Evolutionary Echo State Network: A neuroevolutionary framework for time series prediction

Sebastián Basterrech, Gerardo Rubino

Evolutionary Echo State Network: a neuroevolutionary framework for time series prediction

Sebastián Basterrech[1], Gerardo Rubino[1]

[a]*Department of Computer Science, Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, 17. listopadu 2172/15, Ostrava, 70833, Czech Republic*
[b]*INRIA Rennes–Bretagne Atlantique, Rennes, France*

Abstract

From one side, Evolutionary Algorithms have enabled enormous progress over the last years in the optimization field. They have been applied to a variety of problems, including optimization of Neural Networks' architectures. On the other side, the Echo State Network (ESN) model has become increasingly popular in time series prediction, for instance when modeling chaotic sequences. The network has numerous hidden neurons forming a recurrent topology, so-called *reservoir*, which is fixed during the learning process. Initial reservoir design has mostly been made by human experts; as a consequence, it is prone to errors and bias, and it is a time consuming task.

In this paper, we introduce an automatic general neuroevolutionary framework for ESNs, on which we develop a computational tool for evolving reservoirs, called EVOlutionary Echo State Network (EvoESN). To increase efficiency, we represent the large matrix of reservoir weights in the Fourier space, where we perform the evolutionary search strategy. This frequency space has major advantages compared with the original weight space. After updating the Fourier coefficients, we go back to the weight space and per-

form a conventional training phase for full setting the reservoir architecture. We analyze the evolutionary search employing genetic algorithms and particle swarm optimization, obtaining promising results with the latter over three well-known chaotic time series. The proposed framework leads fast to very good results compared with modern ESN models. Hence, this contribution positions an important family of recurrent systems in the promising neuroevolutionary domain.
Keywords: Neuroevolution, Echo State Networks, Evolutionary Algorithms, Reservoir Computing, Fourier Transform, Swarm Optimization
2000 MSC: 37M10 sep 92B20, 82C32, 65T50, 37N99, 78M50, 37M25

1. Introduction

Recurrent network topologies can be used to incorporate temporal information in the training of Neural Networks (NNs). They are parametric dynamical systems with abilities for learning and memorizing input sequences [**?** **?**]. However, Recurrent Neural Networks (RNNs) have often come at the expense of much longer training times. They are complex and difficult to be properly trained, particularly when the network is large, and when it is necessary to learn long-term data dependencies [**?**]. Since 2001, a type of RNNs has been investigated under the name of Reservoir Computing (RC), and it has captured considerable attention due to its outstanding results for modeling chaotic systems and in solving real-world problems [**?** **?** **?**]. RC models overcome the important limitation of training procedures while introducing no significant disadvantages for time series processing applications. In spite of its simplicity, classic RC methods sometimes outperformed

on certain tasks other well-established recurrent systems as Long Short-term Memory (LSTM) networks [**?**]. The RC paradigm has been developed from three main concepts [**?** **?**]: Liquid State Machine (LSM), Echo State Network (ESN) and Backpropagation-decorrelation (BDPC). These three models share similar concepts regarding both the network design and the training approach. They have a nonlinear parametric dynamical system (i.e. an RNN with fixed weights), called *reservoir*, which has two main roles: feature extraction and memorization of input sequence. The learning protocol is restricted to adjust a simple parametric function of the output weights only, leading to a fast training procedure. The readout structure, i.e. the mapping between the extracted features at the reservoir to the output, is memory-free (often just a linear regression). The approach relies on the assumption that the reservoir is able to memorize historical data, and the readout structure is enough for designing an efficient learning model.

Recent developments in machine learning have also provoked a rising interest in evolutionary algorithms (EAs) as optimization tools for NNs [**?**]. Evolutionary optimization techniques have also been applied for optimizing recurrent architectures. Many early works have used EAs for finding optimal RC hyperparameters [**?** **?** **?** **?**]. Likewise, EAs have also been employed for evolving recurrent topologies. A combination of Particle Swarm Optimization (PSO) and Extreme Learning Machine (ELM) was studied in [**?**], where the optimization is done over a small region of the searching space. Following the ELM and RC principles, the rest of the weights are fixed and don't participate in the network tuning. Recently, Neat and HyperNEAT methods were used for evolving the reservoirs in [**?** **?**]. In addition, EAs

have also been employed for optimizing the hyperparameters of multi-layered stacked reservoirs, which are models with reservoirs organized in a cascade scheme [**?** **?** **?**].

Even though RC methods have accomplished very good performances and wide recognition, the RC area has also received some criticisms [**?** **?**]. The initial design of reservoirs is still often made by experience. Sometimes, even *brute force* is used, by exploring an arbitrary pre-defined region of the hyperparameters' space [**?** **?**]. Thus, it is computational expensive and requires numerous trails. Another known drawback is that different reservoirs with the same hyperparameters may produce significantly different results [**?**]. In addition, since the weights in the reservoir are initialized and remain fixed during the training process, the computational power of the recurrences is likely to be not fully harnessed [**?** **?**]. The model relies on the non-linear dynamics presented in the fixed initialized reservoir [**?**].

Motivated by the above discussion, we aim to improve the potential of large randomly initialized and non-adaptable reservoirs using EAs. Genetic encoding is a significant decision at the moment of the EA design. Specially, if the optimization is made over large complex structures. An efficient indirect encoding for NNs was developed in [**?**]. The weights were represented using the coefficients of the Discrete Cosine Transform (DCT). The evolutionary search was conducted in the frequency-domain instead of in the weight space. A first glimpse into evolving reservoirs in the frequency domain was developed in our previous conference article [**?**], where the EVOlutionary Echo State Network (EvoESN) model was introduced. The EvoESN model is inspired by the works [**?** **?**], where the authors compress the network in the frequency

domain.

None of the above-mentioned papers that apply EAs in the RC area have worked in the Fourier space, they apply direct encodings that present limitations for optimizing large networks. A majority of the works apply EAs to the optimization of the hyperparameters, or to adjust the weights among hierarchical reservoirs. To the best of our knowledge, the EvoESN is the first method that combines the potential of the Fourier compression with the RC area, and fine-tunes the reservoir weights, avoiding the problems of the gradient-based methods.

The purpose of this paper is to improve the EvoENS model by means of a better EA component, to provide a deeper analysis of the model, and to analyze the evolutionary phase and the reservoir stability. We make the following threefold contributions:

(i) **Framework for evolving reservoirs in the frequency domain**. It is well known that the initialization of the recurrent weights impacts on the model's performance. Furthermore, it seems that to have fixed network weights (independently of the input data) does not totally employ the power of recurrent system. We study the integration of EAs for evolving the reservoirs in a compressed domain, and compare them with baseline modern ESN solutions.

(ii) **Exploration characteristics and stability control**. We present a systematic analysis of the framework hyperparameters. Furthermore, we analyze the evolution of the feasible solutions according to the main parameter related to the recurrences' stability. We also establish a

more general control framework that includes an analysis regarding the initialization of the EA population.

(iii) **Search strategy analysis**. We introduce a more exhaustive machine learning study. In our previous work [**?**], the fine-tuning of the reservoir weights was done using Genetic Algorithms (GAs). In this paper, we also discuss the improvement obtained by the use of the PSO algorithm. PSO is a population-based optimization method well-known for its achieved good results in problems with continuous and high-dimensional fitness landscape [**? ?**].

The experimental results show the improvements obtained by the new EvoESN with respect to the initial version. The method finds well-performing reservoirs quickly, and achieves also significant improvements compared with other state-of-the-art recurrent networks. Experiments were made over two widely-investigated chaotic dynamical systems: the Mackey Glass system and the Lorenz system [**? ? ? ? ? ? ?**]. In addition, we present results over a real-world data set entitled Sunspot series [**?**], that has been also widely studied in the RC domain.

The outline of the article is the following. In the next section, we briefly describe the necessary background related to recurrent systems, and review previous work about EAs over RNNs. Section **??** introduces the ESN model, its properties and modern variations. The main contribution (the EvoESN model) is presented in Section **??**. Experimental results are analyzed in Section **??**. A final discussion provides a summary of the limitations and advantages of the framework presented in this paper, and discuss future research directions.

2. Preliminaries

2.1. Problem characterization

Recurrent systems have proven to be effective for solving learning problems where the ordering of data matters, such as time-series forecasting, time-series classification, sequence prediction and generative modeling. Those tasks are often grouped under the global concept of Temporal Learning [**?**], where the learning data is composed of dependent instances, e.g. $\mathbf{u}(t)$ depends on $\mathbf{u}(t \pm \delta)$, $\delta \leq \Delta$, for some non-large Δ value. In contrast, non-temporal problems have instances that are independent of each other.

Let us consider a supervised context where the dataset is composed of N input-output pairs $(\mathbf{u}, \mathbf{y})$; the input space has dimension n and the output space has dimension o. For simplicity, we consider sequences indexed in discrete spaces. The goal in Temporal Learning, that is the most common application of ESNs, is to predict a future value of a sequence (or input signal) given the present value and some values in the past; for instance, given a sequence $\mathbf{u}(t-k), \ldots, \mathbf{u}(t-1), \mathbf{u}(t)$, the goal could be to predict the output $\mathbf{y}(t+\delta)$, with $\delta \geq 0$.

The learning process has an initial training phase where the network is driven by external data, and the network weights are trained solving an optimization problem. Then, an autonomous phase is performed where the input instances are provided by the model itself using previous predicted values [**?**]. In the domain of time series prediction, the autonomous phase often appears under the name of free-run prediction.

2.2. Motivations

There are many variations of RNN architectures, where the recurrences have another form; for more information we suggest [**?**]. However, training recurrent systems is still difficult [**?** **?**]. Classic RNNs, LSTMs or Transformers can be expensive to train, and often requires numerous samples to reach a satisfactory accuracy [**?**]. Moreover, learning algorithms based on gradient and curvature information for training recurrent networks have well-identified technical issues [**?** **?**]. For instance, first-order algorithms (based on the gradient information) may suffer from the vanishing and exploding gradient problem [**?**]. Second order techniques (based on the curvature information) may present scalability issues; for some of them, there is a quadratic relationship between the size of the Hessian matrix and the number of weights [**?**]. Large networks have additional inconveniences, the most common mentioned in the literature are: not guaranteed convergence, large training times, and storage problems [**?** **?** **?**]. The RC paradigm has emerged as a feasible option of using recurrent networks, avoiding the well-known issues on the optimization of the recurrent networks [**?**].

2.3. Neuroevolution: combining the power of neural nets with evolutionary algorithms

Evolutionary computation algorithms have been used for a long time for evolving architectures as well as for finding the network wights [**?** **?**]. Neuroevolution is a nature-inspired computational-matematical model that combines the potential of both NNs with EAs [**?**]. Evolutionary approaches in specific types of large–scale problems outperform gradient descent optimization [**?**]. Recently, more evolutionary techniques have been developed,

specifically for optimizing large NNs [? ? ?]. Neuroevolution provides a general framework for the joint optimization of the network structure, the topology, and the parameters related to activation function and other characteristics of the network [? ?]. There is a wide variety of evolutionary approaches in the neuroevolution field [? ?]. From a global viewpoint, the network structure and weights can be evolved either simultaneously or separately. Looking at the network components, every neuron can be either optimized independently or complete layers can be optimized at the same logical time. A breaking point in the neuroevolutionary research was the introduction of the neuroevolution of augmenting topologies (NEAT) method [?]. The technique starts with a simple architecture and increases the edge connections over time. It has obtained good performances in embedded systems.

2.4. Encoding network architectures

A recurrent topology may produce a significant impact on the model's performance [? ?]. Moreover, the evolution of recurrent systems has additional difficulties. A recurrent structure (formally, a dynamical system) must manage the additional temporal parameter, and the space-complexity of the encoding is really challenging [? ?]. The architecture design predominantly remains driven by randomized and heuristic approaches [?]. Another relevant problem in neuroevolution refers to the evolutionary process itself: how to combine single individuals? [?]. Naive crossover operations are not really suitable for combining two recurrent structures [? ?], they can even impact negatively on the model's performance. Some authors directly avoid crossovers and pay attention only to the selection process [?]. This fact can

be attributed to several reasons such as the invariance of neurons to permutation (i.e., the *competing convention* problem) [**?**], the semantic correctness of the operations, the case of large search spaces, and the high sensitivity issues (small perturbations on single networks can produce significant different results) [**?**].

Some of the before mentioned issues have been tackled by recent advances in the field of neuroevolution. Controlled weight perturbation using output gradient information was introduced in [**?**]. The approach effectively increases the ability of even simple genetic algorithms to evolve neural structures. Similarly, safe operations (controlled crossover operations) were introduced in [**?**]. Augmenting topologies has produced an important advance in the area. However, a significant drawback is that it uses a direct encoding, implying that it is not scalable to large networks [**?** **?**]. Overall, neuroevolution over deep RNNs has become a hot research topic [**?** **?**]. When using a direct encoding (the candidate solution represents the entire network), traditional algorithms such as GNARL or NEAT struggle with the complexity of the problem [**?** **?**]. Although modified and indirect methods such as HyperNEAT [**?**] and coDeepNeat [**?**] overcome some of these initial limitations, new methods are required to more efficiently encode and evolve large recurrent architectures, in order to solve large-scale real–world problems [**?** **?** **?**]. In Section **??** we discuss a recent introduced indirect encoding based on Fourier transforms, which is the one applied in our EvoESN model.

3. Echo State Networks

3.1. Model specification

We first formalize the standard ESN model that probably is still the most popular in the RC area. Let's consider a network with n input neurons, p hidden neurons and o output neurons. The dynamics of the model, as a function of the discrete time variable t, has the form [**?**]:

$$\mathbf{x}(t) = f(\mathbf{W}^{\text{in}}\mathbf{u}(t) + \mathbf{W}^{\text{h}}\mathbf{x}(t-1)), \tag{1}$$

where $\mathbf{W}^{\text{in}}$ is a $p \times n$ matrix with the input-to-hidden weights and the $p \times p$ matrix $\mathbf{W}^{\text{h}}$ has the hidden-to-hidden weights (reservoir weights). The model output $\mathbf{y}(t) \in \mathbb{R}^o$ is computed by the readout layer

$$\mathbf{y}(t) = g(\mathbf{W}^{\text{out}}[\mathbf{u}(t); \mathbf{x}(t)]), \tag{2}$$

where $\mathbf{W}^{\text{h}}$ is the reservoir-to-output weight matrix, $g(\cdot)$ is a predefined coordinate-wise function and $[\,\cdot\,;\,\cdot\,]$ denotes vector concatenation. For simplicity, bias terms on previous expressions are omitted on the equations, a common practice since it is straightforward to implement them by adding dummy terms. The transition function $f(\cdot)$ must satisfy some properties in order to perform stable recurrent updates. The function should be 1-Lipshitz continuous (i.e. continuous, differentiable and bounded gradient, transcendental and monotonically increasing) [**?** **?**]. In the case of the standard ESN, the widely used transition function is the hyperbolic tangent $\tanh(\cdot)$, and the readout expression (**??**) is a linear model.

As we already mentioned, a distinctive characteristic of the ESN model is that the reservoir weights (matrix $\mathbf{W}^{\text{h}}$) are frozen during the learning

phase. Therefore, the transition function is fixed for all the instances; for this reason, (**??**) is sometimes referred to as static random projection. Another common word for expression (**??**) is *expansion function*, because the dimension of the reservoir must be much larger than the dimension of the input space [**?**]. The readout parameters ($\mathbf{W}^{\text{out}}$) can be learned using the regression (**??**) and applying a classical tool such as Tikhonov regularization [**?**].

The reservoir initialization may have an impact on the model's performance [**?** **?**]. In addition, there are several hyperparameters that are relevant in the robustness of the ESN [**?**]. The *reservoir size* defines the dimensionality of the reservoir projection. Large reservoirs may create meaningful feature subspaces and improve the separability of the input data. However, to increase the complexity may provoke overfitting. The *input scaling factor*, as its name specifies, defines the domain of the input weight matrix. Therefore, its value may impact on the recurrences, and then modify the reservoir state trajectories. The *neuron activation functions* impact in the ability to memorize the input sequence, as well as in the linear separability of the input data in the feature space. The most commonly used function type is the hyperbolic tangent [**?** **?**]. Another hyperparameter is the *sparsity* of the reservoir weight matrix. The RC literature suggests using a sparse reservoir weight matrix, with about 15% to 30% of non–zero values [**?** **?**]. Sparsity doesn't significantly affect the model's accuracy [**?**]. However, it naturally has a strong impact on the computational costs: a sparse reservoir matrix enables fast reservoir state updates. The most common readout mapping is the linear regression, but other procedures have been used such as Support Vec-

tor Machines, single layer networks or PCA [**?**]. The pattern of connectivity is another important aspect at the moment of designing an initial reservoir. Reservoirs with specific cyclic jumps are analyzed in [**?**]. In particular, the impact of the degrees of the reservoir nodes (graph theory meaning) on the global model accuracy [**?**]. Multilayer readouts and hierarchical reservoirs have been developed during the last years [**?**]. This is a new promising area inside the RC family. Hierarchical reservoirs, sometimes referred to as deep reservoir architectures, are a kind of cascade of reservoir clusters with connections between them [**? ? ?**]. The simplicity and efficiency of the RC computational schema has made that several works in the community have addressed the problem of implementing these architectures in hardware [**? ? ?**].

3.2. Recurrence stability

Several studies analyze the stability conditions and asymptotic properties of the recurrences of expression (**??**) [**? ? ? ?**]. A structural and fundamental property is the *Echo State Property* (ESP). The ESP guarantees that the reservoir state becomes asymptotically independent of the initial conditions [**?**]. Even if the initial trajectories are strongly random, the reservoir projections should be independent of them in the long term. Additionally, the network should satisfy a type of "functional" relationship where each input sequence is associated with a single output sequence in the long term. These two properties are established in the ESP [**?**].

The spectral radius and the singular values of the reservoir matrix have an important impact on the ESP [**? ? ?**]. Previous mentioned studies and experimental works suggest building reservoirs that hold the ESP,

and a simple way to do it consists of scaling the reservoir weights using the spectral radius of the reservoir matrix [**?** **?**]. Let us denote the spectral radius of the $\mathbf{W}^{\mathrm{h}}$ matrix by $\rho(\mathbf{W}^{\mathrm{h}})$. The common procedure for defining the initial reservoir is as follows: (i) random initialization of the reservoir weights $\mathbf{W}^{\mathrm{h}}_{(\mathrm{rand})}$, (ii) computation of its spectral radius $\rho(\mathbf{W}^{\mathrm{h}}_{(\mathrm{rand})})$, (iii) scaling operation $\mathbf{W}^{\mathrm{h}} := \alpha\mathbf{W}^{\mathrm{h}}_{(\mathrm{rand})}/\rho(\mathbf{W}^{\mathrm{h}})$, where α is a parameter in $[0, 1]$. Using this initialization, the final matrix will have a spectral radius equal to the predefined α value. Actually, pathological situations were analyzed by Jaeger, and it is enough to properly scale the reservoir weights to almost always satisfy the ESP [**?**]. Moreover, the property is violated in autonomous networks without external entries if $\rho(\mathbf{W}^{\mathrm{h}}) > 1$ using a convex activation function [**?**]. In this particular case, the reservoir state may present oscillations or a chaotic behavior [**?** **?**]. Also, the property may be unsatisfied when the spectral radius is less than 1, but it corresponds to unusual and strange topologies (not used in practice) [**?**]. It has been found that optimal computational performance of the reservoir units operates in a regime that lies between stable and chaotic behavior [**?** **?** **?**].

3.3. Modern ESN variations

In recent years, several variations of the canonical ESN have emerged. We integrated some of them in our experimental results.

Feedback connections [**?**]. The recurrent state is computed by

$$\mathbf{x}(t) = f(\mathbf{W}^{\mathrm{in}}\mathbf{u}(t) + \mathbf{W}^{\mathrm{h}}\mathbf{x}(t-1) + \mathbf{W}^{\mathrm{fb}}\mathbf{y}(t)), \tag{3}$$

where $\mathbf{W}^{\mathrm{fb}}$ is a weight $p \times o$ matrix with the feedback connections going from the output neurons to the hidden structure. The feedback can also

be integrated with an arbitrary delay. When the model has feedbacks, the reservoir state is driven by both the input signal and the model output, and this increases the computational power of the neural nets [**?**]. Nevertheless, stability issues may arise with feedback connections [**?**].

How to train feedback weights? In the case of models with feedback, a common strategy called *teacher-forcing* is usually applied for training the readout weights [**?** **?** **?**]. The schema is as follows. During the training phase, the feedback weights are teacher-forced, i.e. instead of feeding back the reservoir with the predicted value $\mathbf{y}(t)$, the real target is used as the feedback value [**?**]. In exploitation mode, a free-run prediction is realized, where the prediction $\mathbf{y}(t)$ is fed back to the reservoir.

Additional noise [**?**]. The reservoir state with injected noise has the form:

$$\mathbf{x}(t) = f(\mathbf{W}^{\text{in}}\mathbf{u}(t) + \mathbf{W}^{\text{h}}\mathbf{x}(t-1) + \nu(t)), \tag{4}$$

where a tiny noise $\nu(t)$ is incorporated during the training procedures. To add noise is a common regularization strategy for increasing the generalization capability of the model. The noise is injected only during the training phase, in exploration mode the noise parameter $\nu(t)$ is set to zero.

Leaky integrator ESN. The reservoir state variation is smoothed as follows:

$$\mathbf{x}(t) = (1-\gamma)\mathbf{x}(t-1) + \gamma\phi(\mathbf{W}^{\text{in}}\mathbf{u}(t) + \mathbf{W}^{\text{h}}\mathbf{x}(t-1)), \tag{5}$$

where $\gamma \in (0, 1]$ is the *leaking rate* parameter [**?**]. It controls the sensitivity over the entry values, and has an impact on the "speed" of changes in the trajectories [**?**].

4. EvoESN: Evolutionary Echo State Networks

4.1. Discrete Cosine Transforms

The Discrete Cosine Transform (DCT) satisfies properties that make this mapping suitable for encoding large matrices [**?** **?** **?**]. We use one of the most common versions, the normalized DCT of Type II): given an arbitrary sequence of H reals $s = (s_0, s_1, \ldots, s_{H-1})$, its DCT is $\mathrm{DCT(s)} = \mathrm{S} = (\mathrm{S_0}, \mathrm{S_1}, \ldots, \mathrm{S_{H-1}})$, where $S(0) = \sqrt{1/H} \sum_{h=0}^{H-1} s_h$ and

$$S_\ell = \sqrt{\frac{2}{H}} \sum_{h=0}^{H-1} s_h \cos\Big(\frac{\pi \ell (2h+1)}{2J}\Big),$$

for $1 \leq \ell \leq H-1$. This transformation has an inverse, denoted IDCT here, that is, with previous notation, $\mathrm{IDCT(S)} = \mathrm{s}$, where

$$s_\ell = \frac{S_0}{\sqrt{H}} + \sqrt{\frac{2}{H}} \sum_{h=1}^{H-1} S_h \cos\Big(\frac{\pi h (2\ell+1)}{2H}\Big).$$

Our proposed approach EvoESN relies on the bijectivity and the energy compaction properties of the DCT. The first point is an immediate consequence of the linearity of the DCT and the IDCT. So, they are actually C^∞ functions from $\mathbb{R}^H$ onto $\mathbb{R}^H$. The second property means, informally, that we can choose some $K < H$ (typically, $K \ll H$) such that if S' is built by keeping the first K elements of $S = DCT(s)$ only (called the *low frequency* terms), replacing the rest by 0s, then $\mathrm{IDCT}(S')$ is pretty close to s.

4.2. Evolutionary algorithms on modern ESN models

The survey article [**?**] covers the first decade of the RC area, including some initial literature regarding evolutionary methods for optimizing reservoirs. More recently, evolutionary algorithms have been used as hyperparameter optimization in the RC area. Both PSO and GAs have been used in [**?**

? ? ? ?]. Authors employed EA for tuning global parameters such as the number of reservoir units and the scaling factors of connectivity properties (density and spectral radius of the reservoir weights matrix).

GAs for finding the global parameters of an extended ESN for modeling medical signal prognosis were studied in [?]. A pre-training algorithm based on swarm optimization was developed in [?]. The authors applied PSO for adjusting a subset of the reservoir weights. Instead of searching the global parameters, the proposed technique works directly on a subset of reservoir weights. In [? ?] the idea of growing RC topologies using Neat and HyperNEAT was investigated. In [?] the evolution of the reservoir topology was made considering a fitness function that combines both the memory capabilities and the model's accuracy. Evolutionary techniques were also applied for optimizing parameters in hierarchical reservoirs, for instance, for estimating the weight connections between two consecutive reservoirs in the hierarchy [? ?]. A variation of GA named microbial GA was used in [?]. In addition, the weight connections among the reservoirs are set in order to make a global contraction mapping (a kind of similar restriction than the ESP in classic ESN). Recently, PSO was introduced as a hybrid approach of swarm optimization and local search in hierarchical ESNs [?].

4.3. Indirect encoding in the Fourier space

In [? ?], an indirect encoding schema for NNs was introduced, where weight connections were represented, in a compressed form, in the frequency domain. Authors apply EAs where the individuals (chromosomes) are represented by the type-II DCT of the weight matrix. That is, the search space in the evolutionary phase is composed of vectors of Fourier-like series coef-

ficients. In parallel, in the original application the authors define a coevolutionary method, called CoSyNE, for searching in the space of the DCT coefficients. Evolino networks, based on an LSTM network, is another example where the network weights are represented in another space and the EAs approach is adopted for solving the optimization side [**?**]. This approach has been successfully applied over rather small fully connected RNNs on three benchmark problems of the area of control: pole balancing, ball throwing and octopus arm [**?**]. The authors have also applied this encoding procedure in other problems related to computer vision and Reinforcement Learning [**?**]. The encoding was recently re-visited in the NN domain [**?** **?**]. In [**?**], the authors performed a DCT encoding over a LSTM network, and the model is applied to solve language modelling problems. The EvoESN, introduced in [**?**], presents a fine-tuning procedure for optimizing the reservoir weights based on the indirect mapping DCT and on GAs [**?**].

4.4. Neuroevolutionary framework

In this section, we describe the EvoESN computational model, and describe its main characteristics and benefits.

Notation. We first choose the number M of (a priori) non-null weights in the reservoir. Reservoirs are commonly designed to be sparse, then we assume that $M \ll p$. We also choose which weights in the reservoir will be frozen to 0, by randomly choosing the remaining M positions among the total p^2 possible ones. We define two auxiliary vectors $\boldsymbol{\gamma}$ and $\boldsymbol{\nu}$ representing positions in $\mathbf{W}^{\mathrm{h}}$ (i.e. (row, column) pairs) and corresponding weights' values, respectively. If $\gamma_k = (i, j)$, we mean that the kth position in $\boldsymbol{\gamma}$ refers to the weight in row i and column j of $\mathbf{W}^{\mathrm{h}}$. The vector $\boldsymbol{\nu}$ (with dimension M) contains those

weights, that is, $\nu_k = \mathbf{W}^{\mathrm{h}}_{\gamma_k} = \mathbf{W}^{\mathrm{h}}_{i,j}$. During all the procedures, the positions (a kind of hash map) will remain fixed; only the corresponding weights will change following the evolutionary search. That is, $\boldsymbol{\gamma}$ remains fixed and $\boldsymbol{\nu}$ evolves during the execution of the proposed method. We denote by C the number of DCT coefficients to be used for representing $\boldsymbol{\nu}$. The integer C must satisfy $C \leq M$. Besides, we will denote by $\boldsymbol{\alpha}$ the vector with the first C coefficients $\alpha_1, \ldots, \alpha_C$ of the DCT of $\boldsymbol{\nu}$.

Coefficient interpretation. As in any application of Fourier-like transformations, the frequency used in α_i is lower than that in α_j when $i < j$, and lower frequency coefficients store more information than higher frequency ones. We say that the coefficient α_i is more significant than α_j when $i < j$. So, the higher C, the better in principle the accuracy of the application, but of course, the higher the computational cost.

Evolutionary search. It consists of moving the vector ν to the frequency domain using the DCT transform, to change it running an evolutionary procedure, to go back to weights through the inverse DCT, to evaluate the introduced changes by training the ESN with the just obtained reservoir, and to repeat this process until some convergence condition is satisfied. As we will see, we actually never need to perform a DCT, only its inverse is required in the algorithm.

The first step is to choose the initial values of the C coefficients. We discuss this initialization procedure in Subsection **??**. Denote now by $\phi(S)$ the inverse DCT transformation of sequence S (IDCT). If we write $s = \Phi(S)$, both sequences S and s have the same size. To refer to the evolutionary process used in [**?**], a genetic one, we will also call *chromosome* the DCT

of vector $\boldsymbol{\nu}$ (that is, vector $\boldsymbol{\alpha}$), and $\boldsymbol{\nu}$ itself will be called *phenotype*. In case $C < M$ (the usual one), we need to extend the chromosome $\boldsymbol{\alpha}$ with $M - C$ zeros (the operation is called *padding*) to obtain a vector $\phi(\boldsymbol{\alpha})$ of dimension M. The padding operation is a standard method used to increase the values in the frequency space. We also call *extended chromosome* the vector $\boldsymbol{\alpha}$ when $C < M$. Once we have possibly extended the chromosome $\boldsymbol{\alpha}$ and computed $\boldsymbol{\nu} = \Phi(\boldsymbol{\alpha})$, the ESN is trained as in the classic procedure. If the test step doesn't validate the training, we execute an evolutionary search in the frequency space, in order to obtain a new chromosome $\boldsymbol{\alpha}$, leading to a new phenotype $\boldsymbol{\nu} = \Phi(\boldsymbol{\alpha})$. The new ESN is evaluated. The whole procedure is repeated until some convergence condition is satisfied. Figure **??** visualizes the global procedure. The proposed framework has the following main operations:

(i) Initialization of the chromosomes (sometimes called particles, depending on the terminology used by the EA).

(ii) Padding operation: it is the chromosome extension, in which zero values are concatenated until dimension M is reached.

(iii) Indirect encoding: IDCT application.

(iv) Genotype-phenotype mapping: to assign the elements of the phenotype to their positions in the reservoir.

(v) Fitness computation: for each chromosome, training of the readout structure and ESN evaluation over testing data.

(vi) Evolutionary search: to define a protocol for changing from one generation to the next one.

Pseudocode description. We describe the proposed computational model in two pseudocodes: the evolutionary search is presented in Algorithm **??** and the fitness function computation for each candidate (particle/chromosome) is presented in Algorithm **??**. Notice, that the particle extended chromosome is first transformed into the original space of weights (through $\boldsymbol{\nu} = \phi(\boldsymbol{\alpha})$), and then, before training, it is possible to re-scale the reservoir weights (in order to control the ESP). This operation of re-scaling (line 8) is optional. In the experimental section, we present results with scaling and without scaling the reservoir weights.

4.5. Evolutionary search using PSO

In the introductory work about EvoESN presented in [**?**], the evolutionary search was performed using GAs. Here, we also incorporate the results of swarm optimization [**?**], because of the improvement they bring to the project. The PSO evolution is based on a simulation of swarming behavior of birds within a flock, as well as other groups of social animals [**?**]. At each iteration, the particles in the swarm exchange information about their positions. Each particle is represented by its position and velocity, and it can be interpreted as a multidimensional moving point with an associated velocity. The common notation in the literature for both variables are $\mathbf{x}$ and $\mathbf{v}$. Though we have already denoted the reservoir state with $\mathbf{x}$, in this brief PSO description we will anyway use the notation $\mathbf{x}$ for the particle position. In addition, each particle i remembers its best reached position $\mathbf{l}_i$ (in terms of

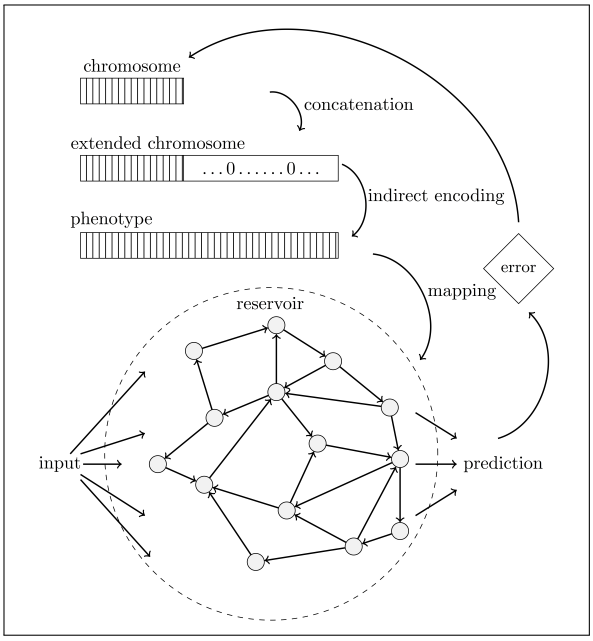

Figure 1: Visualization of the genotype-phenotype mapping of EvoESN. We see the chromosome (the vector of the DCT transform of the non-null weights in the reservoir), extended with zeros if necessary, then sent to the weight space as a new phenotype through the inverse DCT (we also call this *indirect encoding*), used by the ESN to make, after training, a prediction. The error is considered in the evolutionary process for computing the fitness value of each chromosome.

fitness function), and it knows the best multidimensional position discovered so far by the whole swarm, $\mathbf{g}$. The position represents the actual candidate problem solution, and the velocity determines regions in the searching space to be explored. Here, we use indistinctly the terms particle and chromosome. The points $\mathbf{l}_i$ and $\mathbf{g}$ are sometimes referred to as the best local position of i and the best global position of the swarm. In each iteration t, the velocity

of particle i is updated according to [**?**]:

$$\mathbf{v}_i^{(t+1)} = \iota \mathbf{v}_i^{(t)} + c_1 r_1^{(t)}(\mathbf{l}_i^{(t)} - \mathbf{x}_i^{(t)}) + c_2 r_2^{(t)}(\mathbf{g}^{(t)} - \mathbf{x}_i^{(t)}), \tag{6}$$

where ι is called inertia parameter, c_1 and c_2 are positive acceleration parameters and r_1 and r_2 are vectors of random values sampled from the uniform distribution. The inertia controls the direction of future explorations. The parameters c_1 and c_2 are called the *cognitive* and the *social* parameters, respectively. They control the trade-off between exploration and exploitation. Vector $\mathbf{l}_i^{(t)}$ represents the best position known by the particle i until iteration t, and vector $\mathbf{g}^{(t)}$ is the best position visited by the whole swarm also until t. The position of particle i is updated by [**?**]

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+1)}. \tag{7}$$

Due to its simplicity and good performances, there are many works regarding swarm optimization both in theory and in applications [**?** **?** **?**]. In the experimental section, we follow a suggestion of [**?**] for tuning the basic PSO parameters. We empirically analyze the inertia values using a grid search in $[0.5, 1]$ with the additional condition that $1 > \iota > 0.5(c_1 + c_2) - 1 \geq 0$. The trajectory of a particle converges if the previous condition is held [**?**]. In addition, we explore the searching space updating at each iteration t the parameters c_1 and c_2 in order to avoid stagnation as follows [**?**]:

$$c_1^{(t)} = (c_{1,min} - c_{1,max})\frac{t}{T} + c_{1,max} \quad \text{and} \quad \mathrm{c}_2^{(\mathrm{t})} = (\mathrm{c}_{2,\max} - \mathrm{c}_{2,\min})\frac{\mathrm{t}}{\mathrm{T}} + \mathrm{c}_{2,\min}, \tag{8}$$

where $c_{1,min}$, $c_{1,max}$, $c_{2,min}$ and $c_{2,max}$ are pre-selected constants (minimum and maximum values of c_1 and c_2), and T is the maximum number of iterations. In our problem, the exploration space is composed of the vectors of coefficients, living in the frequency domain.

4.6. How to properly initialize the chromosomes?

The IDCT operation transforms points from the frequency domain into the weight space. The weights should belong to an appropriate domain in order to avoid neuron saturation. Large weight values will provoke the neuron saturation phenomenon, where the reservoir state will predominantly be close to the asymptotic ends of the activation function (hyperbolic tangent). On the other side, excessively small weight values may slow down the training progress and incorporate some numerical issues.

In general, the initialization of the particles has a crucial impact on the global approach. Figure **??** shows the impact of the particle initialization. As an example, the particles have a dimension of 100, and there are 100000 non-zero values of reservoir weights. The three figures in the first column show the particle values. The swarm is initialized with different uniform distributions ($\mathcal{U}[0,1], \mathcal{U}[-1,1], \mathcal{U}[-4,4]$). The three graphics, in the second column, show the values of the obtained weights after performing the padding operation and the IDCT function. It is common to initialize the reservoir weights using a symmetric random initialization with distributions $\mathcal{U}[-0.2, 0.2]$ or $\mathcal{U}[-0.5, 0.5]$ [**?** **?**]. Therefore, in our experiments, we increase the initial diversity of the swarm with particle initialization using different uniform distributions. We initialized particles with elements in the range$[-a, a]$ with $a = 4, 5$ and 6.

5. Experimental analysis

We evaluate the EvoESN tool on three selected dynamical systems, and we compare with the standard ESN and its popular modern variations dis-

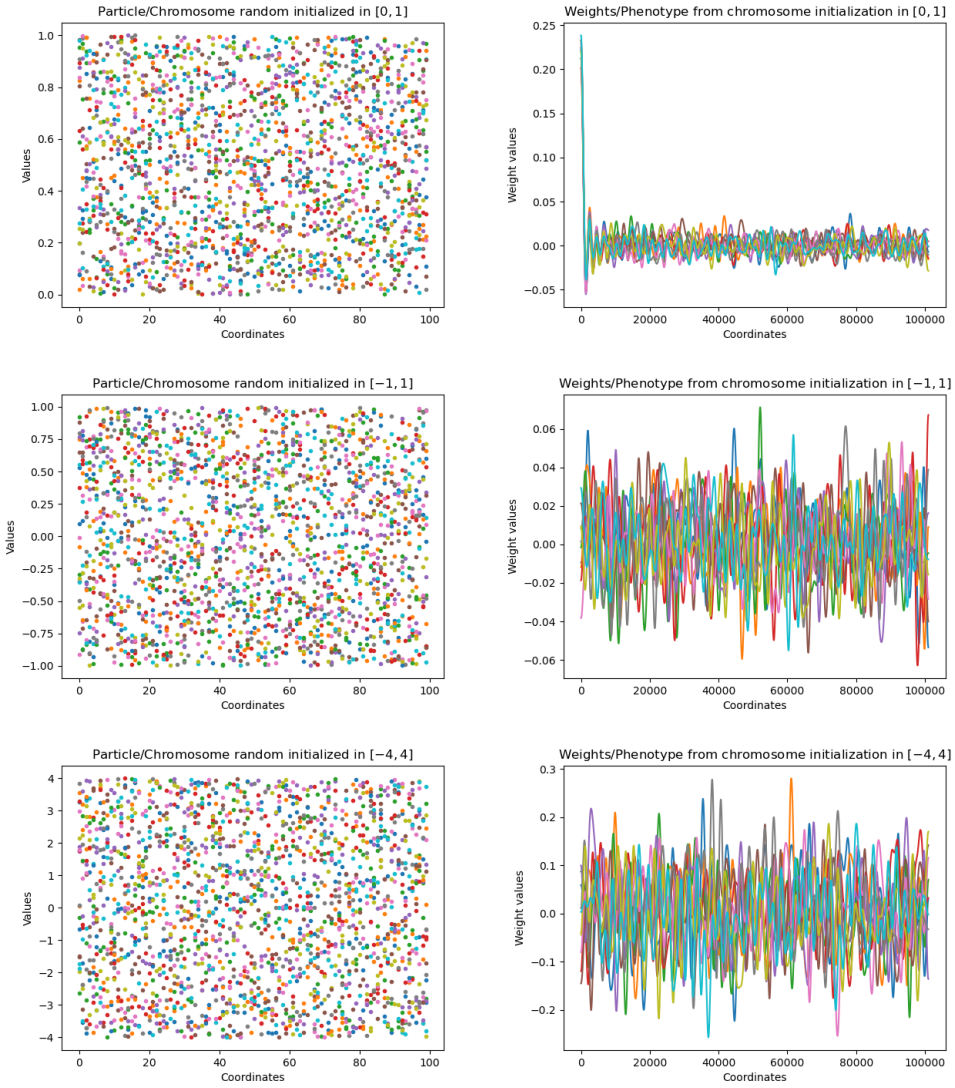

Figure 2: Impact of the initialization of the chromosomes on the weight space. The IDCT encoding projects the extended chromosome in the weight space. However, the domain of the weight space is relevant for avoiding neuron saturation. The figure shows how different uniform distributions obtain pretty different domains of the weight space.

Algorithm 1: Evolutionary search using PSO for reservoir weights in the Fourier space.

Inputs : Fitness function, learning data, PSO parameters, ESN parameters, network model.

Outputs: Best global particle.

`// Swarm initialization using Section ??.`

Initialization of $\{\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(I)}\}$;

`// Evaluation of each particle.`

Compute fitness for each $\boldsymbol{\alpha}^{(i)}$ using Algorithm **??**;

`// Apply evolutionary search.`

while (Convergence criterion is not satisfied) **do**

- Update particle velocity for all i using expression (**??**);
- Update particle position for all i using expression (**??**);
- Update inertia and acceleration parameters (ι, c_1, c_2) using expressions (**??**);
- Update local best for all i;
- Update global best;
- Compute fitness for each $\boldsymbol{\alpha}^{(i)}$ using Algorithm **??**;

Return best global particle;

Algorithm 2: Computation of the fitness value.

Inputs : Input particle, learning data, ESN parameters, network model.

Outputs: Fitness evaluation.

if $(C < M)$ **then**

    `// Chromosome/particle extension.`

    Padding operation for $\boldsymbol{\alpha}$;

`// Frequency to weight domain.`

$\boldsymbol{\nu} = \text{IDCT}(\boldsymbol{\alpha})$;

`// Filling reservoir.`

Assign $\boldsymbol{\nu}$ values to $\mathbf{W}^{\text{h}}$ according to positions in $\boldsymbol{\gamma}$;

`// Optional operation.`

Reservoir scaling to satisfy ESP;

`// ESN training.`

Train readout weights;

Return fitness error for the evaluated particle;

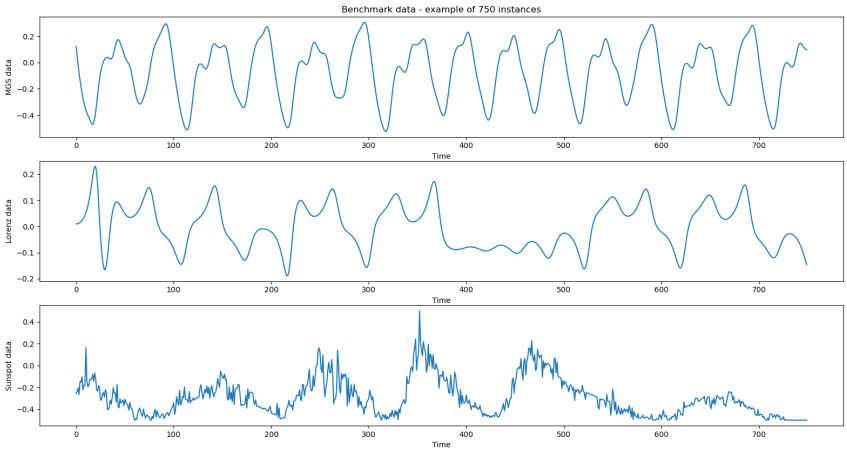

Figure 3: Visualization of 750 instances of the three investigated problems.

cussed in Section **??**. We used the Mackey-Glass system (MGS), the Lorenz system and the Monthly Sunspot index data, since those benchmark problems cover different types of chaotic time-series and have been largely studied in the RC community [**?** **?** **?** **?** **?**]. Lorenz and MGS problems have been tested in almost every nonlinear system modeling [**?**]. The Monthly Sunspot data has measurements of the magnetic field in the Sun's outer regions. The series is publicly available at [**?**]. Sunspot data has also been analyzed using RC models in at least [**?** **?** **?** **?**]. Figure **??** shows 750 instances of the three studied problems, where it is possible to see the diversity in the behavior of the three chaotic time series. For experimental reproducibility, benchmark datasets and source code are available in: `https://github.com/sebabaster/EvoESN`.

5.1. Benchmark problems

Theoretical data: Mackey-Glass and Lorenz systems. The standard ESN model acquired a wide recognition for their obtained success in forecasting the MGS data [**?**]. Actually, the record-breaking achieved accuracy in modeling MGS instances was obtained with one of the ESN variations presented in Section **??**, where the recurrent state operates with an injected noise and feedback connections. The MGS model is a type of nonlinear auto-regressive (NAR) dynamical system used for analyzing bifurcations in physiological applications. It is defined by the differential equation

$$\frac{\partial y}{\partial t} = \frac{\alpha y(t-\tau)}{1+y(t-\tau)^{\beta}} - \gamma y(t),$$

with parameters $\alpha, \beta,\ \gamma$ and τ. Despite its simplicity, this equation can generate periodic and complex chaotic dynamics. We assigned to the parameters the following values, that are the most commonly used in the literature: $\alpha = 0.2$, $\beta = 10$, $\gamma = 0.1$ and $\tau = 17$. Chaotic attractors appear when the parameter τ (so-called delay) is larger than 16.8 [**?**]. The data preprocessing and experimental setup was designed following the setting realized in [**?** **?**]. An initial time window with 1000 samples was used for cleaning up the initial transient phase. The training phase was made with 3000 points. The testing phase has 2084 samples, in which we used the first 2000 time steps in a teacher-forced mode. From this last time step, we run the network in exploitation mode.

The well-known Lorenz system is defined by the differential equations

$$\frac{\partial x}{\partial t} = \sigma(y-x), \quad \frac{\partial y}{\partial t} = rx - y - xz, \quad \frac{\partial z}{\partial t} = xy - bz.$$

This system was initially developed as an attempt to model the atmosphere convection. Lorenz's equations have received a lot of attention because the system is presented also in other physical phenomena such as the dynamics of electro-rotation, thermohaline ocean flow circulation, Malkus waterwheel and dynamics of whether transition [**?**]. The data was generated using the original parameters in the equations: $r = 28$, $b = 8/3$, $\sigma = 10$. The generated data in the x-coordinate trajectory was rescaled by a factor of 0.01, following [**?**]. Then, the obtained series is used for the learning process. For the learning procedures, we replicated the experimental setting presented in [**?**]. The first 1000 samples were used for removing the transient effects, that is, for eliminating the impact of the initial conditions. The training sequence has a length of 6000 samples, and the testing sequence has 1600 points. The first 1000 samples were used in teacher-forced mode, and the last 600 points in exploitation mode.

Real world data: Monthly sunspot series. It has been observed that the Sun has occasionally sunspots, which became a distinctive feature of the solar cycle. This series is used for obtaining a better knowledge of the solar cycles, and insights regarding the evolution of the total number of spotless days [**?**]. We used a sequence with 3276 data points from the period Jan. 1749 to Dec. 2021 [**?**]. The experimental setting was replicated from the experiments analyzed in [**?** **?**]. We standardized the variance and normalized the data in $[-0.5, 0.5]$, in order to obtain a better control of the recurrent stability and to avoid neurons' saturation. We used an initial window with 100 samples for washout the initial transient. For the learning procedure, we used 1600 instances for training and 500 instances for testing evaluation.

5.2. Results

Evaluation metrics. For the model evaluation, we replicated the experimental protocol conducted in [**?** **?**]. The general procedure is as follows. We ran the trained model over the testing dataset, and we discarded a washout with initial transients. Then, we computed the forecasting error using H time steps ahead. The estimation of the predictive error was made performing several metrics. We used normalized-root-mean-squared error (NRMSE) for the MGS and Lorenz problems, that was also used in [**?** **?** **?**]. In the case of the sunspot series problem, we used the normalized-mean-squared error (NMSE), since it was employed in [**?**]. In addition, for the sensitive analysis of the parameters, we used the root-mean-square error (RMSE) computed over a time windows with H samples. For a better visualization, we used a 10-logarithmic scale in the error curves.

In the RC area, it is common to see the evaluation of the learning procedure using two criteria, often denoted as NMRSE(H) and NMRSE_H. NMRSE(H) denotes the NMRSE computed in the testing phase using the residuals over *the whole horizon* H. The NRMSE_H is computed in the testing phase after H autonomous steps, and the error is computed only using the residual at the H time step [**?** **?**]. The NRMSE_{84} metric has become the standard error function for evaluating the MGS prediction [**?**].

Hyperparameter selection. We conducted a grid search strategy to define the main global parameters. We also follow suggestions from the literature at the moment of defining parameters, such as network size, input scaling, added noise, non-linearity type of projections, and feedback connections [**?**]. A grid search in the range $[0.1 : 0.1 : 1.4]$ was performed for the spectral

radius of the reservoir matrix. Even though the ESP is not guaranteed, we evaluated matrices with spectral radius larger than 1, similarly than [**?** **?** **?**]. Actually, in our experiments, we figured out that some reservoirs with large spectral values also reach good accuracy for a specific window forecast. For solving the Sunspot task, we adopted both grid search and the experimental setting employed in [**?**]. Following [**?**], the objective in the Sunspot problem is to predict one single time-step ahead [**?**]. In other words, the network doesn't perform in exploitation mode over large testing time windows. In the case of the Sunspot task, we studied only networks without feedback connections. We did a grid search in $[0.1 : 0.1 : 0.5]$ for the sparsity parameter. The leaky rate was set in the range $[0.01 : 0.05 : 0.5]$. The regularization parameter used for computing the readout weights was evaluated with values 10^{-i}, with $i \in \{3, 4, 5, 6, 7, 8\}$. The input scaling factor was evaluated in the set $\{0.01, 0.02, 0.03, 0.05, 0.07, 0.09, 0.1, 0.15, 0.2, 0.3, 0.5\}$, and it was selected the 0.1 value. The pattern of connectivity of the reservoir was randomly initialized using uniform distributions. In addition, we filled the upper diagonal of the reservoir matrix with random values, and also added a random value in the last row and first column. This step is commonly made to guarantee at least one cycle in the reservoir. The range of the uniform distribution was specific for each problem, following the suggestions discussed in [**?**]. The readout was computed using a ridge regression with a regularization factor of 10^{-9} (MGS prediction task), 10^{-6} (Lorenz prediction task) [**?**], and 10^{-5} (Sunspot prediction task).

The PSO parameters have been tuned following the suggestions presented in [**?**]. Figure **??** shows an empirical analysis of the impact of the inertia

Table 1: Sensitive analysis of the inertia parameter of the PSO algorithm with 75 particles. The 95% confidence interval was computed using the results of 10 explorations with 100 generations each of them. The error is the RMSE over a window of 84 time steps.

Inertia	Mean	CI lower bound	CI upper bound
0.5	0.0133	0.0116	0.0151
0.6	0.0125	0.0109	0.014
0.7	0.0136	0.0112	0.0161
0.8	0.0134	0.012	0.0147
0.9	**0.0109**	0.0072	0.0145
0.95	0.0153	0.0146	0.0159

values on the evolution. A grid search in the range $[0.5 : 0.1 : 1]$ with the additional convergence condition $1 > \iota > 0.5(c_1+c_2)-1 \geq 0$ was performed [**?**]. The c_1 and c_2 values were initialized in the interval $[0.5, 2.5]$ [**?**]. Table **??** presents a statistical analysis for different values of the inertia parameter. We performed a total of 10 experiments with different inertia values over the MGS data, then we computed 95% t-student confidence intervals that are shown in Table **??**. Consequently, we selected an inertia value equal to 0.9 that we used in our further experiments. Another hyperparameter is the initialization procedure for the coefficients, which defines the initial particles. We partially investigated its impact. Some results are presented in Figure **??**. We decided to create swarms with particles initialized in a different range of values. The swarm has particles randomly initialized with uniform distribution in $[-a, a]$ with $a \in \{4, 5, 6\}$.

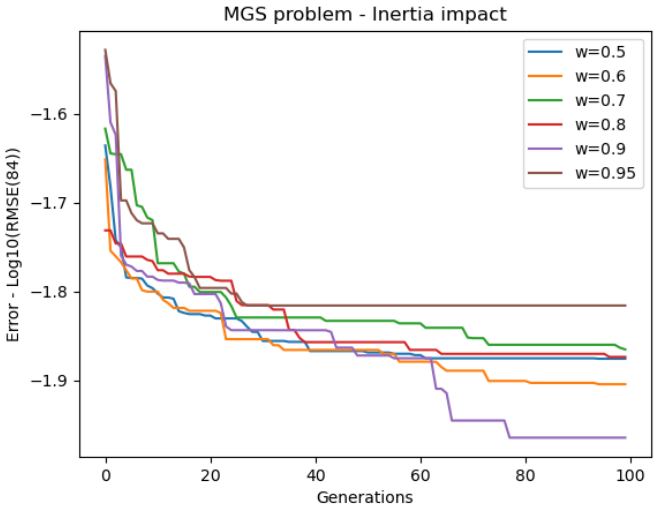

Figure 4: Error evolution according to the inertia parameter. Curves present the results of swarm experiments. It is shown the error average obtained by 5 different initial populations. The particle size is 75 and the population size was 10. Metric error was the RMSE over a testing window with 84 points.

Results analysis. Figure **??** has two graphics showing the error in logarithmic scale, for MGS and Lorenz tasks. The curves for different values of C show the evolution of the average of RMSE across the 10 independent experiments. In these experiments, the particles were initialized with values in $\mathcal{U}[-a, a]$ with $a = \{4, 5, 6\}$, and the spectral radius of the reservoir matrix wasn't scaled. In other words, the swarm has an initial diversity with respect to the reservoir matrix spectrum's. The graphs in Figures **??** also have a constant curve labeled ESN+ with 95% confidence interval margins (computed over an average of 30 experiments). It shows the error obtained with the "optimal" ESN architecture (according to the results in [**?** **?**] using our own implementation). Remarkably, a few swarm iterations are

enough to overcome the results achieved with baseline modern ESN. Note that ESN+ is a complex model with feedback connections, injected noise, regularization parameter, etc. We explicitly showed the ESN+ performance as a kind of threshold obtained by the state of the art. Observe that the accuracy becomes roughly independent of C. These results aren't consistent with previous ones where the evolution was made using GAs [**?**]. In the case of applying GA, larger chromosomes obtained a better accuracy. In the case of applying PSO, this seems to behave differently. Maybe, it is related to the evolutionary search characteristics of the PSO algorithm, where the small-size particles explore more efficiently the searching space. A further analysis regarding the relationship between the C value, population size and exploration-exploitation characteristics of the evolutionary search can eventually also be studied in the future. Figure **??** presents results obtained with the MGS data. It illustrates the impact of the initialization of the coefficients when the reservoir matrix is scaled for controlling the initial spectral radius. In addition, the graphic in the right shows the evolution of the spectral radius through the generations. Occasionally, the best swarm solution represents reservoirs with spectral radius larger than 1. Figure **??** has four graphics; it visualizes the advantages of starting the swarm with particles that represent reservoirs with controlled spectral radius. The graphics were done with the Lorenz data, but the same behavior has been presented in other analyzed tasks as well. The spectral radius of the reservoir is a central parameter of the model. To initialize particles representing reservoirs with spectral radius in an appropriate range is better than a total random initialization, the error decreases much faster during the evolution. Also, it is visible that some best

particles in the swarm represent reservoirs with spectral radius larger than 1. Therefore, the swarm is also able to explore other regions. In other words, reservoirs with spectral radius larger than 1, are also evaluated during the evolutionary search.

Comparisons with related works. Table **??** summarizes the results obtained for the MGS problem. In [**?**] was introduced a learning approach for correcting the bias introduced by the teacher-forcing schema. In addition to the readout training, the authors apply a training for bias correction. As far as we know, Jaeger *et al.* obtained the best performance for the MSG task using this specific algorithm, that they called *refined learning method*. We evaluated the model using linear and hyperbolic tangent readout. Table **??** also shows the results obtained by EvoESN with GA, it presents the average of errors after 50 trials obtained using GA with chromosomes with a size of 500, a population of 20, and 70 and 300 generations. More information about the GA implementation in the EvoESN model is available in [**?**]. Note that EvoESN slightly overcomes the SOTA accuracy reported in [**?**]. However, even though the variance is very low, we cannot say that the difference is significant with the error reported in [**?**] using the additional learning schema (ESN with FB (II), see Table **??**). However, it is relevant to note that a few PSO iterations are enough to significantly decrease the error, as it was shown in Figure **??**.

Table **??** shows results obtained for the Lorenz prediction task. The EvoESN with PSO has feedback connections and a non-linear readout. The initial spectral radius is controlled. The errors are averages computed over 5 independent experiments. The NRMSE_{84} error using GA has slighter lower

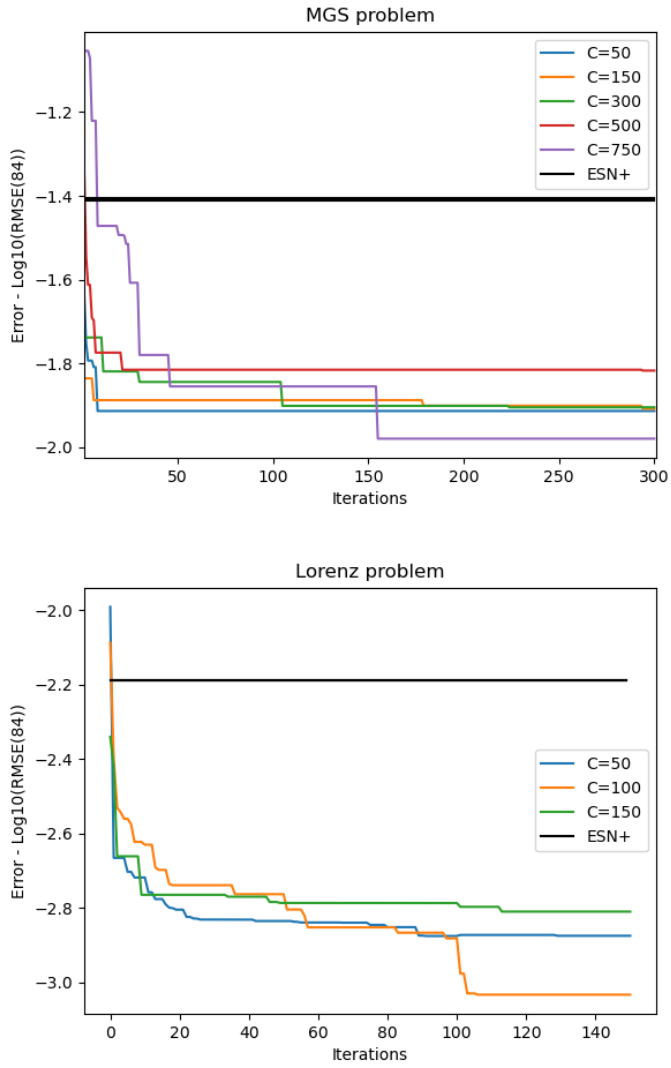

Figure 5: Impact of the number of coefficients on the accuracy. The results presented with EvoESN were computed without spectral radius control over the reservoirs matrices. The constant curve (ESN+) shows the reached accuracy by the ESN with noise and feedback connections designed for MGS and Lorenz problems in [**?**].

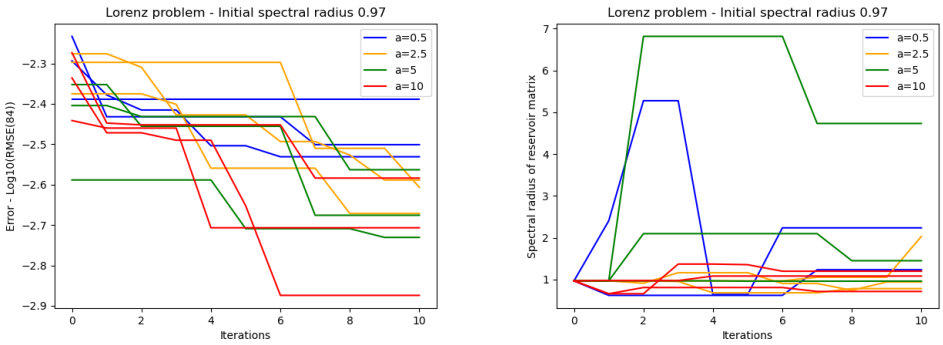

Figure 6: Impact of the initialization of the coefficients. The initial spectral radius of the reservoir was 0.97. The coefficients were randomly initialized using the uniform distribution in $[-a, a]$. The graphic on the left shows the error evolution in the first 10 iterations. On the right side, we show the evolution of the spectral radius of the best global particle.

error than the result obtained by the refined learning schema [**?**]. PSO with a smaller number of coefficients (particle size) and in a half number of generations reach equal order of magnitude than GA and the SOTA result [**?**]. Table **??** presents the test errors for the Sunspot series problem. EvoESN (with GA and PSO) outperforms the results reported in [**?**]. Furthermore, it also outperforms the results obtained by ESN with leaky integrator and ESN with feedback connections. We defined a randomly selected reservoir topology, and we ran 50 independent random weights initialization. Independently of the selected leaky rate, the obtained error variance has been small ($\approx\ 1 \times 10^{-6}$). For this problem (with the evaluated number of iterations), it is clear how PSO achieves a better accuracy than GA. Figure **??** presents the evolution of the error in logarithmic scale, the EvoESN with PSO is able to decrease the error very fast, only few generations are needed.

Algorithmic complexity. EvoESN has a higher computational cost than

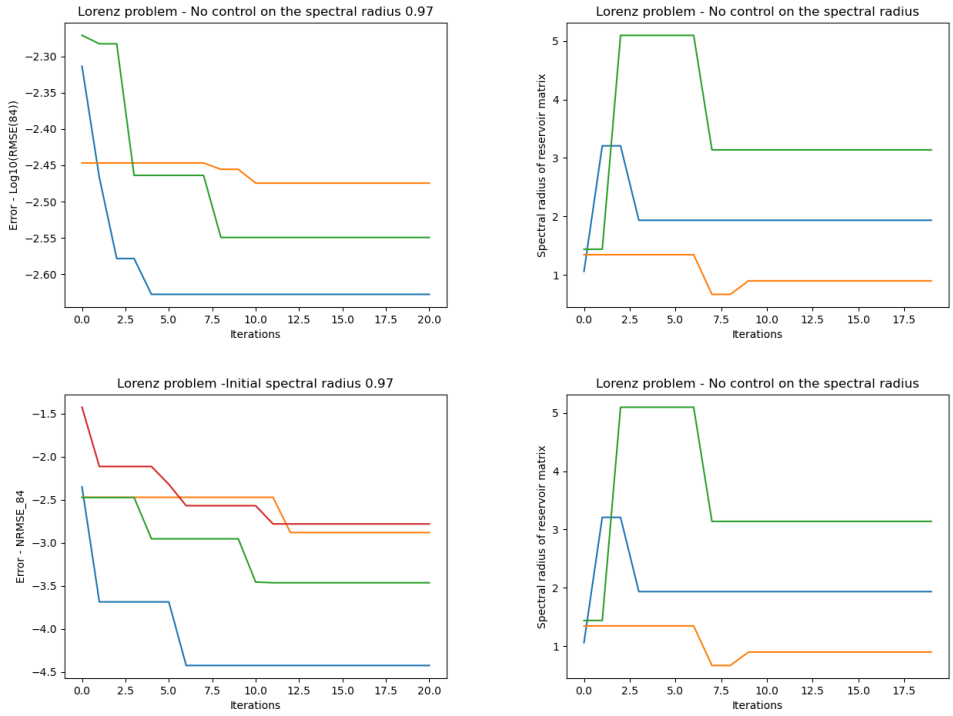

Figure 7: Impact of controlling the spectral radius of the reservoir. The coefficients were randomly initialized using a uniform distribution in $[-5, 5]$. The two figures above show results with reservoir without control of the spectral radius. The two figures below show results, with initial reservoirs with controlled spectral radius (fixed to 0.97). The graphics on the left shows the error evolution in the first 20 iterations. On the right side, we show the evolution of the spectral radius of the best global particle. The particle size is 50 and the population size is 10.

the standard ESNs. Note that at each generation, the EA performs the following operations: training of the readouts using a linear regression, computing the inverse DCT, and performing the evolutionary operations. Training the readouts needs matrix operations (matrix products, matrix times vectors, matrix inverses). Those operations also appear in the standard ESN. However, they are performed over each particle and per each generation in the EvoESN model. Other additional operation comes from the Fourier transform. A Fast Fourier transform (FFT) algorithm has a computational complexity of $O(M \log(M))$, where M is the number of Fourier coefficients. Evolutionary operations performed from one generation to the next one are less costly (matrix-vector operations). Resuming, the computational complexity of EvoESN lies on the FFT method and the selected EA. Nevertheless, we observed in the different experiments that a few generations of the PSO algorithm are enough for achieving much lower errors with respect to ESNs. According to our results, GA has a lower convergence speed than PSO, and also requires encoding the weights in larger vectors.

Computational aspects. The IDCT was chosen with the orthogonal norm using the `fftpack` Python package. PSO algorithm was designed according to the description presented in Section **??**. The results of EvoESN using GA were conducted using the `DEAP` package [**?**]. Statistical tests have been done using the `Statsmodels` package [**?**]. The `Numpy` package was used for matrix operations such as spectral radius computation and matrix inverse [**?**].

Table 2: Summary of MGS results. Comparison between the proposed EvoESN optimized using GA and PSO with other ESN variations and results reported in the literature. ESN with FB (I) means ESN with feedbacks, and ESN with FB (II) means ESN with feedback and the refined learning method.

RC model	Characteristics	Error	Metric
H. Jaeger *et al* [**?**]	ESN-FB (II)	$\approx 10^{-4.2}$	NRMSE_{84}
J. Schmidhüber et al. [**?**]	Evolino	$\approx 1.9 \times 10^{-3}$	NRMSE_{84}
ESN	without FB, non-linear readout	0.1770	NRMSE ($84H$)
ESN	without FB, linear readout	0.29857	NRMSE ($84H$)
ESN	with FB (I), linear readout	0.19620	NRMSE ($84H$)
ESN	with FB (I), non-linear readout	0.19376	NRMSE ($84H$)
ESN	with FB (II), non-linear readout	0.07684	NRMSE ($84H$)
EvoESN (GA)	$C = 500$, 20 chromosomes, 70 generations	$\approx 4.485 \times 10^{-4}$	NRMSE ($84H$)
EvoESN (GA)	$C = 500$, 20 chromosomes, 300 generations	$\approx 1.075 \times 10^{-4}$	NRMSE_{84}
EvoESN (PSO)	$C = 50$, 30 particles, 150 generations	$\approx 4.172 \times 10^{-4}$	NRMS_{84}
EvoESN (PSO)	$C = 100$, 10 particles, 150 generations	$\approx 7.068 \times 10^{-4}$	NRMS_{84}

6. Discussion

From previous theoretical analysis and the conducted experiments, we identify the following characteristics, advantages, limitations and future research directions.

Initialization scheme. We studied different initialization forms for the particles. According to the domains of the DCT mapping, we conclude that an appropriate choice is to use uniform distributions on $[-a, a]$ with, say, $a \in [4, 6]$, in order to avoid neuron saturation. Although, when we scale the initial reservoirs for satisfying the ESP, the type of random initialization for each particle does not significantly affect the results.

Table 3: Summary of Lorenz results. Comparison between the proposed EvoESN optimized using GA and PSO with other ESN variations and results reported in the literature.

RC model	Characteristics	Error	Metric
H. Jaeger *et al* [**?**]	ESN-FB (I)	≈ -2.33	$log_{10}\text{NRMSE}_{84}$
H. Jaeger *et al* [**?**]	ESN-FB (II)	≈ -4.27	$log_{10}\text{NRMSE}_{84}$
EvoESN (GA)	$C = 50$, 300 generations	-3.4666	log_{10} NRMSE84
EvoESN (GA)	$C = 100$, 300 generations	-3.6014	log_{10} NRMSE84
EvoESN (GA)	$C = 150$, 300 generations	-4.1379	log_{10} NRMSE84
EvoESN (PSO)	$C = 50$, 30 particles, 150 generations	≈ -4.2852	log_{10} NRMSE_{84}

Dimensionality reduction. How to encode recurrent networks is still an open and rich research area. Here, we proposed an optimization procedure performed in a frequency space whose dimension is much smaller than the number of non-null reservoir weights. The "energy compaction" property of DCTs makes that few coefficients have enough information about the corresponding reservoir weights.

Particle/chromosome size. GA seems to be much more sensitive to the C value than PSO. Populations with larger chromosomes achieve lower errors, although the searching space exploration is much more costly. According to the results, we didn't observe a similar behavior for the PSO algorithm. It seems that PSO is less sensitive to the particle size than GA. PSO can achieve similar results than GA with a smaller number of parameters. Even though the searching space of PSO has a lower dimensionality than the searching space of GA, PSO can achieve the same order of accuracy. Those results also suggest, for the near future, to realize a further investigation about the

Table 4: Summary of Sunspot results. The EvoESN has feedback connection and non-linear readout. The errors are averages computed over 5 independent experiments.

RC model	Characteristics	Error	Metric
Rodan *et al.* [**?**]	ESN	≈ 0.1042	NMSE
Rodan *et al.* [**?**]	DLR, DLRB, SCR	≈ 0.1039	NMSE
ESN	without FB	0.085358	NMSE
LI-ESN	leaky rate 0.1	0.033120	NMSE
LI-ESN	leaky rate 0.7	0.031816	NMSE
EvoESN (GA)	$C = 100$, 150 generations	0.0292394	NMSE
EvoESN (GA)	$C = 200$, 150 generations	0.0293263	NMSE
EvoESN (GA)	$C = 300$, 150 generations	0.0309722	NMSE
EvoESN (PSO)	$C = 25$, 20 particles, 100 generations	0.001714	NMSE
EvoESN (PSO)	$C = 50$, 20 particles, 100 generations	0.001516	NMSE
EvoESN (PSO)	$C = 75$, 20 particles, 100 generations	0.000461	NMSE

relationship among the dimensionality of the error landscape (given by C), the sampling strategies (affected by the population size, that is, the number of points used for covering the landscape) and the exploration strategy (which depends on the selected EA).

Control of the spectral radius of reservoir matrix. The results using PSO show the convenience of starting the evolutionary search with scaled reservoirs (matrices with appropriate spectral radius). A natural strategy can be to proceed to the selection of the spectral radius of the reservoir matrix following recommendations from the literature [**? ?**], then to apply the PSO algorithm with particles representing reservoirs with the selected spectral radius. In our experiments for PSO, we did not re-scale the reservoir at each

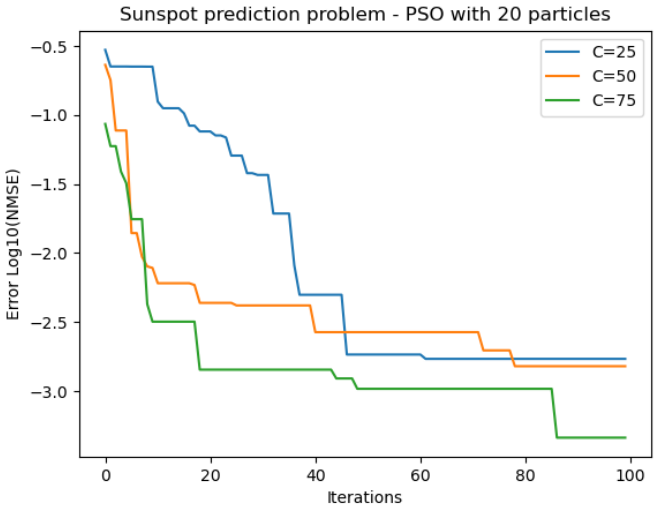

Figure 8: Error evolution for the Sunsport problem when it was used PSO with 100 particles. The error is presented in logarithmic scale for a better visualization.

iteration. It was only computed for each particle in the initial population. Computing the spectral radius at each iteration will considerably increase the computational cost.

Initial diversity. We increased the initial diversity of the swarm by considering several values of $a \in [4, 6]$. It can also be interesting to explore, in the future, the impact of incorporating other type of diversity on the swarm. By diversity, we refer to the spectral radius of the reservoir that is a crucial parameter controlling the stability.

Decoupling between readout training and evolutionary fitness function. Note that the fitness function used in the evolutionary search can be different from the one used for training the readout weights. This opens several interesting possibilities, since the evolutionary optimization can be done

over other properties of the reservoir, for instance, over stability control.

Searching strategy. Observe that the search for good reservoirs can include modifying the value of C. It is possible to start with a low value (low dimensional chromosomes) and then successively increase it for adding higher frequencies (the computational power of the reservoir increases with the number of coefficients).

Weight normalization. Due to the fact that the approach depends on the IDCT mapping, it would be good to investigate in future works if a control of the weights can be beneficial. Probably, weight normalization after each generation can help to avoid neuron saturation, with additional computational cost to the method.

7. Conclusions and future work

In this paper, we added to the standard RC approach an evolutionary procedure for finding an optimal (in some sense) recurrent structure (reservoir matrix). Applying evolutionary methods to optimize recurrent structures seems to be a good strategy to avoid the well-known limitations presented in gradient-based methods. To reduce the searching space, we applied a compression of the reservoir weights, by moving them into the frequency domain using the DCT transform. Then, the reservoir weights were found by means of an evolutionary search.

We have analyzed the model using two very popular EAs: GAs and PSO. An empirical analysis over different datasets showed the potential of the idea. In general, we needed just a few evolutionary generations for achieving a higher accuracy than the standard ESN and its most popular modern

variations. With respect to the initial version [**?**], we integrated many results. We improved the global procedure using the PSO approach. PSO is more stable than GA (with respect to the number of Fourier coefficients to use), and achieves similar accuracy than GA with a smaller number of parameters and of iterations. We also explored the impact of the initialization of the search in the Fourier space, and proposed specific ways of starting this exploration. In addition, we analyzed the evolution of the reservoirs and the stability issues.

We believe that our technique opens new possibilities for improving these tools, because we obtained better results from moving to the Fourier space leading to a dimensionality reduction, or in the EA side, by choosing a better optimization method. The exploration of the proposed approach in hierarchical architectures is an obvious research direction to be investigated. To compare our results with other indirect encoding approaches deserves also attention in the near future.