The optimal combination: Grammatical swarm, particle swarm optimization and neural networks

Luis Fernando de Mingo López*, Nuria Gómez Blas, Alberto Arteta

A B S T R A C T

Social behaviour is mainly based on swarm colonies, in which each individual shares its knowledge about the environment with other individuals to get optimal solutions. Such co-operative model differs from competitive models in the way that individuals die and are born by combining information of alive ones. This paper presents the particle swarm optimization with differential evolution algorithm in order to train a neural network instead the classic back propagation algorithm. The performance of a neural network for particular problems is critically dependant on the choice of the processing elements, the net architecture and the learning algorithm. This work is focused in the development of methods for the evolutionary design of artificial neural networks. This paper focuses in optimizing the topology and structure of connectivity for these networks.

1. Introduction

Natural sciences, and especially biology, represent a rich source of modelling paradigms. Well-defined areas of artificial intelligence (genetic algorithms, neural networks), mathematics, and theoretical computer science (*L* systems, *DNA* computing) are massively influenced by the behaviour of various biological entities and phenomena [1]. In the last decades, new emerging fields of so-called natural computing identify new (unconventional) computational paradigms in different forms [2]. There are attempts to define new mathematical and theoretical models inspired by nature. Moreover, computational paradigms suggested by biochemical phenomena [3] are object of study.

Particle swarm optimization (*PSO*) [4,5] is a global optimization algorithm for dealing with complex problems. Optimal solution to these problems is represented as a surface in an n-dimensional space. Hypotheses are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles [2]. Particles then move through the solution space, and are evaluated by some criteria after each step. Particles accelerate upon time towards other particles which have better fitness values [3]; this process occurs within the communication group. This idea shows an improvement when compared with other global minimization strategies such as simulated annealing; for example, the large number of members that in the particle swarm make the technique resilient to the problem of local minima [4,6,7].

Grammatical swarm (*GS*) adopts a particle swarm learning algorithm which is linked to a grammatical evolution (*GE*) [8] genotype–phenotype mapping to generate programs in an arbitrary language. Grammatical evolution (*GE*) [9] is an evolutionary algorithm that can evolve computer programs in any language, and can be considered as a form of grammar-based [10] genetic programming. Rather than representing programs as parse trees [11], a linear genome representation is used. A genotype–phenotype mapping is deployed for each individual binary string variable; it keeps the information to select production rules from a Backus Naur Form (*BNF*) grammar. This information is stored in codons (groups of 8 bits). The grammar allows to build programs in an arbitrary language and guarantees syntax correction. This method is used as a generative grammar, as opposed to the classical use of grammars in compilers to check syntactic correctness of sentences.

Neural networks are usually seen as a method to implement complex non-linear mappings using simple elementary units interrelated through connections with adaptive weights.

2. Neural networks

Neural networks are non-linear systems whose structure is based on principles observed in biological neuronal systems. A neural network may be considered as a system capable of answering queries or providing inputs to given outputs. The in/out combination, i.e. the transfer function of the network is not programmed but obtained through a "training" process on empiric datasets.

The network builds the function that relates "input" to "output" by processing correct input/output pairs. For each input the network returns an output which is not exactly the desired output, so the training algorithm modifies some parameters of the network in the desired direction. Hence, every time an example is input, the algorithm adjusts its network parameters to the optimal values for the given solution: in this way the algorithm tries to reach the optimal solution for all the examples. These parameters are essentially the weights or linking factors between each neuron that forms our network.

Neural networks application fields are typically those where classic algorithms fail because of their inflexibility (they need precise input datasets) [12]. Usually problems with imprecise input datasets are those whose number of possible inputs datasets is so big that they cannot be classified. For example, in the image recognition field probabilistic algorithms have a lower efficiency than neural networks (even lower than the low flexible neural networks). Classic algorithms have also troubles in analysis of phenomena that do not respond to mathematical rules.

There are indeed rather complex algorithms which can analyse these phenomena; however, neural networks turn out to be the most efficient [13,14] by far. These algorithms use Fourier's transformation to divide phenomena in frequential components; that is the reason why the results are highly complex (they only extract a few number of harmonics by generating a big number of approximations). A neural network trained with complex phenomena's data is able to estimate also frequential components, However there is a big problem to solve when implementing one of these neural networks: The election of a valid architecture. This must be carefully chosen in order to obtain better results.

This paper proposes a grammatical swarm algorithm that decides the right architecture/topology to use in every case. Furthermore, the training process uses particle swarm optimization. Then a neural network is obtained just by running these two processes. (Obtaining the right topology and training the neural networks by using the appropriate weights.) Weights are defined by using ideas from social intelligence. Cross-disciplinary tools are useful in computational sciences [15].

Next sections describe how to implement a model which obtains neural network topology and trains it.

3. Particle swarm optimization

Particle swarm optimization (*PSO*) is a fairly recent population based stochastic optimization technique introduced by Kennedy and Eberhart [16]. It belongs to evolutionary computation area which uses iterative progress such as development in a set of numbers or codes. This set is addressed as *population*. The population is being moved over a searching space by an algorithm that looks for a solution. A particle is an element of the population and represents a candidate solution to the problem. The model is inspired, like any other evolutionary algorithm, by a social-psychological model [5]. The case of our study is an algorithm which simulates the social behaviour of a group of fish or flock of birds; this means that particles move in swarms and thus stay relatively close together. A swarm has no leader and no one coordinates its behaviour.

Let the following symbols represent properties of a particle:

- x_i is the current position of particle i,
- v_i is the current velocity of particle i,
- p_{Best} is the personal best position of the particle,
- g_{Best} is the global best particle.

With these notations, the formula to calculate a particle's velocity is:

$$v_i(t+1) = v_i(t) + c_1 * r_1 * (p_{Best} - x_i) + c_2 * r_2 * (g_{Best} - x_i)$$

The next formula, for the new position of the particle, adds the newly calculated velocity to its current position:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

r_1 and r_2 are randomly generated for every velocity update and $0 \leq r_1, r_2 \leq 1$. They should both be different at each iteration. And c_1, c_2 are user defined values called acceleration coefficients where $0 \leq c_1, c_2 \leq 2$. Their value depends on the problem to be optimized.

Term *velocity clamping* has not been previously included, but it is necessary to restrain velocity updates. Without doing so, particles would move too far from the search space, ignoring the current solution. If the search space has a range $[-x_{max}, x_{max}]$ then the velocity should also have a range $[-v_{max}, v_{max}]$. Our proposal for v_{max} is $k^* x_{max}$, k being the clamping factor between 0.1 and 1. The maximum velocity works as a constraint to control the global exploration of a swarm. If the value is too high particles might bypass good solution and the exploration would be poor. On the other hand, slow particles might only be searching locally which would be good for local solutions but not for finding a global one; this means that the exploiting process becomes poor too. Balancing between exploration and exploitation improves considerably *PSO*. Introducing an inertia weight, for example, would sort that out.

Adding an inertia weight in the velocity formula is a very small but effective update:

$$v_i(t+1) = w * v_i(t) + c_1 * r_1 * (p_{Best} - x_i) + c_2 * r_2 * (g_{Best} - x_i)$$

As shown, a relatively large w empowers the global search. On the other hand, small values for w help to search locally. The higher the value is, the faster the particle moves; thus, if values are small particles will move slowly. In most cases, when $w \geq 1$, velocity will increase upon time, reaching the maximum velocity if clamping is used. If $w < 1$, particles will slow down. This variable helps balancing between exploration and exploitation but does not eliminate the need for v_{max}. This new formula consists of 3 parts:

- The inertia component $w * v_i(t)$.
- The cognitive component $c_1 * r_1 * (p_{Best} - x_i)$: this part represents the particle's common sense, its memory. With the help of this component the particle is encouraged to move towards or around its personal best position.
- The social component $c_2 * r_2 * (g_{Best} - x_i)$: makes sure that the particle moves towards the best region found by the whole swarm.

Note that the behaviour also depends on the values of c_1 and c_2. For most optimization problems, the following values form a suitable combination: $w = 0.7$, $c_1 = c_2 = 1.4$.

By changing dynamically w, performance gets improved too. Depending on the problem, one can dynamically increase or decrease the inertia weight. Code and examples used in this paper are provided with a constant inertia weight. Fig. 1 shows the benchmark results when using the (*Black Box Optimization Benchmark*) with 24 noisy-free functions.

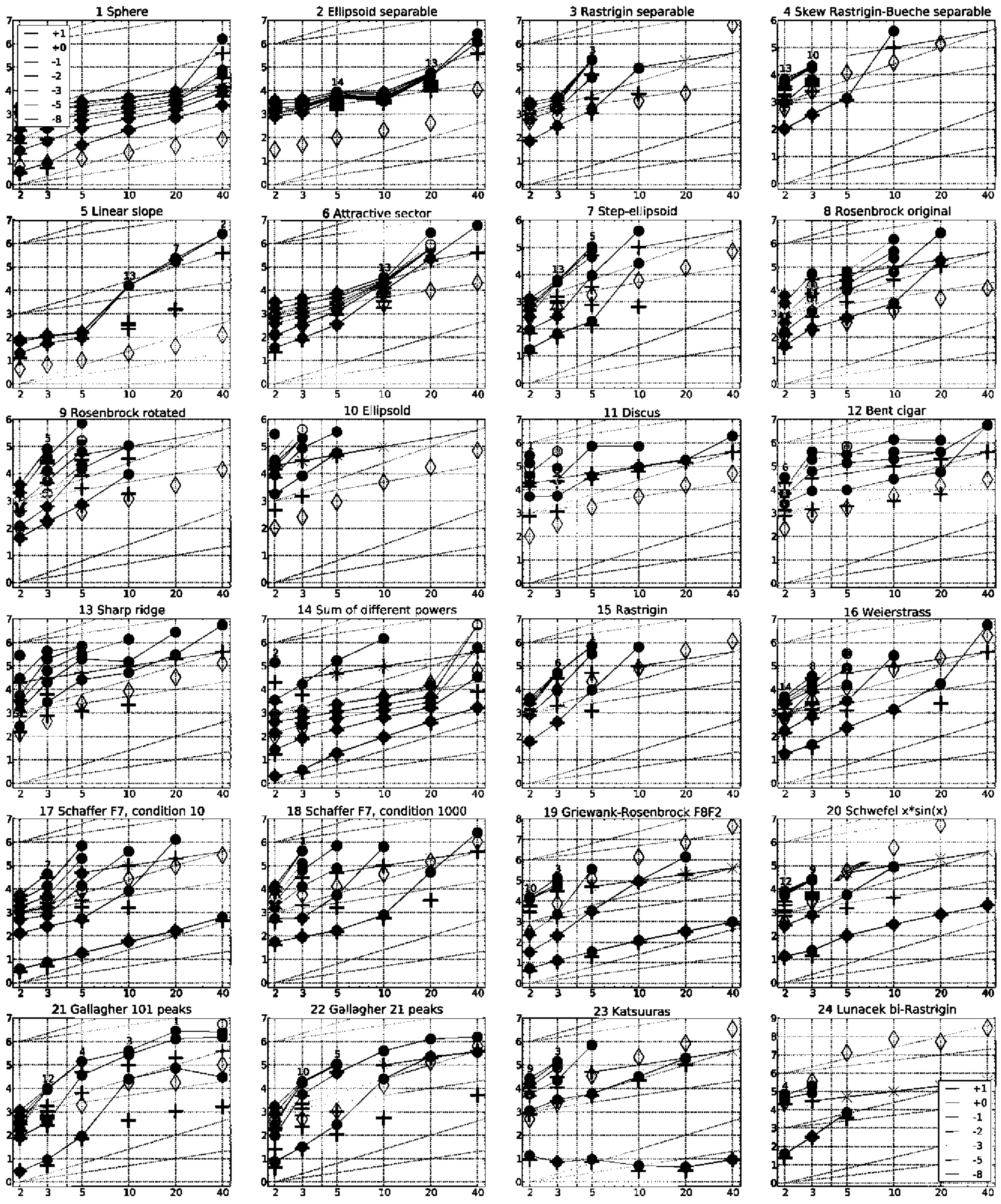


Fig. 1. PSO expected running time (ERT, •) to reach $f_{\text{opt}} + \Delta f$ and median number of f-evaluations from successful trials (+), for $\Delta f = 10^{\{+1,0,-1,-2,-3,-5,-8\}}$ (the exponent is given in the legend of f_1 and f_{24}) versus dimension in log–log presentation. For each function and dimension, $ERT(\Delta f)$ equals to #FEs(Δf) divided by the number of successful trials, where a trial is successful if $f_{\text{opt}} + \Delta f$ was surpassed. The #FEs(Δf) are the total number (sum) of f-evaluations while $f_{\text{opt}} + \Delta f$ was not surpassed in the trial, from all (successful and unsuccessful) trials, and f_{opt} is the optimal function value. Crosses (×) indicate the total number of f-evaluations, #FEs($-\infty$), divided by the number of trials. Numbers above ERT-symbols indicate the number of successful trials. Y-axis annotations are decimal logarithms. The thick light line with diamonds shows the single best results from BBOB-2009 for $\Delta f = 10^{-8}$. Additional grid lines show linear and quadratic scaling.

3.1. Differential evolution and PSO

Differential evolution is, like *PSO*, a stochastic and population-based optimization technique. It was first introduced in 1996 by Storn and Price [17]. Differential evolution is capable of handling non-differentiable, nonlinear and multimodal objective functions and is fairly fast in doing so. *DE* has participated in the *First International IEEE Competition on Evolutionary Optimization (ICEO)* and was proven to be one of the fastest evolutionary algorithms.

The *DE* algorithm also works with a population of potential solutions. The principle is same as *PSO*: a particle can gain by using information from other particles as well as the results of their own search. However, in the case of differential evolution, that information is sampled randomly. *DE-PSO* is basically a differential evolution algorithm mixed with ideas of particle swarm optimization [18].

PSO and *DE* define the population in the same way. It is necessary to implement population and velocities as two 2-*D* arrays of data. The higher the dimension, the more values each particle has. In order to initialize the velocities, one can either choose random values between the predefined bounds, or zero.

First, 3 random particles (r_1, r_2, r_3) should be chosen in such a way that they are different from each other. The second task is to create a mutated value for each dimension *j* of the particle according to the differential evolutionary algorithm.

When the mutation is done for every dimension, the mutated one is evaluated with the fitness function, and then compared it to the evaluation of the non-mutated one, i.e. the current particle. If it is smaller, then the mutated particle replaces the old one in the population; in other case the particle swarm optimizer is triggered.

If the *DE*-part of the algorithm does not find a better solution, *PSO* is activated. A new particle is created according to *PSO* formulas: velocity and new position. A basic velocity and position clamping is performed as well. We then just check whether the new values (velocity, position) exceeds the threshold defined in the algorithm.

If the newly created particle is proven to be better, it is replaced by the old one for the next generation. Both personal best and global best particle vectors are updated as well. This whole process occurs over again until the Halt condition is reached.

More hybrid versions between particle swarm optimization and differential evolution have been proposed, for example the one proposed by José García, Enrique Alba and Javier Apolloni in: *"Noiseless Functions Black-Box Optimization: Evaluation of a Hybrid Particle Swarm with Differential Operators"* [19]. Their model is also simple and is proven to obtain an accurate level of coverage range. The algorithm also contains 2 main parts. The first one is the differential variation, in which new velocities and positions are calculated according to the following formulas:

$$v_{ij}(t+1) = w * v_{ij}(t) + \mu + \phi * (g_{bestj} - x_{ij})$$
$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1)$$

j is the dimension and $i = 1, 2, \ldots$ population size, μ is a scaling factor ($\mu = UN(0, 1)$)[1] and ϕ is the social coefficient ($\phi = UN(0, 1)$). The second part is the mutation, which is calculated according to formula of *DE*. It is basically a new particle position between the specified bounds.

Figs. 1 and 2 show the benchmark results of both approaches (*PSO* and *DE-PSO*) using the *BBOB*2010 (*Black Box Optimization Benchmark*) with 24 noisy-free functions described in the benchmark procedure.

4. Neural network training using *DE-PSO*

Given a neural network architecture, every weight is coded as a genotype. Then we train the network by running the particle swarm optimization algorithm. The fitness function can be computed using the mean squared error of the net with the training data set. Some variations aim to get better fitness values with more generalization properties just by running validation and testing sets.

Equations used in the particle swarm optimization training process are below: c_1 and c_2 are two positive constants, R_1 and R_2 are two random numbers belonging to [0, 1] and w is the inertia weight. These equations define how the genotype values change along iterations; in other words this equations show how neural network weights change.

$$v_{in}(t+1) = wv_{in}(t) + c_1R_1(p_{in} - x_{in}(t)) + \qquad (1)$$
$$c_2R_2(p_{gn} - x_{in}(t)) \qquad (1)$$
$$x_{in}(t+1) = x_{in}(t) + v_{in}(t+1) \qquad (2)$$

Previous equations modify the network weights until the Halt condition is reached, that is to say, either a lower mean squared error or a maximum number of iterations is reached. Figs. 3 and 4 show two examples of a neural network training using the *DE-PSO* algorithm, the *XOR* and the binary coding problems (Table 1).

Best neural network weights with a 8-3-3 architecture and computed by the particle swarm algorithm are shown in Tables 2 and 3 . These two neural examples reveal that the *DE-PSO* previously defined can be successfully applied to the training stage in order to solve the convergence of the algorithm when working with high dimension individuals. The *XOR* example with dimension 9 and the binary-coding example with dimension 39 are a good candidates to start with and to combine classical neural networks with swarm intelligence.

5. Grammatical swarm

Grammatical swarm (*GS*) [20] relates particle swarm algorithm to a grammatical evolution (*GE*); genotype–phenotype mapping to generate programs in an arbitrary language [21]. The equations for the particle swarm algorithm are updated by adding new constraints to velocity and location dimension values, such us v_{max} (bounded to ±255), and dimension which is bounded to the range [0, 255] (denoted as c_{min} and c_{max}, respectively). Note that this is a continuous swarm algorithm with real-valued particle vectors. The standard *GE* mapping function is adopted, with the real-values in the particle vectors being rounded up or down to the nearest integer value for the mapping process. In the current implementation of *GS*, fixed-length vectors are used, which implies that it is possible for a variable number of dimensions to be used during the program construction genotype–phenotype mapping process. A vector's elements (values) may be used more than once if wrapping occurs, and it is also possible that not all dimensions are used during the mapping process. (This can happen whenever a program is generated before reaching the end of the vector.) In this latter case, the extra dimension values are simply ignored and are considered as introns that may be switched on in subsequent iterations.

5.1. Neural network topology using GS: first approach

Previous *PSO* model applied to a fixed neural network is a good training solution, however it does not define any kind or topology properties as it only obtains the best weight values. Following grammars can be used with grammatical swarm algorithms in order to obtain a network topology for a given problem.

This grammar can specify a feed-forward neural network topology with consecutive layers, that is to say, a classical multilayer perceptron.

```
<layers> ::= <layer> | <layer>, <layers>
<layer> ::= <digit>
<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

[1] UN(0,1) function provides information about the uniform distribution on the interval from *min* = 0 to *max* = 1 and generates random deviates. The uniform distribution has density *f*(*x*) = 1/(*max* − *min*) for *min* ≤ *x* ≤ *max*. UN(0,1) will not generate either of the extreme values unless *max* = *min* or *max* − *min* is small compared to *min*.

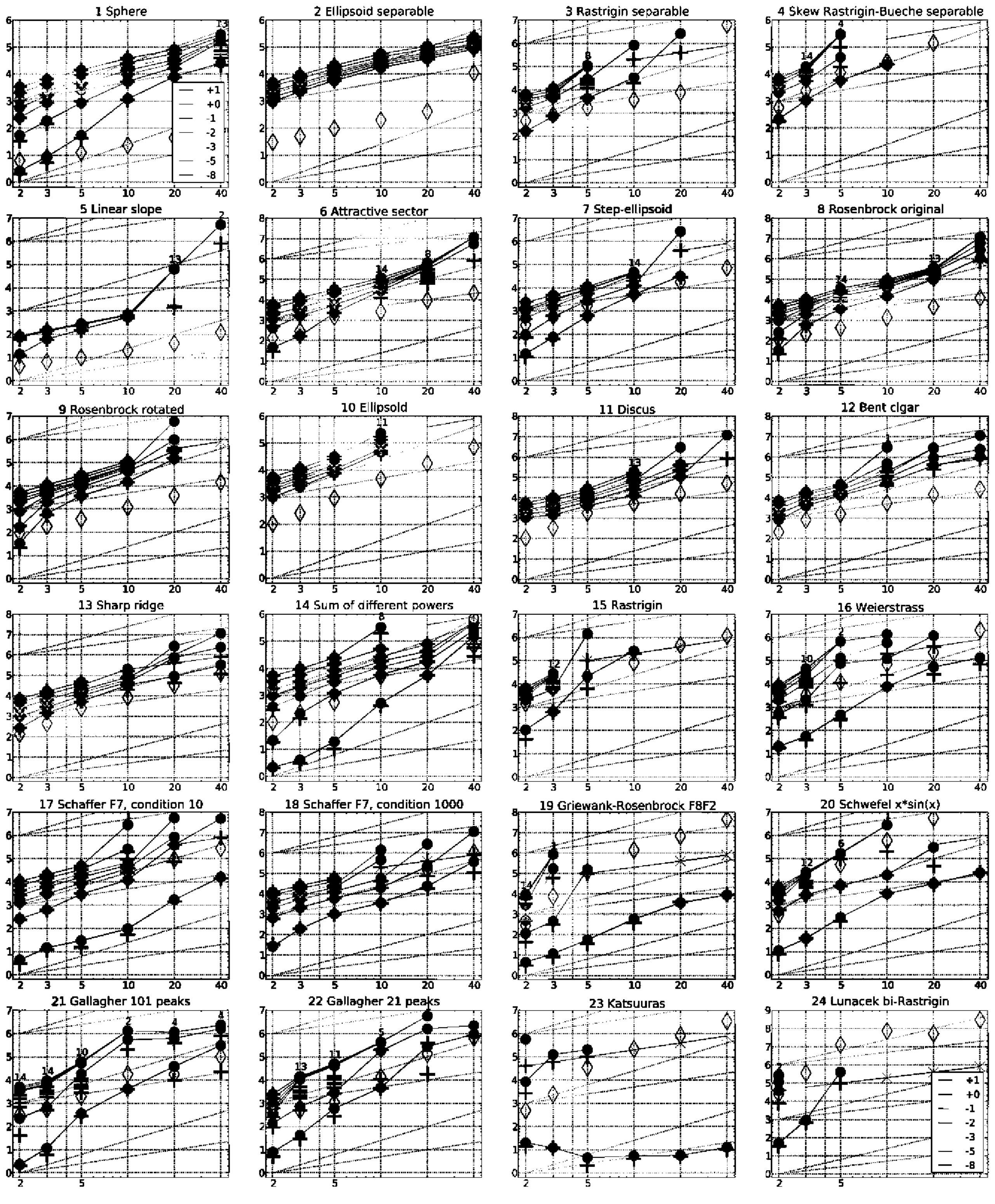
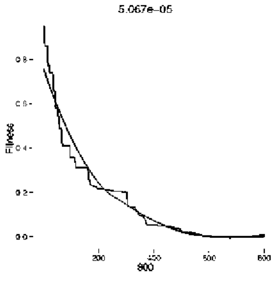


Fig. 2. DE-PSO expected running time (ERT, •) to reach $f_{\rm opt}+\Delta f$ and median number of f-evaluations from successful trials (+), for $\Delta f = 10^{\{+1,0,-1,-2,-3,-5,-8\}}$ (the exponent is given in the legend of f_1 and f_{24}) versus dimension in log–log presentation. For each function and dimension, $ERT(\Delta f)$ equals to #FEs(Δf) divided by the number of successful trials, where a trial is successful if $f_{\rm opt}+\Delta f$ was surpassed. The #FEs(Δf) are the total number (sum) of f-evaluations while $f_{\rm opt}+\Delta f$ was not surpassed in the trial, from all (successful and unsuccessful) trials, and $f_{\rm opt}$ is the optimal function value. Crosses (×) indicate the total number of f-evaluations, #FEs($-\infty$), divided by the number of trials. Numbers above ERT-symbols indicate the number of successful trials. Y-axis annotations are decimal logarithms. The thick light line with diamonds shows the single best results from BBOB-2009 for $\Delta f = 10^{-8}$. Additional grid lines show linear and quadratic scaling.

Table 1
Binary coding: neural network pattern set and best fitness value reached.



Input								Output		
1	1	1	1	1	1	1	−1	1	1	1
1	1	1	1	1	1	−1	1	1	1	−1
1	1	1	1	1	−1	1	1	1	−1	1
1	1	1	1	−1	1	1	1	1	−1	−1
1	1	1	−1	1	1	1	1	−1	1	1
1	1	−1	1	1	1	1	1	−1	1	−1
1	−1	1	1	1	1	1	1	−1	−1	1
−1	1	1	1	1	1	1	1	−1	−1	−1

Best fitness value: `5.067e−05`.

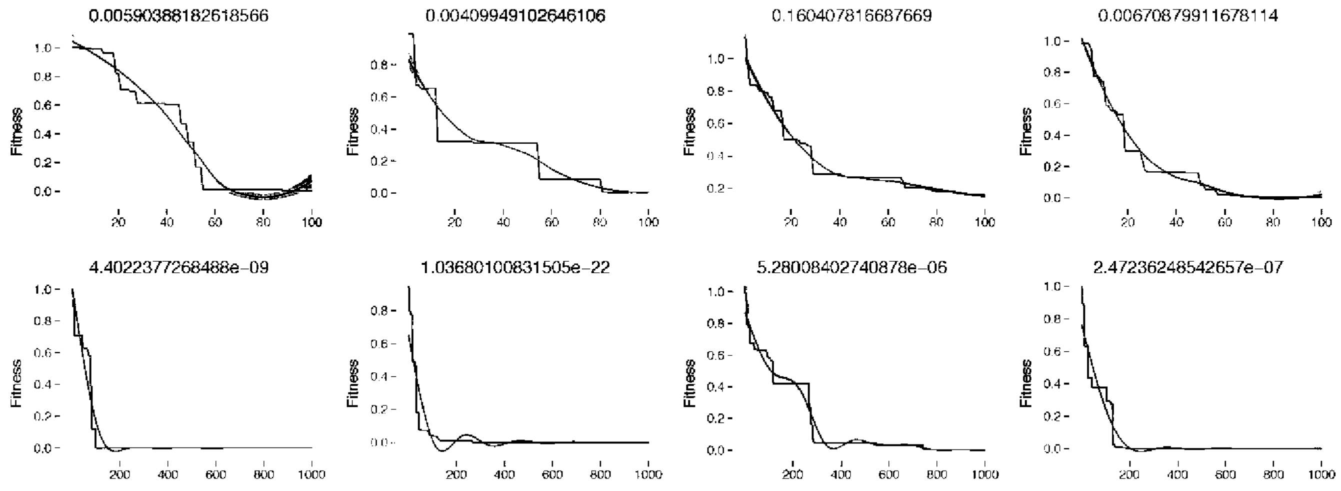


Fig. 3. *XOR* multilayer perceptron with 2 hidden neurons and a particle swarm optimization learning with 10 individuals (individuals have 9 dimensions). Each column represents a different random initialization and each row a number of iterations (100 and 1000).

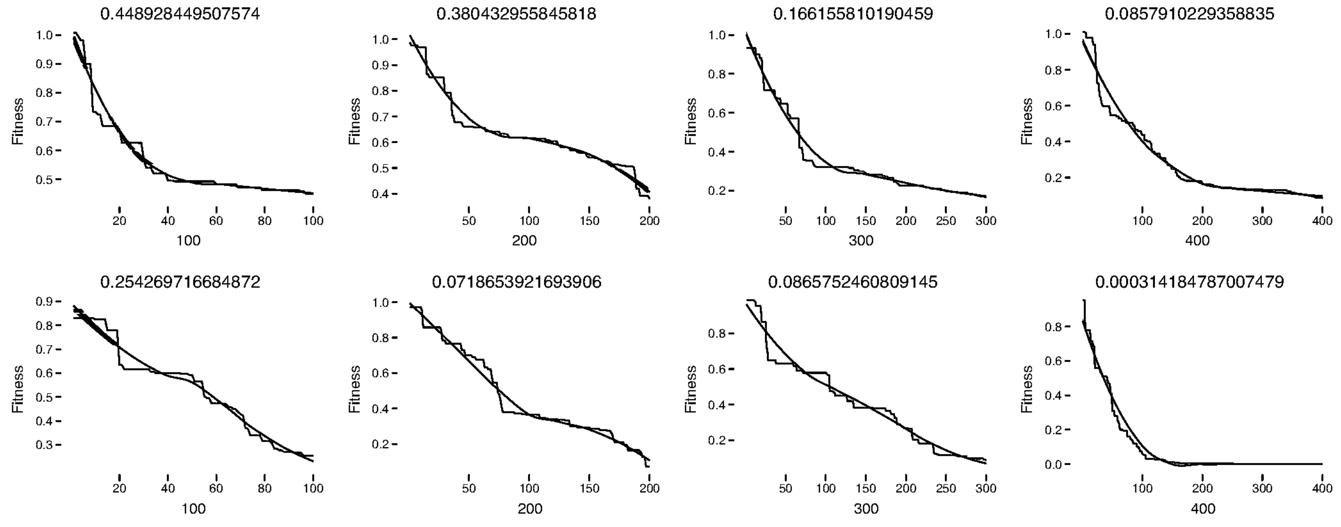


Fig. 4. Binary coding neural network form 8 inputs to 3 outputs. First row is a neural network with 3 hidden neurons and second row a neural network with 5 hidden neurons. Mean squared error (fitness of the particle swarm optimization) decreases as the number of iterations (100–400) increases.

Table 2
Binary coding: neural network weights, from the input layer to the hidden layer, taking into account the bias term.

weight $_{ij}$	hidden neuron 1	hidden neuron 2	hidden neuron 3
input neuron 1	0.1156528	1.097272	−0.946379977
input neuron 2	−0.3683220	25.945492	−1.703378035
input neuron 3	1.5325933	−9.765752	−0.636187430
input neuron 4	0.4830886	26.536611	0.002121948
input neuron 5	−1.5133460	0.790667	−0.131921926
input neuron 6	−1.7465932	−3.369892	0.987214704
input neuron 7	1.0519552	−4.920479	0.005404300
input neuron 8	0.3397246	−1.924014	3.273439670
Bias	0.7092471	−5.304714	−0.331283300

Table 3
Binary coding: neural network weights, from the hidden layer to the output layer with the bias term.

weight $_{ij}$	output neuron 1	output neuron 2	output neuron 3
hidden neuron 1	1.261584	−4.759320	0.9853185
hidden neuron 2	148.541038	−1.054461	−3.1161444
hidden neuron 3	−162.388447	−2.017318	−3.4691962
Bias	0.090192	1.207254	1.9260548

Next grammar is able to generate feed-forward connections not only with one consecutive layer but also with more than one consecutive layer. Such connections are defined by the `<connections>` non-terminal, where the `<digit>` means the n-consecutive layer.

```
<layers> ::= <layer> | <layer>, <layers>
<layer> ::= <digit> - <connections> -
<connections> := <digit> | <digit >, <connections>
<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

The whole algorithm is summarized as follows:

```
1 Create an initial population of genotypes.
2 For genotype i
  (a) Using genotype and grammar to obtain a neural architecture.
  (b) Compute fitness of genotype.
      • Apply previous PSO algorithm to train the genotype network.
  (c) Modified the best individual if appropriate.
3 Update velocity of genotype i.
4 Update position of genotype i.
5 If stop condition is not satisfied go to step 2.
```

This neural network model is a powerful one since only with the input and output pattern sets a network topology is chosen and also trained. Both tools, topology and training, are based on grammatical swarm and particle swarm optimization respectively, see Fig. 5.

Another useful approach could be to define a grammar with the weight values of neural networks connections instead of training the topology using a *PSO* algorithm.

6. Related work and experimental results

This work is focused in the development of methods for the evolutionary design of artificial neural networks. This paper focuses in optimizing the topology and structure of connectivity for these networks.

Artificial neural networks are well-established tools used with success in many problems such as pattern recognition, classification problems and optimization. The use of genetic algorithms in the evolution of neural networks has been widely used in the literature. However, in most cases limited experimental results are presented. In evolving a neural network using a genetic algorithm, two main approaches are used. In the first approach, the network topology and the network weights are evolved simultaneously [6,22–24], while in the second approach only the network topology is evolved and the parameters are estimated using a standard approach such as a gradient based optimization method [13,14]. Other approaches of defining the network's topology other than using genetic algorithms include pruning algorithms [13,14], simulated annealing based algorithms [13,14] and particle swarm optimization techniques [25–27,18].

The representation of the neural network is another important aspect of the evolutionary approach. Three major approaches are adopted in the bibliography. In the first, a binary coding of the topology and weights is used. In the second approach, a table structure is used to represent the connectivity between the neurons and the network's weights. The third approach is considered the most sophisticated since it involves using a higher-level language such as a grammar to describe the network topology. The latter approach gives better control to the network architect since he can use create node layers and better describe complex topologies.

Evolutionary neural networks have been widely used in solving various problems. Proposed method is tested against neural networks that are trained with various algorithms:

- RPROP [23] which is an improved version of the well-known back propagation method local optimization method.
- A Powell's variant of the well-known BFGS [28] local optimization method.
- A global optimization method namely MinFinder [29] which is capable under some conditions, to find all the local minimums of a function.
- Using and a genetic algorithm to estimate the neural network's initial parameters and the BFGS [30,31] local optimization method for fine tuning using a local search. This method is similar with the one described in [24]. In all cases, the function to be minimized in the case of neural networks is the train error. In almost all cases, the proposed method outperforms its competitors.

The datasets used for evaluating the proposed method, are known datasets that are available for download from the Internet and refer to both classification and regression problems. The number of datasets that are presented and evaluated are 9 classification problems and 9 regression problems.

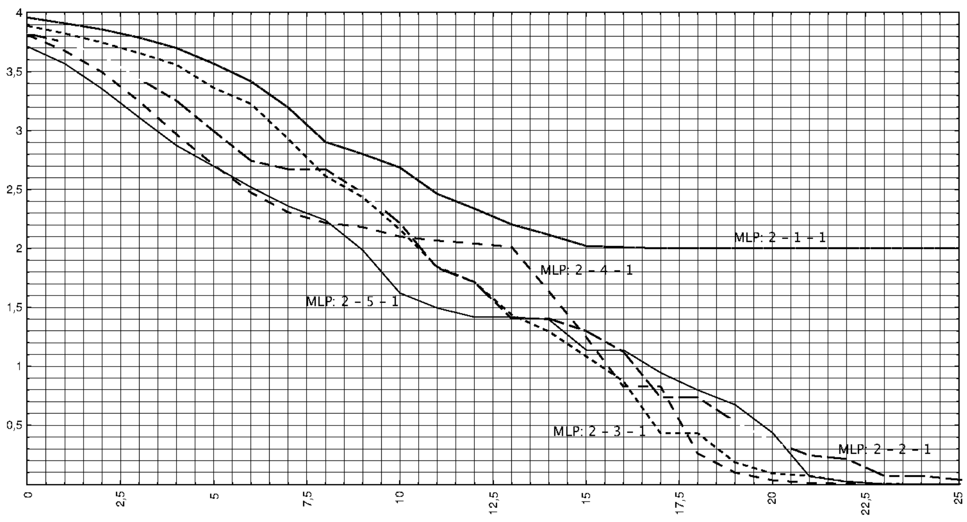


Fig. 5. Neural network topologies and training results obtained with a grammatical swarm for the topology and particle swarm optimization to train the net – XOR problem – *y* axis: mean squared error vs. *x* axis: iterations.

6.1. Classification datasets

- Wine: The wine recognition dataset (WINE) contains data from wine chemical analysis. It contains 178 examples of 13 attributes each that are classified into three classes.
- Glass: This dataset (GLASS) contains glass component analysis for glass pieces that belong to 6 classes. The dataset contains 214 examples with 10 attributes each.
- Pima Indians Diabetes: The PIMA dataset contains 768 examples of 8 attributes each that are classified into two categories: healthy and diabetic.
- Wisconsin Diagnostic Breast Cancer: The Wisconsin Diagnostic Breast Cancer dataset (WDBC) contains data for breast tumours. It contains 569 training examples of 30 attributes each that are classified into two categories.
- Circular Artificial data: The circular artificial dataset (CIRCULAR) contains 1000 examples that belong to two categories. The data in the first class belong to a circle and the data of the second class belong to a circular disc outside the first circle. Each example vector has two attributes. It is expanded by adding 3 more attributes generated randomly (noise) using a normal distribution.
- Spiral Artificial data: The spiral artificial dataset (SPIRAL) contains 1000 examples that belong to two classes (500 examples each). The data in the first class are created using the following formula: $x = 0.5tcos(0.08t), y = 0.5tcos(0.08 + \pi/2)$, and the second-class data using: $x = 0.5tcos(0.08t + \pi)$, $y = 0.5tcos(0.08t + 3\pi/2)$, The original features vector has two attributes (x, y).
- Spiral Artificial data 2: The second spiral dataset (SPIRAL2) is created as the first dataset. Its difference is that its primitive set is expanded by adding 3 more noisy attributes using normal distribution.
- Liverdisorder: This dataset contains blood analysis data from people with liver disorders. It consists of 345 examples of 6 attributes each.
- Ionosphere dataset: The ionosphere dataset contains data from the Johns Hopkins Ionosphere database. It contains 351 examples of 34 attributes each that are split into two classes.

6.2. Regression datasets

- BL: This dataset can be downloaded from StatLib (http://stat.cmu.edu/datasets/). It contains data from an experiment on the affects of machine adjustments on the time to count bolts.
- FA: The FA dataset contains percentage of body fat, age, weight, height, and ten body circumference measurements. The goal is to fit body fat to the other measurements.
- LW: This dataset is produced from a study that was to identify risk factors associated with giving birth to a low birth weight baby (weighing less than 2500 g). Data were collected on 189 women, 59 of which had low birth weight babies and 130 of which had normal birth weight babies.
- NT: This dataset contains data from a paper in the Journal of the American Medical Association that examined whether the true mean body temperature is 98.6 °F.
- PO: This dataset is available from StatLib (http://stat.cmu.edu/datasets/). It contains pollution data.
- PW: This dataset contains numeric prediction data using instance-based learning with encoding length selection.
- SN: This dataset contains data on a viticultural experiment that was conducted to investigate different methods of trellising and pruning.
- MB: This dataset is available from Smoothing Methods in Statistics (http://stat.cmu.edu/datasets/).
- BK: This dataset comes from Smoothing Methods in Statistics (http://stat.cmu.edu/datasets/).

6.3. Results

Results from the application of the proposed method against the methods *RPROP*, *BFGS* and MinFinder are listed, see Tables 4 and 5 . Each method was tested for different topologies of the resulting neural network (e.g. the number of hidden neurons) and the topology with the best results was selected. The genetic algorithm's parameters have been estimated experimentally after observing that the algorithm converges faster using these parameters. The parameters are: maximum number of generations: 500, population size: 500, chromosome length: 100, crossover rate: 0.95, mutation rate: 0.05, tournament size: 10.

The method proposed in this paper uses grammatical evolution in order to construct a neural network and particle swarm optimization to train it. The proposed method encodes in the grammar the network topology. The method is evaluated on 9 known classification and 9 known regression problems and compared against the state of the art methods: *RPROP*, *BFGS* and MinFinder. An accurate comparison of the four methods is presented that uses 10-fold cross validation and 30-fold experiment replication.

Table 4
Test error for the classifications problems. All mentioned methods are compared against the proposed method, combination of grammatical swarm and particle swarm optimization (GS + PSO).

	WINE	GLASS	PIMA	WDBC	CIRCULAR	SPIRAL	SPIRAL2	LIVER	IONSPHERE
RPROP	0.5993	0.7137	0.3319	0.3271	0.4494	0.4906	0.5022	0.4356	0.1578
BFGS	0.4886	0.5393	0.3656	0.2091	0.0795	0.4531	0.4847	0.3886	0.1708
MINFINDER	0.1178	0.4941	0.3004	**0.0489**	0.0857	**0.4343**	**0.4704**	0.3559	0.1627
GENETIC	0.1426	0.4801	0.3216	0.0687	**0.0784**	0.4358	0.4815	0.3584	0.1699
GS + PSO	**0.0541**	**0.5145**	**0.2133**	0.0512	0.0811	0.4490	0.4880	**0.3118**	**0.0999**

Bold values represent the best obtained mean squared error.

Table 5
Mean squared error for the regression problems. All mentioned methods are compared against the proposed method, combination of grammatical swarm and particle swarm optimization (GS + PSO).

	PO	BL	FA	LW	PW	NT	SN	MB	BK
RPROP	0.34	0.57	0.94	0.51	0.17	0.56	1.08	0.49	0.54
BFGS	0.25	0.33	0.44	0.91	0.21	3.14	2.31	0.68	4.44
MINFINDER	0.26	0.21	0.25	2.41	0.27	1.72	1.13	57.94	3.06
GENETIC	0.29	0.26	0.29	2.01	0.22	2.81	2.04	8.45	13.64
GS + PSO	**0.21**	**0.08**	**0.12**	**0.11**	**0.10**	**0.05**	**0.46**	**0.28**	**0.09**

Bold values represent the best obtained mean squared error.

The experimental results show that the proposed method outperforms the other methods, see Tables 4 and 5. Two major advantages of proposed model is that it is significantly faster than MinFinder by several orders of magnitude and is natively parallel. This gives it an edge over traditional methods since it can take advantage of distributed computing technologies.

7. Conclusion

This paper analyses a few optimization strategies in the natural computation area. Some competitive and collaborative models have been described in order to understand the ability to extract some biological ideas and then apply them in computational models. Such biological inspired models have proven to be a powerful tool in order to solve non-common problems in a collaborative/competitive way.

As a powerful application, neural networks can take advantage of such swarm optimization models. This paper has proposed a grammatical definition that chooses the best network topology by using grammatical swarm and network training.

Particle swarm optimization often fails when searching the global optimal solution when the objective function has a large number of dimensions. The reason of this phenomenon is not just existence of the local optimal solutions but also the degenerative process of the particles velocities (this means that particles stay in a subregion within the search area) [27]. This is a subplane which is defined by a finite number of particle velocities. Local optima problem in *PSO* is object of study. New proposals are based on specific equations [26,25] which modify the basic particle. They use a randomized method (e.g. mutation in evolutionary computations) for either to maintain particles velocities or to accelerate them. Although such improvements work well and have ability to avoid falls in the local optima, the problem of early convergence by the degeneracy of some dimensions still exists, even when local optima do not exist. Hence the *PSO* algorithm does not always work well for the high-dimensional function.

Usually, neural network parameters are in a high-dimensional space and then *PSO* algorithms are not very efficient ones when dealing with such individuals. Best solution could be the integration of *PSO* and *GA* in a new model *GPSO* taking advantages of both models, or at least to improve the impact of the high dimensional individuals in the *PSO* algorithm.

References

[1] N. Agarwal, X. Xu, Social computational systems, Journal of Computational Science 2 (2011) 189–192.

[2] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm Intelligence: Form Natural to Artificial Systems, Oxford University Press, 1999.

[3] J. Kennedy, R. Eberhart, Y. Shi, Swarm Intelligence, Morgan Kauff-man, 2001.

[4] T. Jayabarathi, S. Chalasani, Z.Z. Shaik, N.D. Kodali, Hybrid differential evolution and particle swarm optimization based solutions to short term hydro thermal scheduling, WSEAS Transactions on Power Systems 11 (2007) 245–254.

[5] N. Beji, B. Jarboui, M. Eddaly, H. Chabchoub, A hybrid particle swarm optimization algorithm for the redundancy allocation problem, Journal of Computational Science 1 (2010) 159–167.

[6] P. Haiguo, W. Zhixin, Z. Huaqiang, Cooperative-pso-based pid neural network integral control strategy and simulation research with asynchronous motor controller design, WSEAS Transactions on Circuits and Systems 8 (2009) 136–141.

[7] L. Ren, X. Jiang, G. Sheng, B. Wu, A new study in maintenance for transmission lines, WSEAS Transactions on Circuits and Systems 7 (2008) 35–37.

[8] M. O'Neill, C. Ryan, Grammatical evolution, IEEE Transactions on Evolutionary Computation 5 (2001) 349–358.

[9] M. O'Neill, C. Ryan, M. Keijzer, M. Cattolico, Crossover in grammatical evolution, Genetic Programming and Evolvable Machines 4 (2003) 67–93.

[10] J.R. Koza, D. Andre, F.H. Bennett, M. Keane, Genetic Programming 3: Darwinian Intention and Problem Solving, Morgan Kauff-man, 1999.

[11] J.R. Koza, M. Keane, M.J. Streeter, W. Mydlowec, J. Yu, G. Lanza, Genetic Programming IV: Routine Human-competitive Machine Intelligence, Kluwer Academic Publishers, 2003.

[12] A. Neme, S. Hernández, O. Neme, An electoral preferences model based on self-organizing maps, Journal of Computational Science 2 (4) (2011) 345–352.

[13] Y.H. Hu, J. Hwang, Handbook of Neural Network Signal Processing VE Profiling, CRC Press, 2001.

[14] S. Katagiri, Handbook of Neural Networks for Speech Recognition, Artech House, 2000.

[15] P.M.A. Sloot, The cross-disciplinary road to true computational science, Journal of Computational Science 1 (2010) 131.

[16] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings IEEE International Conference on Neural Networks, volume IV, 1995, pp. 1942–1948.

[17] R. Storn, K. Price, Minimizing the real functions of the icec'96 contest by differential evolution, in: IEEE Conference on Evolutionary Computation, 1996, pp. 842–844.

[18] M. Pant, R. Thangara, C. Grosan, A. Abraham, Hybrid differential evolution—particle swarm optimization algorithm for solving global optimization problems, in: IEEE International Conference on Digital Information Management, 2008, pp. 18–24.

[19] J. Garcia-Nieto, E. Alba, J. Apolloni, Noiseless functions black-box optimization: evaluation of a hybrid particle swarm with differential operators, in: Genetic and Evolutionary Computation Conference, 2009, pp. 2231–2238.

[20] M. O'Neill, A. Brabazon, Grammatical swarm: the generation of programs by social programming, Natural Computing 5 (2006) 443–462.

[21] M. O'Neill, A. Brabazon, Grammatical swarm, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2004, pp. 163–174.

[22] G.G. Yen, H. Lu, Hierarchical genetic algorithm for near-optimal feedforward neural network design, International Journal Neural Systems (2002) 31–43.

[23] M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: the rprop algorithm, in: Proceedings IEEE International Conference on Neural Networks, 1993, pp. 586–591.

[24] V.M. Rivas, J.J. Melero, P.A. Castillo, M.G. Arenas, J.G. Castellano, Evolving rbf neural networks for time-series forecasting with evrbf, Information Sciences 165 (2004) 207–220.

[25] T. Hendtlass, A particle swarm algorithm for high dimensional, multi-optima problem spaces, in: Proceedings of Swarm Intelligence Symposium, 2005, pp. 149–154.

[26] K. Parsopoulos, V.P. Plagianakos, G.D. Magoulas, Stretching technique for obtaining global minimizers through particle swarm optimization, in: Proceedings of Particle Swarm Optimization Workshop, 2001, pp. 22–29.

[27] M.R. Rapaic, Z. Kanovic, Z.D. Jelicic, A theoretical and empirical analysis of convergence related particle swarm optimization, WSEAS Transactions on Systems and Control 11 (2009) 541–550.

[28] M.J.D. Powell, A tolerant algorithm for linearly constrained optimization calculations, Mathematical Programming 45 (1989) 547–566.

[29] I.G. Tsoulos, I.E. Lagaris, Minfinder: locating all the local minima of a function, Computer Physics Communications 174 (2006) 166–179.

[30] C.G. Broyden, The convergence of a class of double-rank minimization algorithms, Journal of the Institute of Mathematics and Its Applications 6 (1970) 76–90.

[31] R.E. Fletcher, A new approach to variable metric algorithm, Computer Journal 13 (1970) 317–322.