance is applied to rank solutions from all 27 setups, with the setup with the highest-ranked solution selected as the best. Finally, the best model is selected based on its minimum ε obtained from 20 runs of the best setup. Figures 7(b),7(e), 7(h) and 7(k) show the model structures that generated these outputs. Also, outputs generated by ESN_{base} (Figures 8(a) – 8(d)) and the corresponding model structures (7(c), 7(f), 7(i) and 7(l)) used to generate them are shown for comparing against the models learned by the optimised ESN models. Table 5.1.1 shows the correctly predicted, missing and additional links present in the learned structures obtained by both the optimised ESN and ESN_{base} in order to demonstrate how close these structures are to the target ones. Calculation of these links are based on com-



(a) WI conceptual model (b) LV conceptual model (c) VdPO conceptual model (d) LE conceptual model

(e) WI simulated outputs (f) LV simulated outputs (g) VdPO simulated outputs (h) LE simulated outputs

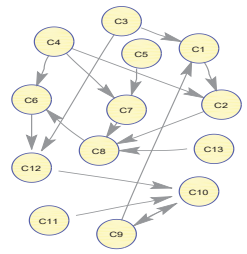
Figure 6. Target conceptual models and simulated outputs of complex systems examples

paring those in the learned structure with those in the target one for each example. For a directed link from variable x_i to variable x_j in the target structure, if it is also in the learned structure, it is called a correct link, otherwise a missing link and, if it is not in the target structure but in the learned one, it is called an additional link.

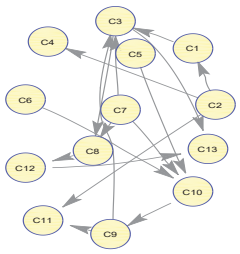
For the WI, LV and VdPO examples, the outputs generated from the best learned structures overlap with the target outputs (Figures 8(a), 8(b) and 8(c)). However, in LE example, the generated outputs do not exactly match but the error is not large (Figure 8(d)). The learned structures (Figures 7(b), 7(e) and 7(h)) that generate those outputs are quite similar to the target structures in the WI, LV and VdPO examples (Figures 7(a), 7(d) and 7(g)). In WI, the learned structure has 2 correct links out of 17 links, 15 missing and 14 additional (Table 5.1.1) while, in both LV and VdPO, they have more correct links, 4 out of 17 and 4 out of 10, respectively (Table 5.1.1). In the LV example, the numbers of missing and additional links are 13 and 9, respectively. In the VdPO example, there are fewer missing and additional links than in both the WI and LV examples, 6 and 5, respectively. The LE example has 7 correct links out of 14, the highest of all the examples. However, its learned structure (Figure 7(k)) is much more dense than the target (Figure 7(j)) and other ones, as supported by its large number of additional links 33 (Table 5.1.1).

The outputs from the best structures learned by the ESN_{base} overlap with the target outputs for the first three cases (Figures 8(a), 8(b) and 8(c)). However, in the LE example, the optimised ESN performs better in terms of matching the target output's behaviour (Figure 8(d)). In the WI example, the structure learned by the ESN_{base} (Figure 7(c)) is more similar to the target structure (Figure 7(a)) than the learned by the optimised ESN (Figure 7(b)) in terms of more correct links (4 links) and fewer missing and additional ones (13 and 11, respectively). In the other examples, LV, VdPO and LE, the structures learned by the ESN_{base} are very dense (Figures 7(f), 7(i) and 7(l)) in terms of their large number of additional links (57 for LV, 42 for VdPO and 21 for LE) (Table 5.1.1). However, in the LE example, the number of additional links (21) is less than those in the structure learned by the optimised ESN (33) (Table 5.1.1).

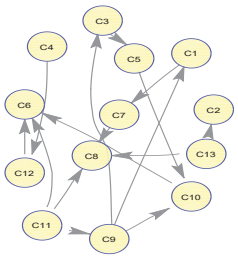
In addition to showing the errors and structures of the best (or to be accurate, the best of the best) learned models over all 27 setups, Table 3 shows the mean and standard deviation values of their averaged output errors (ε). The best models for different weight combinations for each EA and different EAs for each weight combination are identified using two types of statistical tests. The one-way



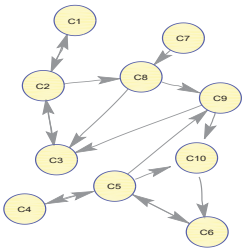
(a) WI target structure



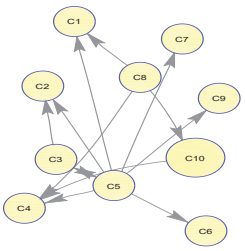
(b) WI best model learned using optimised ESN



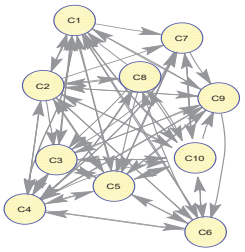
(c) WI best model learned using ESN_{base}



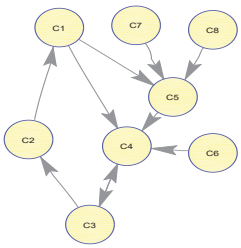
(d) LV target structure



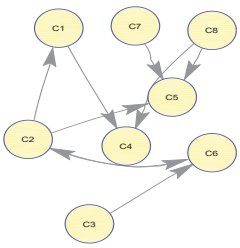
(e) LV best model learned using optimised ESN



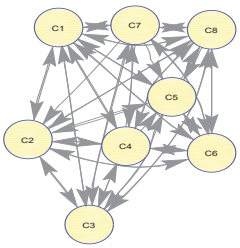
(f) LV best model learned using ESN_{base}



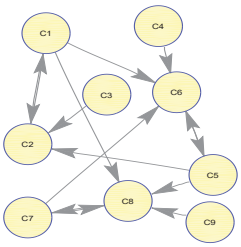
(g) VdPO target structure



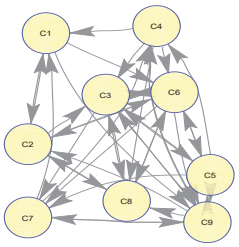
(h) VdPO best model learned using optimised ESN



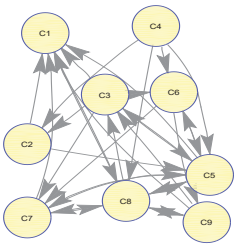
(i) VdPO best model learned using ESN_{base}



(j) LE target structure



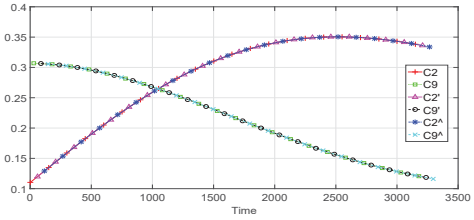
(k) LE best model learned using optimised ESN



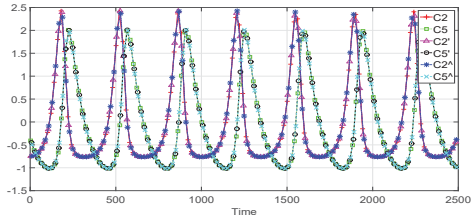
(l) LE best model learned using ESN_{base}

Figure 7. Best structures learned by optimised ESN and ESN_{base} compared with target structures based on output error ε of all examples

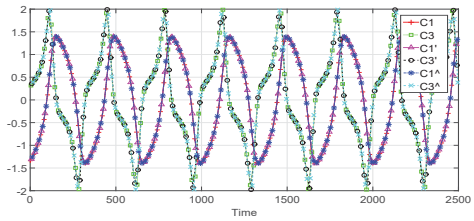
ANOVA [49] is used as a significance test and the Friedman test [48] for ranking. Different symbols (†, ∗ and •) are used to differentiate the significantly better models across different setups, as explained in the caption for Table 3.



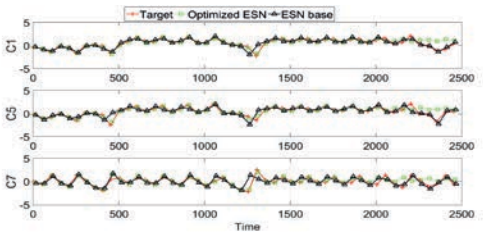
(a) WI target outputs (C2, C9), outputs generated by best learned model (C2', C9') and ESN_{base} ($\hat{C}2$, $\hat{C}9$)



(b) LV target outputs (C2, C5), outputs generated by best learned model (C2', C5') and ESN_{base} ($\hat{C}2$, $\hat{C}5$)



(c) VdPO target outputs (C1, C3), outputs generated by best learned model (C1', C3') and ESN_{base} ($\hat{C}1$, $\hat{C}3$)



(d) LE target outputs (C1, C5, C7), outputs generated by best learned model and ESN_{base}

Figure 8. Outputs generated from best structures learned by optimised ESN and ESN_{base} compared with target outputs based on output error (ε) and structure error (ψ). Due to the exact matching with the generated outputs from best structures learned based on ψ, the outputs based on ε are only shown to avoid repetition

Table 2. Correct (C), missing (M) and additional (A) links for each best structure learned by optimised ESN and ESN_{base} based on output error ε (Total number of links for target structure for each example presented next to example name as $T : N$, where N number of total links)

	C	M	A
WI - T:17			
Optimised ESN	2	15	14
ESN_{base}	4	13	11
LV - T:17			
Optimised ESN	4	13	9
ESN_{base}	13	4	57
VdPO - T:10			
Optimised ESN	4	6	5
ESN_{base}	9	1	42
LE - T:14			
Optimised ESN	7	7	33
ESN_{base}	10	4	21

There are significant differences among the results obtained from the EAs for different weight settings in all examples except LV. In the WI one, the best results using the GA and DE are obtained with WC_4, and using PSO, with WC_6. In the VdPO example, the best results obtained using the GA are with WC_2 and, using PSO and DE, with WC_1. Finally, in the LE example, the best results obtained using the GA are with WC_2 and, using DE and PSO, with WC_1. There are significant differences among the results obtained from the EAs for almost all weight settings, with the best for all examples, from GA. Also, there are significant differences among the results obtained for all setups in all examples. For the WI example, the best results are obtained using the GA with WC_4, for both LV and VdPO ones, using the GA with WC_2 and, for the LE one, using PSO with WC_1.

In summary, in all examples except WI, the fitness function weight settings make a difference in terms of generating similar behaviours, with WI exhibiting the simplest. The GA outperforms the other EAs for the weight settings and overall setups, except that, for the LE example, PSO learns the best model. Most of the best models learned in all examples are found at earlier weight settings (WC_1 to WC_4) which is reasonable as we rely on the minimum output error to find the best models learned.

Table 3. ε means and standard deviations of best models learned for all 27 setups († indicates best for each EA, ∗ best for each weight setting and bold • overall best)

	WI		
	GA	**DE**	**PSO**
	mean(±std)	**mean(±std)**	**mean(±std)**
WC_1	∗ 4.8e-4(±2.75e-4)	5.38e-4(±3.31e-4)	5.74e-4(±3.29e-4)
WC_2	4.64e-4(±2.54e-4)	4.06e-4(±2.4e-4)	∗ 4.2e-4(±2.6e-4)
WC_3	∗ 3.48e-4(±2.4e-4)	6.28e-4(±4.04e-4)	4.24e-4(±2.8e-4)
WC_4	† ∗ • **2.61e-4(±1.33e-4)**	† 4.83e-4(±3.2e-4)	4.48e-4(±3.28e-4)
WC_5	∗ 4.13e-4(±3.03e-4)	4.77e-4(±2.14e-4)	4.85e-4(±3.07e-4)
WC_6	∗ 2.69e-4(±1.87e-4)	4.27e-4(±2.31e-4)	† 4.5e-4(±3.28e-4)
WC_7	∗ 3.28e-4(±1.68e-4)	4.16e-4(±2.57e-4)	4.2e-4(±3.1e-4)
WC_8	∗ 2.82e-4(±1.75e-4)	4.78e-4(±2.61e-4)	5.52e-4(±3.22e-4)
WC_9	∗ 3.75e-4(±2.81e-4)	3.05e-4(±1.91e-4)	4.29e-4(±2.98e-4)
	LV		
WC_1	∗ 1.38e-4(±6.82e-5)	1.09e-3(±5.27e-4)	† 3.72e-4(±3.19e-4)
WC_2	† ∗ • **1.28e-4(±9.74e-5)**	9.74e-4(±1.14e-3)	3.54e-4(±2.34e-4)
WC_3	∗ 1.81e-4(±1.14e-4)	1.28e-3(±9.77e-4)	5.28e-4(±6.92e-4)
WC_4	∗ 1.7e-4(±1.05e-4)	1.41e-3(±2.01e-3)	3.97e-4(±2.77e-4)
WC_5	∗ 1.28e-4(±6.63e-5)	1.22e-3(±1.45e-3)	4.38e-4(±2.4e-4)
WC_6	∗ 1.51e-4(±1.1e-4)	1.32e-3(±1.03e-3)	3.73e-4(±2.45e-4)
WC_7	∗ 2.18e-4(±1.9e-4)	9.95e-4(±5.01e-4)	4.59e-4(±4.62e-4)
WC_8	∗ 1.82e-4(±1.25e-4)	1.13e-3(±1.04e-3)	3.41e-4(±3.37e-4)
WC_9	∗ 1.67e-4(±1.64e-4)	† 8.32e-4(±6.59e-4)	6.88e-4(±6.36e-4)
	VdPO		
WC_1	∗ 1.44e-3(±7.62e-4)	† 4.4e-3(±1.99e-3)	† 2.27e-3(±1.13e-3)
WC_2	† ∗ • **1.29e-3(±8.62e-4)**	5.85e-3(±2.05e-3)	2.31e-3(±1.21e-3)
WC_3	∗ 1.9e-3(±1.43e-3)	6.06e-3(±4.02e-3)	3.78e-3(±2.85e-3)
WC_4	∗ 1.36e-3(±7e-4)	1.12e-2(±6.08e-3)	3.51e-3(±2.85e-3)
WC_5	∗ 1.95e-3(±1.59e-3)	1.19e-2(±5.16e-3)	3.66e-3(±3.22e-3)
WC_6	∗ 1.85e-3(±1.12e-3)	8.96e-3(±6.22e-3)	3.25e-3(±3.47e-3)
WC_7	∗ 1.49e-3(±1.11e-3)	1.3e-2(±1.06e-2)	5.35e-3(±4.29e-3)
WC_8	∗ 1.77e-3(±1.1e-3)	1.12e-2(±9.38e-3)	3.41e-4(±3.37e-4)
WC_9	∗ 2.34e-3(±1.86e-3)	1.57e-2(±1.63e-2)	1e-2(±1.63e-2)
	LE		
WC_1	∗ 0.7477(±1.19e-1)	† 0.8163(±4.93e-2)	† • **0.7614(±6.82e-2)**
WC_2	† 0.7277(±1.39e-1)	0.8276(±5.54e-2)	∗ 0.7835(±4.1e-2)
WC_3	∗ 0.7509(±9.19e-2)	0.8383(±5.82e-2)	0.8019(±3.86e-2)
WC_4	∗ 0.7706(±8.44e-2)	0.8533(±6.02e-2)	0.8137(±3.25e-2)
WC_5	∗ 0.798(±9.26e-2)	0.8477(±6.1e-2)	0.8216(±3.72e-2)
WC_6	∗ 0.7538(±9.65e-2)	0.8546(±6.94e-2)	0.806(±6.32e-2)
WC_7	0.8674(±6.85e-2)	0.8635(±7.32e-2)	∗ 0.8275(±8.15e-2)
WC_8	0.8217(±0.1564)	0.9025(±5.5e-2)	∗ 0.8604(±4.59e-2)
WC_9	∗ 0.9214(±4.31e-2)	0.9222(±5.02e-2)	0.9287(±4.9e-2)

5.1.2 Best Models Learned Based on the Structural Error (ψ)

In this Section, we show the best models learned based on the structural error ψ by applying the process discussed in Section 5.1.1. The main aim is to demonstrate how these structures are similar to target ones and how the outputs generated from these models are similar to the target ones. The structures of the best models learned for each of the four examples are provided in Figures 9(b), 9(e), 9(h) and 9(k). The outputs generated from these models are almost match the outputs generated from the models based on output error (ε) as shown in Figure 8. Also, the ESN_{base} models (Figures 9(c), 9(f), 9(i) and 9(l)) and their generated outputs (Figure 8) are compared with the models learned by the optimised ESN. Table 4 shows the correctly predicted, missing and additional links present in the models learned by both optimised ESN and ESN_{base} in order to demonstrate how close these models are to the target ones.

Table 4. Correct (C), missing (M) and additional (A) links for each best structure learned by optimised ESN and ESN_{base} based on structural error ψ (Total number of links for target structure for each example presented next to example name as $T : N$, where N number of total links)

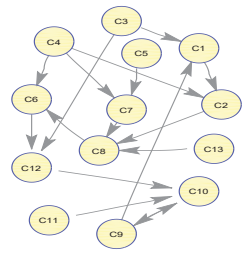
	C	M	A
WI - T:17			
Optimised ESN	4	13	15
ESN_{base}	4	13	11
LV - T:17			
Optimised ESN	3	14	9
ESN_{base}	13	4	57
VdPO - T:10			
Optimised ESN	2	8	8
ESN_{base}	9	1	42
LE - T:14			
Optimised ESN	2	12	10
ESN_{base}	10	4	21

In the WI example, the learned structure (Figure 9(b)) has 4 correct links out of 17, 13 missing and 15 additional (Table 4) while, in the remaining examples, there are fewer correct links in the learned structures (Figures 9(e), 9(h) and 9(k)) (3 in the LV and 2 in both the VdPO and LE examples) and also fewer missing and additional ones (Table 4). The outputs generated from the best structures learned for the WI, LV and VdPO examples overlap with those of the target (Figures 8(a), 8(b) and 8(c)). However, for the LE example, while the generated outputs do not exactly match, the error is not large until time step 500, after which the outputs are fixed at constant values for the rest of the time horizon (Figure 8(d)). For all examples, the structures learned by the ESN_{base} (Figures 9(c), 9(f), 9(i) and 9(l)) are much denser than those learned by the optimised ESN, as supported by the large number of additional links exist in these structures (Table 4).

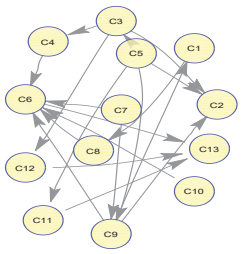
Table 5 shows the mean and standard deviation values of the structural errors (ψ) averaged over all 27 setups. The best models obtained for different weight combinations using each EA and different EAs for each weight combination are identified using the two types of statistical tests discussed in Section 5.1.1. Different symbols (†, ∗ and •) are used to identify the significantly better models in the different setups, as explained in the captions of Table 5.

Of the EAs, in the WI example, the best results are obtained by the GA with WC_4, DE with WC_7 and PSO with WC_6. In the LV example, the GA obtains the best results with WC_8, DE with WC_5 and PSO with WC_6. In the VdPO example, the GA obtains the best results with WC_3, DE with WC_5, and PSO with WC_2. Finally, in the LE example, both the GA and DE obtain the best results with WC_9 and PSO with WC_8. However, the results obtained from both EAs and different fitness weight settings for all examples are not statistically significant based on the ANOVA statistical test, except for LE example. Over all setups, for the WI example, the best results are obtained using PSO with WC_6, for the LV example, using PSO with WC_2, for the VdPO example, using DE with WC_5 and, for the LE example, using PSO at WC_9. However, there is no statistical significant among the results obtained across all setups for all examples except for LE.

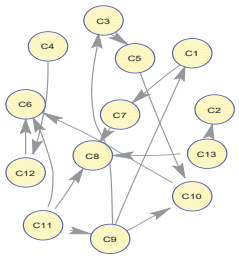
In summary, changing weight settings does not significantly affect the performance of the ESN for any EA, except for LE example which is due to its complexity in terms of structure and behaviour. Most of the best models obtained by each EA are with weight settings of WC_8 and WC_9. Overall, the best learned structures are found by PSO in all examples, except for VdPO one where it is found



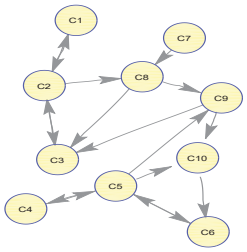
(a) WI target structure



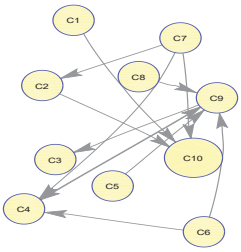
(b) WI best model learned using optimised ESN



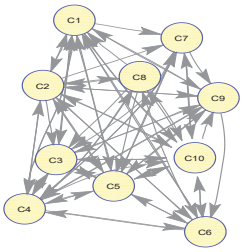
(c) WI best model learned using ESN_{base}



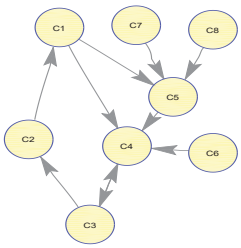
(d) LV target structure



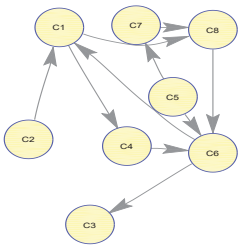
(e) LV best model learned using optimised ESN



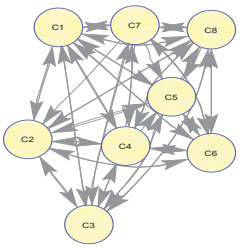
(f) LV best model learned using ESN_{base}



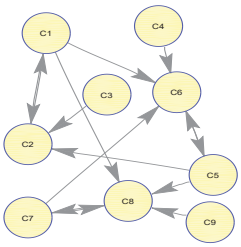
(g) VdPO target structure



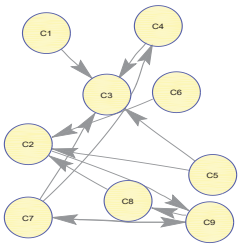
(h) VdPO best model learned using optimised ESN



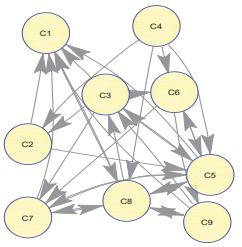
(i) VdPO best model learned using ESN_{base}



(j) LE target structure



(k) LE best model learned using optimised ESN



(l) LE best model learned using ESN_{base}

Figure 9. Best structures learned by optimised ESN and ESN_{base} compared with target structures based on output error ψ of all examples

Table 5. ψ means and standard deviations of best models learned for all 27 setups († indicates best for each EA, $*$ best for each weight setting and bold • overall best)

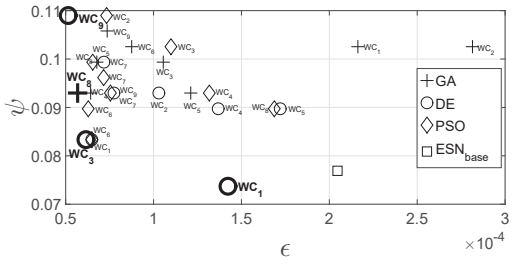
	WI		
	GA	**DE**	**PSO**
	mean(±std)	**mean(±std)**	**mean(±std)**
WC_1	0.0997(±0.0143)	0.0946(±0.0119)	$*$ 0.0949(±0.0092)
WC_2	$*$ 0.0954(±0.011)	0.0968(±0.0091)	0.0963(±0.0128)
WC_3	0.1(±0.0121)	$*$ 0.0938(±0.0094)	0.0954(±0.0092)
WC_4	† 0.0933(±0.0093)	$*$ 0.0931(±0.0079)	0.0957(±0.0077)
WC_5	$*$ 0.0936(±0.0101)	0.0968(±0.0085)	0.0963(±0.0107)
WC_6	0.0933(±0.0081)	0.0973(±0.0107)	† $*$ • **0.091(±0.0102)**
WC_7	$*$ 0.0931(±0.0093)	† 0.0921(±0.0104)	0.0946(±0.0123)
WC_8	$*$ 0.0966(±0.0061)	0.0984(±0.0131)	0.0995(±0.0116)
WC_9	0.0987(±0.0069)	0.0941(±0.012)	$*$ 0.0921(±0.0103)
	LV		
WC_1	$*$ 0.1375(±0.0196)	0.1427(±0.0111)	0.1416(±0.0157)
WC_2	0.1475(±0.0253)	0.1361(±0.0156)	$*$ • **0.1325(±0.0132)**
WC_3	$*$ 0.1375(±0.0218)	0.1383(±0.0149)	0.1427(±0.0152)
WC_4	0.138(±0.0138)	0.1397(±0.0148)	$*$ 0.1358(±0.019)
WC_5	0.1433(±0.0253)	† $*$ 0.1375(±0.0147)	0.1375(±0.0173)
WC_6	0.1372(±0.0182)	0.1438(±0.0155)	† $*$ 0.1325(±0.0133)
WC_7	0.1411(±0.0203)	$*$ 0.1375(±0.019)	0.138(±0.0196)
WC_8	† $*$ 0.13417(±0.0212)	0.1386(±0.0172)	0.14(±0.0162)
WC_9	0.138(±0.0172)	$*$ 0.1372(±0.0134)	0.1372(±0.0114)
	VdPO		
WC_1	0.1468(±0.0202)	$*$ 0.1468(±0.0248)	0.1468(±0.0212)
WC_2	0.1437(±0.0185)	0.1562(±0.0253)	† $*$ 0.1383(±0.0094)
WC_3	† $*$ 0.1419(±0.0147)	0.1468(±0.0175)	0.1468(±0.0218)
WC_4	0.1526(±0.0275)	0.1406(±0.0174)	$*$ 0.1428(±0.0148)
WC_5	0.15(±0.0227)	† $*$ • **0.1348(±0.0183)**	0.1442(±0.0217)
WC_6	0.1486(±0.0151)	0.1517(±0.0261)	$*$ 0.1375(±0.0231)
WC_7	0.15(±0.0175)	$*$ 0.1442(±0.0245)	0.1455(±0.0243)
WC_8	0.15(±0.0227)	0.1491(±0.0249)	$*$ 0.145(±0.0239)
WC_9	$*$ 0.1419(±0.0254)	0.1477(±0.0321)	0.1477(±0.0321)
	LE		
WC_1	0.3055(±0.0658)	$*$ 0.2201(±0.0711)	0.2847(±0.0433)
WC_2	0.2298(±0.0719)	$*$ 0.1947(±0.058)	0.2677(±0.0538)
WC_3	0.2142(±0.0616)	$*$ 0.1784(±0.0344)	0.2149(±0.0481)
WC_4	0.1881(±0.0345)	$*$ 0.1694(±0.0259)	0.1913(±0.0342)
WC_5	0.1697(±0.0323)	$*$ 0.1618(±0.0249)	0.1836(±0.0303)
WC_6	0.1822(±0.0339)	$*$ 0.1628(±0.0241)	0.1833(±0.0342)
WC_7	$*$ 0.1513(±0.0212)	0.1593(±0.0245)	0.1718(±0.0259)
WC_8	0.1611(±0.0248)	0.1562(±0.018)	† $*$ 0.1496(±0.0227)
WC_9	† 0.1489(±0.0237)	† 0.1527(±0.017)	$*$ • **0.1472(±0.0166)**

using DE. In LE example, both the best learned structures selected based on the output error (ε) and structural error (ψ) are found by PSO at the two extremes of weight settings, WC_1 for that based on (ε) and WC_9 for that based on (ψ). This shows that, for more complex examples, PSO is the best and the weight settings depend on the measure used to select the best model.

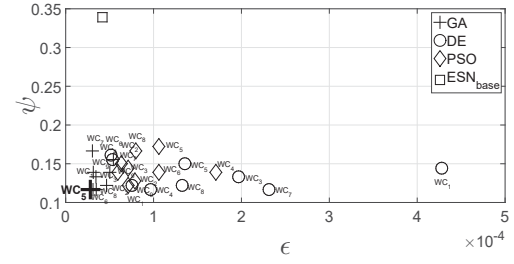
5.1.3 ε and ψ Trade-off Analysis

In this Section, we analyse the trade-off between the output error and structural error measures, ε and ψ, respectively, for the best models learned over all 27 setups and the ESN_{base} one. The aim of this type of analysis is to allow us to investigate the non-dominated experiemtnal setup for the two objectives, ε and ψ. To perform it, we first sort all the 28 best models using non-dominated sorting algorithm [50] according to these two objectives. The Pareto fronts resulting from this sorting for each example are shown in Figures 10(a) to 10(d), respectively, by highlighting the first Pareto with bold. The best model learned is selected based on the minimum fitness value over the 100 generations, a process repeated for runs as explained in Section 5.1.1.

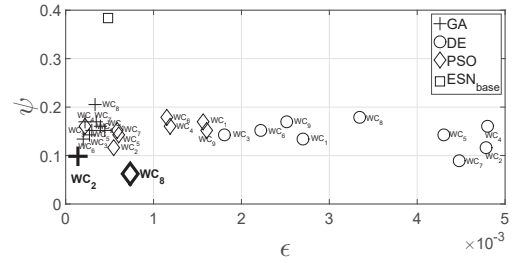
In the WI example, the Pareto (Figure 10(a)) contains four models learned by both the GA and DE algorithm with weight settings of WC_1, WC_3, WC_8 and WC_9 that dominate all other learned models. In the LV example, the Pareto (Figure 10(b)) contains only one model learned by the GA with WC_5 that dominates all the other models in all setups. In the VdPO example, the Pareto (Figure 10(c)) contains two models, one learned by the GA with WC_2 and the other by PSO with WC_8. These two models dominate the others over all the setups. In the LE example, the Pareto (Figure 10(d)) contains five models, four learned by the GA with weight settings of WC_1, WC_2, WC_5, and WC_7, and one by the ESN_{base}. In summary, the GA is able to learn models that have both minimum output and structural errors for both the LV and LE examples while the best models learned for the WI examples are found by the DE and PSO, respectively.



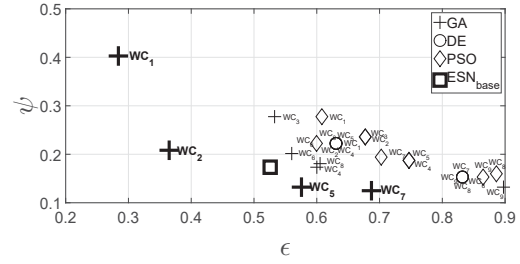
(a) WI



(b) LV



(c) VdPO



(d) LE

Figure 10. Pareto fronts resulting from non-dominated sorting of the 28 best models learned (27 setups from optimised ESN + ESN_{base}, with first Pareto highlighted with bold)

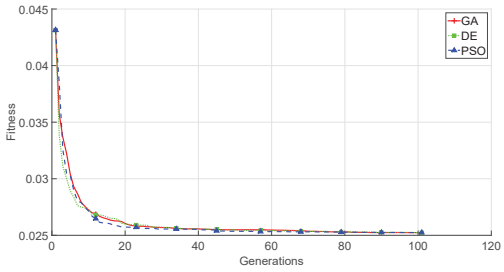
5.2 Evolutionary Algorithms Convergence Analysis

In this Section, the convergence speeds of the EAs for each example are compared. As most weight settings have no statistical significance, as discussed in Section 5.1, WC_5 is chosen for each EA. The aim of this analysis is to investigate the differences among the EAs for each examples and whether an example's complexity affects the convergence speed. For each example (Figures 11(a) to 11(d)), we illustrate the convergence plots that show how the fitness values of the best solutions change over generations which are averaged over the 20 independent runs.

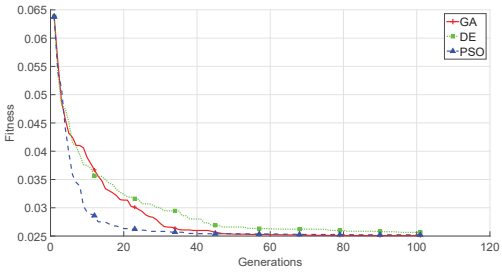
In the WI example, there is no difference among the EAs as most converge at the same rate (Figure 11(a)). In the LV (Figure 11(b)) and VdPO (Figure 11(c)), PSO converges faster than the other EAs followed by the GA and DE. In the LE (Figure 11(d)), the GA converges faster than the other EAs followed by PSO and then DE. Based on these observations, we can say that these examples affect in the algorithms' convergence speeds. Also, in most cases, PSO converges faster than the other EAs.

6 Conclusions

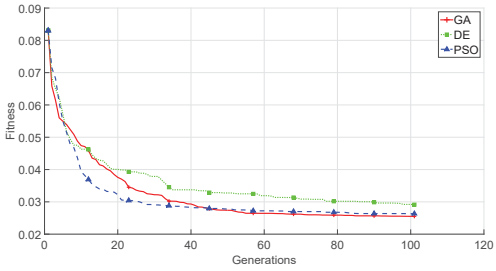
Conceptual models play a crucial role in modelling a complex dynamical system by providing a modeller with high-level abstract view of the system and how its components influence each other. They consist of a system's variables and the directional connections that describe how these variables interact. Developing such models requires a great deal of effort from modellers to obtain an acceptable representation for a particular system.



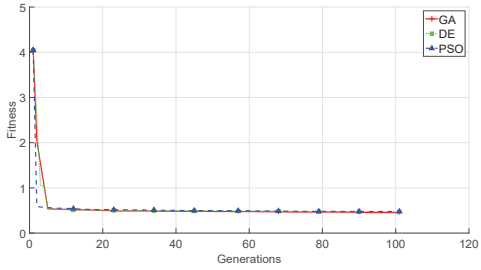
(a) WI



(b) LV



(c) VdPO



(d) LE

Figure 11. Convergence plots for EAs with weight setting WC_5 for each example

In this paper, a computational intelligence-based methodology for learning these conceptual models from observations of a system's output is proposed. Echo state networks, which are types of recurrent neural networks, are adopted to represent and simulate conceptual models. Three evolutionary algorithms, a genetic algorithm, differential evolution and particle swarm optimization are applied to optimize these ESNs. Several modifications to standard ESNs based on a few similarities between the structures of the conceptual models and an ESN's dynamic reservoir are proposed. The design of the fitness function combines two objectives to be minimised, the output error and connectivity probability of the ESN's dynamic reservoir. Also,

a varying penalty term is added to the fitness function in order to handle the appearance of any infeasible solution. Four examples of complex dynamical systems from different domains are selected to provide diversity in the complexity of both the models' structures and different behavioural patterns to validate the performance of the proposed methodology.

Different types of analysis are applied using statistical tests for significance and selecting the best models. The experimental results show the capability of the proposed methodology to learn sparse models similar to target structures and generate similar behaviours. with the optimised ESNs outperforming the base one. The GA and PSO demonstrate the best results for all the different setups for finding the best learned structures based on their output and structural errors, respectively. Considering both these types of errors, the GA produces the best results over all setups for all examples. The proposed methodology is considered a promising approach that could be applied to modelling different complex systems when relying on building conceptual models as the first step in building a whole model.

Using other types of EAs and different penalty mechanisms to handle more challenging complex dynamical systems examples to learn structures that exactly match the target structures could be an extension of this work.

References

[1] J. D. Sterman, Business Dynamics: Systems Thinking and Modeling for a Complex World, vol. 19. Irwin/McGraw-Hill Boston, 2000.

[2] F. C. Billari, Agent-based computational modelling: applications in demography, social, economic and environmental sciences. Taylor & Francis, 2006.

[3] R. A. Howard and J. E. Matheson, Influence diagrams, Decis. Anal., vol. 2, no. 3, pp. 127–143, 2005.

[4] F.-R. Lin, M.-C. Yang, and Y.-H. Pai, A generic structure for business process modeling, Bus. Process Manag. J., vol. 8, no. 1, pp. 19–41, 2002.

[5] L. Schruben, Simulation modeling with event graphs, Commun. ACM, vol. 26, no. 11, pp. 957–963, 1983.

[6] S. Robinson, Simulation: the practice of model development and use. Palgrave Macmillan, 2014.

[7] J. Ryan and C. Heavey, Requirements gathering for simulation, in Proceedings of the 3rd Operational Research Society Simulation Workshop. The Operational Research Society, Birmingham, UK, 175-184, 2006.

[8] A. Medina-Borja and K. S. Pasupathy, Uncovering complex relationships in system dynamics modeling: Exploring the use of CART, CHAID and SEM, in Proceedings of the 25th International Conference of the System Dynamics Society, (Boston, USA), pp. 1–24, 2007.

[9] V. Quiñones-Avila and A. Medina-Borja, Universal healthcare: key behavioural factors affecting providers and recipients value propositions: a structural causal model of the puerto rico experience, Int. J. of Behav. and Hlthc. Res., vol. 3, no. 1, pp. 25–45, 2012.

[10] M. Drobek, W. Gilani, T. Molka, and D. Soban, Automated equation formulation for causal loop diagrams, Lecture Notes in Business Information Processing, vol. 208, pp. 38–49, 2015.

[11] E. Pruyt, S. Cunningham, J. Kwakkel, and J. De Bruijn, From data-poor to data-rich: system dynamics in the era of big data, in Proceedings of the 32nd International Conference of the System Dynamics Society, Delft, The Netherlands, 20-24 July 2014.

[12] H. Jaeger, The 'echo state' approach to analysing and training recurrent neural networks-with an erratum note, Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, vol. 148, p. 34, 2001.

[13] H. Abdelbari and K. Shafi, Learning causal loop diagram-like structures for system dynamics modeling using echo state networks, Syst. Dynam. Rev. - In Press, 2017.

[14] D. E. Goldberg, Genetic algorithms. Pearson Education India, 2006.

[15] R. Storn and K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Global. Optim., vol. 11, no. 4, pp. 341–359, 1997.

[16] J. Kennedy, Particle swarm optimization, in Encyclopedia of machine learning, pp. 760–766, Springer, 2011.

[17] Z. Wang, J. Zhang, J. Ren, and M. N. Aslam, A geometric singular perturbation approach for planar stationary shock waves, Physica D, vol. 310, pp. 19–36, 2015.

[18] C. K. Jones, R. Marangell, P. D. Miller, and R. G. Plaza, On the stability analysis of periodic sine–gordon traveling waves, Physica D, vol. 251, pp. 63–74, 2013.

[19] V. V. Gursky, J. Reinitz, and A. M. Samsonov, How gap genes make their domains: An analytical study based on data driven approximations, Chaos, vol. 11, no. 1, pp. 132–141, 2001.

[20] P. Young, Data-based mechanistic modelling of environmental, ecological, economic and engineering systems, Environ. Modell. Softw., vol. 13, no. 2, pp. 105–122, 1998.

[21] Y. Zhao, T. Weng, and M. Small, Response of the parameters of a neural network to pseudoperiodic time series, Physica D, vol. 268, pp. 79–90, 2014.

[22] Y. Feng, Y. Liu, X. Tong, M. Liu, and S. Deng, Modeling dynamic urban growth using cellular automata and particle swarm optimization rules, Landscape Urban Plan., vol. 102, no. 3, pp. 188–196, 2011.

[23] N. Petrov and A. Gegov, Model optimization for complex systems using fuzzy networks theory, in Proceedings of the 8th WSEAS international conference on Artificial intelligence, knowledge engineering and data bases, pp. 116–121, World Scientific and Engineering Academy and Society (WSEAS), 2009.

[24] I. M. Greca and M. A. Moreira, Mental models, conceptual models, and modelling, Int. J. Sci. Educ, vol. 22, no. 1, pp. 1–11, 2000.

[25] J. D. Sterman, Systems dynamics modeling: tools for learning in a complex world, IEEE Eng. Manag. Rev., vol. 30, no. 1, pp. 42–42, 2002.

[26] G. Desthieux, F. Joerin, and M. Lebreton, Ulysse: a qualitative tool for eliciting mental models of complex systems, Syst. Dynam. Rev., vol. 26, no. 2, pp. 163–192, 2010.

[27] K.-i. Funahashi and Y. Nakamura, Approximation of dynamical systems by continuous time recurrent neural networks, Neural networks, vol. 6, no. 6, pp. 801–806, 1993.

[28] H. Jaeger, Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach, Tech. Rep. 159, Fraunhofer Institute for Autonomous Intelligent Systems (AIS), 2002b.

[29] D. Koryakin, J. Lohmann, and M. V. Butz, Balanced echo state networks, Neural Networks, vol. 36, pp. 35–45, 2012.

[30] I. B. Yildiz, H. Jaeger, and S. J. Kiebel, Re-visiting the echo state property, Neural networks, vol. 35, pp. 1–9, 2012.

[31] M. Lukoševičius, A practical guide to applying echo state networks, in Neural Networks: Tricks of the Trade, pp. 659–686, Springer, 2012.

[32] C. E. Martin and J. A. Reggia, Fusing swarm intelligence and self-assembly for optimizing echo state networks, Comput. Intell. Neurosci., vol. 2015, p. 9, 2015.

[33] A. A. Ferreira and T. B. Ludermir, Comparing evolutionary methods for reservoir computing pretraining, in Proceedings of the 2011 International Joint Conference on Neural Networks, San Jose, California, USA, pp. 283–290, July 31 - August 5 2011.

[34] A. Deihimi and A. Solat, optimised echo state networks using a big bang–big crunch algorithm for distance protection of series-compensated transmission lines, Int. J. Elec. Power., vol. 54, pp. 408–424, 2014.

[35] A. A. Ferreira, T. B. Ludermir, and R. R. De Aquino, An approach to reservoir computing design and training, Expert. Syst. Appl., vol. 40, no. 10, pp. 4172–4182, 2013.

[36] D. Liu, J. Wang, and H. Wang, Short-term wind speed forecasting based on spectral clustering and optimised echo state networks, Renew. Energ., vol. 78, pp. 599–608, 2015.

[37] J. L. Gross and J. Yellen, Handbook of graph theory. CRC press, 2004.

[38] R. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput., vol. 1, no. 2, pp. 146–160, 1972.

[39] V. Petridis, S. Kazarlis, and A. Bakirtzis, Varying fitness functions in genetic algorithm constrained optimization: the cutting stock and unit commitment problems, IEEE Trans. Syst., Man, Cybern., Part B: Cybern., vol. 28, no. 5, pp. 629–640, 1998.

[40] A. E. Smith and D. M. Tate, Genetic optimization using a penalty function, in Proceedings of the 5th international conference on genetic algorithms, pp. 499–505, Morgan Kaufmann Publishers Inc., 1993.

[41] K. Langfield-Smith and A. Wirth, Measuring differences between cognitive maps, J. Oper. Res. Soc., pp. 1135–1150, 1992.

[42] Y.-C. Chuang, C.-T. Chen, and C. Hwang, A simple and efficient real-coded genetic algorithm for constrained optimization, Appl. Soft. Comput., vol. 38, pp. 87–105, 2016.

[43] J. Lane, A. Engelbrecht, and J. Gain, Particle swarm optimization with spatially meaningful neighbours, in Swarm Intelligence Symposium, 2008. SIS 2008. IEEE, pp. 1–8, IEEE, 2008.

[44] R. C. Eberhart and Y. Shi, Comparing inertia weights and constriction factors in particle swarm optimization, in Proceedings of the 2000 Congress on Evolutionary Computation, vol. 1, pp. 84–88, IEEE, 2000.

[45] S. N. Grösser and M. Schaffernicht, Mental models of dynamic systems: taking stock and looking ahead, Syst. Dynam. Rev., vol. 28, no. 1, pp. 46–68, 2012.

[46] E. M. Aylward, P. A. Parrilo, and J.-J. E. Slotine, Stability and robustness analysis of nonlinear systems via contraction metrics and sos programming, Automatica, vol. 44, no. 8, pp. 2163–2170, 2008.

[47] M. Rafferty, Butterflies and buffers, in Proceedings of the 27th International Conference of the System Dynamics Society, Albuquerque, Mexico, USA, July 26-30 2009.

[48] E. Theodorsson-Norheim, Friedman and quade tests: Basic computer program to perform nonparametric two-way analysis of variance and multiple comparisons on ranks of several related samples, Comput. Biol. Med., vol. 17, no. 2, pp. 85–99, 1987.

[49] M. R. Stoline, The status of multiple comparisons: simultaneous estimation of all pairwise comparisons in one-way anova designs, Am. Stat., vol. 35, no. 3, pp. 134–141, 1981.

[50] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput., vol. 6, no. 2, pp. 182–197, 2002.

Hassan Abdelbari received a B.Sc. in 2006 and a M.Sc. in 2010, both in computer science from Faculty of Computers and Informatics, Zagazig University, Egypt. He is currently working in his PhD at School of Engineering and Information Technology at University of New South Wales, Canberra, Australia. His PhD research focuses on supporting and automating the complex systems modelling process, system dynamics modelling process in specific, by using methods from computational intelligence field. His research interests focus on complex systems modelling and simulation, system dynamics, agent based modelling, neural networks, evolutionary computation, and computational intelligence.

Dr. Shafi holds a PhD in computer science, a M.Sc. in telecoms engineering and a B.Sc. in electrical engineering. His research focuses on the development of computational intelligence techniques that can be applied at various stages of data-centric predictive modelling in order to provide effective solutions to real world decision problems in diverse domains including national defence, logistics and computer security. In this context, he has contributed in several disciplines including genetic-based machine learning, game theory and optimisation. He is currently a Lecturer in the School of Engineering and Information Technology at UNSW, Canberra. He also provides consultancy to different Australian Government Departments in the area of data science and modelling and simulation in order to support business planning.