Particle Swarm Optimization models applied to Neural Networks using the R language

NURIA GÓMEZ, LUIS F. MINGO
JESUS BOBADILLA, FRANCISCO SERRADILLA
Universidad Politécnica de Madrid
Escuela Universitaria de Informática
Crta. de Valencia km. 7, 28031 Madrid
SPAIN
{ngomez,lfmingo,jbobi,fserra}@eui.upm.es

JOSE A. CALVO MANZANO
Universidad Politécnica de Madrid
Facultad de Informática
Boadilla del Monte, 28660 Madrid
SPAIN
jacalvo@fi.upm.es

Abstract: There exists a clear difference between cooperative and competitive strategies. The former ones are based on the swarm colonies, in which all individuals share its knowledge about the goal in order to pass such information to other individuals to get optimum solution. The latter ones are based on genetic models, that is, individuals can die and new individuals are created combining information of alive one; or are based on molecular/celular behaviour passing information from one structure to another. A Grammatical Swarm model is applied to obtain the Neural Network topology of a given problem, training the net with a Particle Swarm algorithm in R. This paper just shows some ideas in order to obtain an automatic way to define the most suitable neural network topology for a given patter set. High dimension problem is also mentioned when dealing with the particle swarm algorithm.

Key–Words: Social Intelligence, Neural Networks, Grammatical Swarm, Particle Swarm Optimization.

1 Introduction

Natural sciences, and especially biology, represented 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. In the last decades or so, new emerging fields of so-called natural computing identify new (unconventional) computational para-digms in different forms. There are attempts to define and investigate new mathematical or theoretical models inspired by nature, as well as investigations into defining programming paradigms that implement computational approaches suggested by biochemical phenomena. Especially since Adleman's experiment, these investigations received a new perspective. One hopes that global system-level behaviour may be translated into interactions of a myriad of components with simple behaviour and limited computing and communication capabilities that are able to express and solve, via various optimizations, complex problems otherwise hard to approach.

A number of computational paradigms, inspired or gleaned from biochemical phenomena, are becoming of growing interest building a wealth of models, called generically Molecular Computing. New advances in, on the one hand, molecular and theoretical biology, and on the other hand, mathematical and computational sciences promise to make it possible in the near future to have accurate systemic models of complex biological phenomena.

2 Social Intelligence

This section shows some new computational paradigms that are based on the co-operation of individuals instead on the competition of individuals (typically modelled by genetic algorithms). Such social intelligence makes individuals to evolve towards the best solution using information from other individuals but none of them disappear. This is a new approach taken from the biology, in essence, social behaviour helps individuals to adapt to their environment, as it ensures that they obtain access to more information than that captured by their own senses.

Two popular variants of swarm models exist, those inspired by studies of social insects such as ant colonies, and those inspired by studies of the flocking behaviour of birds and fish.

2.1 Ant Colony Optimization

Ant colony optimization (ACO) is a class of optimization algorithms modelled on the actions of an ant colony. ACO methods are useful in problems that need to find paths to goals. Artificial 'ants' - simulation agents - locate optimal solutions by moving through a parameter space representing all possible solutions. Real ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions.[2] One variation on this approach is the bees algorithm, which is more analogous to the foraging patterns of the honey bee.

2.2 Particle Swarm Optimization

Particle swarm optimization (PSO) is a global optimization algorithm for dealing with problems in which a best solution can be represented as a point or 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 [8, 1]. Particles then move through the solution space, and are evaluated according to some fitness criterion after each timestep. Over time, particles are accelerated towards those particles within their communication grouping which have better fitness values. The main advantage of such an approach over other global minimization strategies such as simulated annealing is that the large number of members that make up the particle swarm make the technique impressively resilient to the problem of local minima [6, 5, 17].

3 Grammatical Swarm

Grammatical Swarm (GS) adopts a Particle Swarm learning algorithm coupled to a Grammatical Evolution (GE) genotype-phenotype mapping to generate programs in an arbitrary language. Grammatical Evolution (GE) is an evolutionary algorithm that can evolve computer programs in any language [12, 13], and can be considered a form of grammar-based genetic programming. Rather than representing the programs as parse trees, as in GP [9, 10], a linear genome representation is used. A genotype-phenotype mapping is employed such that each individuals variable length binary string, contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. The grammar allows the generation of programs in an arbitrary language that are guaranteed to be syntactically correct, and as such it is used as a generative grammar, as opposed to the classical use of grammars in compilers to check syntactic correctness of sentences. The user can tailor the grammar to produce solutions that are purely syntactically constrained, or they may incorporate domain knowledge by biasing the grammar to produce very specific forms of sentences. BNF is a notation that represents a language in the form of production rules.

Let us suppose the following BNF grammar:

```
<expr> :: = <expr><op><expr>
          | <var>
<op> :: = +
        | -
        | *
<var> :: = x
         | y
```

And the following genotype:

14	8	27	254	5	17	12

In the example individual (see figure 3), the left-most `<expr>` in `<expr> <op> <expr>` is mapped by reading the next codon integer value 240 and used in $240\%2 = 0$ to become another `<expr> <op> <expr>`. The developing program now looks like `<expr> <op> <expr> <op> <expr>`. Continuing to read subsequent codons and always mapping the left-most non-terminal the individual finally generates the expression `y * x - x - x + x`, leaving a number of unused codons at the end of the individual, which are deemed to be introns and simply ignored.

This is the classic benchmark problem in which evolution attempts to find the five input even-parity boolean function [2]. The grammar adopted here is:

```
<prog> ::= <expr>
<expr> ::= <expr> <op> <expr>
         | ( <expr> <op> <expr> )
         | <var> | <pre-op> ( <var> )
<pre-op> ::= not
<op> ::= "|" | & | ^
<var> ::= d0 | d1 | d2 | d3 | d4
```

The result is given by the best individual, see transcript bellow. Figure 3 shows a graphic with the best, average and variance of the swarm population. This figure has been obtained using the *GEVA* simulator [2].

```
( not ( d1 ) | d2 ^ d4 ) &
not ( d3 ) ^ ( not ( d1 ) &
( not ( d2 ) &
( not ( d2 ) |
( d1 ^ not ( d3 ) ^ not ( d1 ) ^
( not ( d1 ) ^ ( d0 | not ( d4 ) ) ) ) ) ^ d4 )
 ^ not ( d0 ) ) ^ d1
```

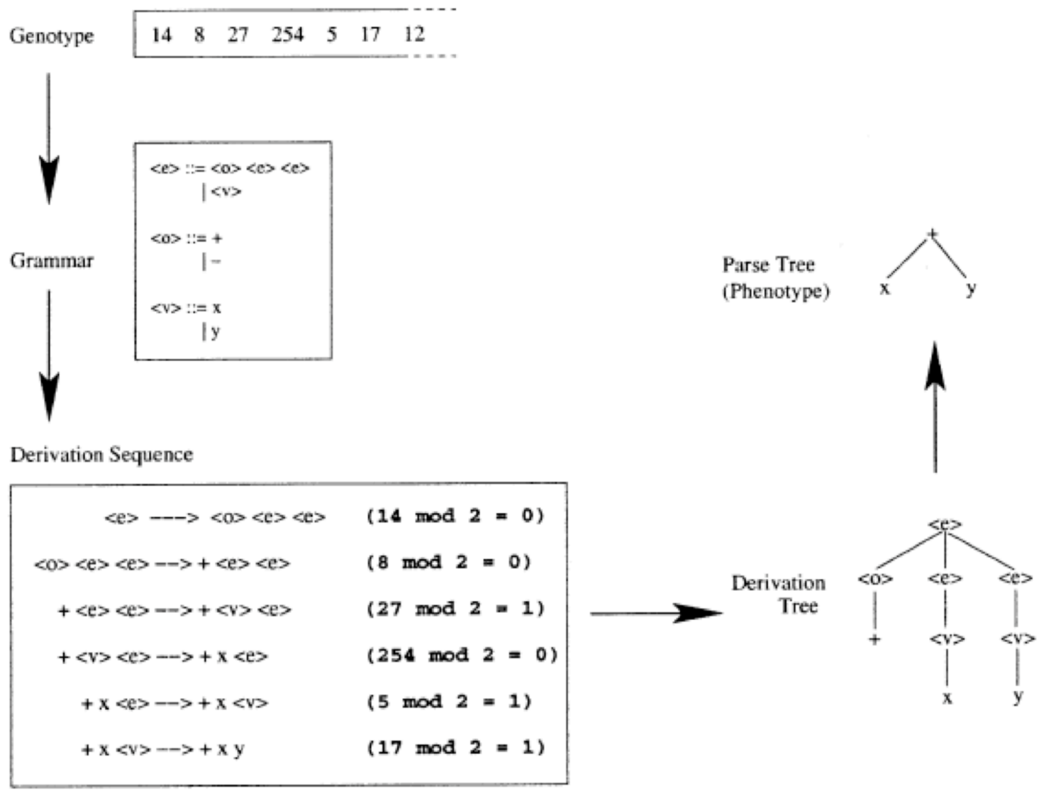
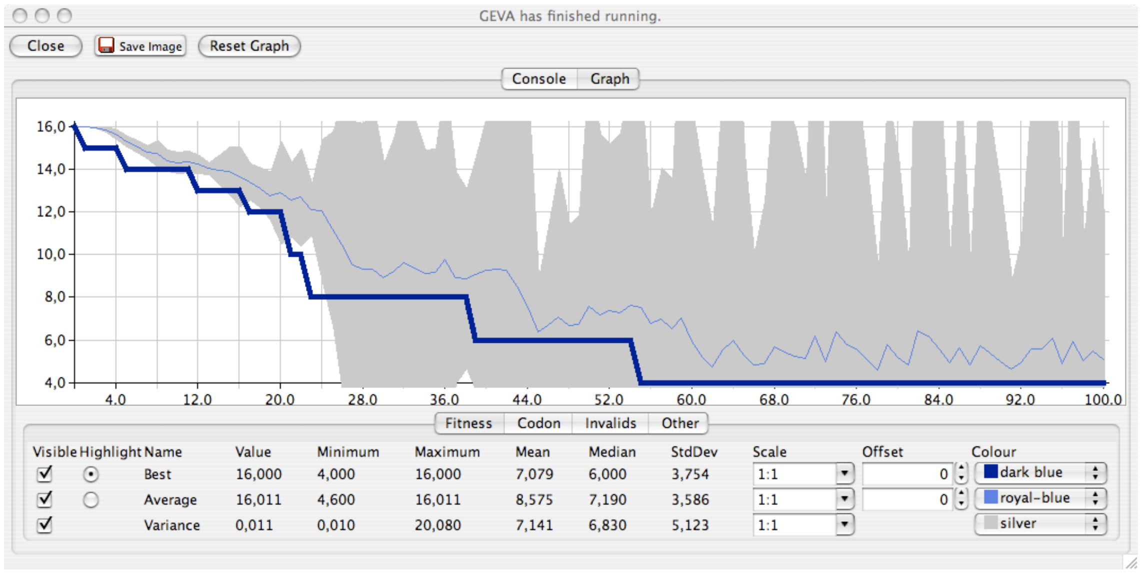

Genotype
14 8 27 254 5 17 12
Grammar
<e> ::= <o> <e> <e>
| <v>
<o> ::= +
| -
<v> ::= x
| y
Derivation Sequence
<e> ---> <o> <e> <e> (14 mod 2 = 0)
<o> <e> <e> ---> + <e> <e> (8 mod 2 = 0)
+ <e> <e> ---> + <v> <e> (27 mod 2 = 1)
+ <v> <e> ---> + x <e> (254 mod 2 = 0)
+ x <e> ---> + x <v> (5 mod 2 = 1)
+ x <v> ---> + x y (17 mod 2 = 1)
Derivation Tree
<e>
<o> <e> <e>
+ <v> <v>
x y
Parse Tree (Phenotype)
+
x y

Figure 1: Grammatical Swarm Concepts.



Visible	Highlight	Name	Value	Minimum	Maximum	Mean	Median	StdDev	Scale	Offset	Colour
✓	●	Best	16,000	4,000	16,000	7,079	6,000	3,754	1:1	0	🟦 dark blue
✓	○	Average	16,011	4,600	16,011	8,575	7,190	3,586	1:1	0	🟦 royal-blue
✓		Variance	0,011	0,010	20,080	7,141	6,830	5,123	1:1		⬜ silver

Figure 2: Results of *even-5-parity* problem simulated with GEVA [2].

4 Neural Networks and Grammatical Swarm

Neural networks are non-linear systems whose structure is based on principles observed in biological neuronal systems. A neural network could be seen as a system that can be able to answer a query or give an output as answer to a specific input. The in/out combination, i.e. the transfer function of the network is not programmed, but obtained through a "training" process on empiric datasets. In practice the network learns the fuction that links input together with output by processing correct input/output couples. Actually, for each given input, within the learning process, the network gives a certain 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 best solution for all the examples. These parameters we are speaking about are essentially the weights or linking factors between each neuron that forms our network.

Neural Networks application fields are tipically those where classic algorithms fail because of their unflexibility (they need precise input datasets). Usually problems with unprecise input datasets are those whose number of possible input datasets is so big that they cant be classified. For example in image recognition are used probabilistic algoithms whose efficency is lower than neural networks and whose caratheristics are low flexibility and high development complexity. Another field where classic algorithms are in troubles is the analisys of those phenomena whose matematical rules are unknown. There are indeed rather complex algorithms which can analyse these phenomena but, from comparisons on the results, it comes out that neural networks result far more efficient [3, 7]: these algorithms use Fouriers trasform to decompose phenomena in frequential components and for this reason they result highly complex and they can only extract a limited number of harmonics generating a big number of approximations. A neural network trained with complex phenomenas data is able to estimate also frequential components, this means that it realizes in its inside a Fouriers trasform even if it was not trained for that! One of the most important neural networks applications is undoubtfully the estimation of complex phenomena such as meteorological, financial, socio-economical or urban events.

Thanks to a neural network its possible to predict, analyzing hystorical series of datasets just as with these systems but there is no need to restrict the problem or use Fouriers tranform. A defect common to all those methods its to restrict the problem setting certain hypothesis that can turn out to be wrong. We just have to train the neural network with hystorical series of data given by the phenomenon we are studying. Calibrating a neural network means to determinate the parameters of the connections (synapsis) through the training process. Once calibrated there is need to test the netowrk efficency with known datasets, which has not been used in the learning process. There is a great number of Neural Networks which are substantially distingushed by: type of use, learning model (supervised/non-supervised), learning algorithm, architecture, etc.

But the most common trouble consists on what architecture must be used in order to get better results. That is the reason this paper proposes a Grammatical Swarm algorithm to get a right architecture/topology. Moreover, the training process can be done using Particle Swarm Optimization. With these two models the whole neural network is obtained (topology and weights) using ideas from social intelligence. Next sections describe how to implement a model in order to get a neural network topology and how to train this topology.

4.1 Training using Particle Swarm Optimization

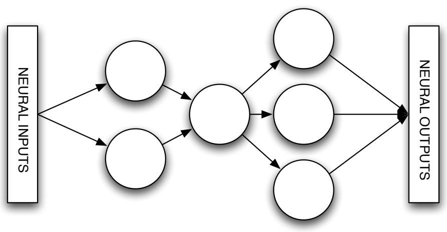
Given a fixed neural network architecture, all weights in connections can be coded as a genotype and apply the particle swarm optimization algorithm in order to train the network where the fitness function must be the mean squared error of the net with the training set. Some variations can be done just using validation and testing sets to get better fitness values with more generalization properties.

Equations used in the particle swarm optimization training process are the following ones, where 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. This equations define how the genotype values are changing along iterations, in our case, how neural network weights are changing.

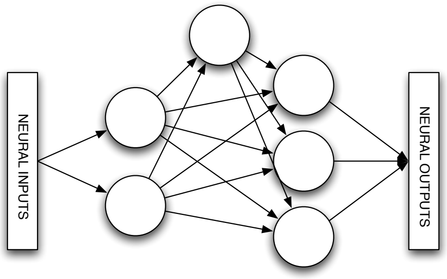
$$\begin{aligned} v_{in}(t+1) = w v_{in}(t) + \\ c_1 R_1 (p_{in} - x_{in}(t)) + \\ c_2 R_2 (p_{gn} - x_{in}(t)) \end{aligned} \tag{1}$$

$$x_{in}(t+1) = x_{in}(t) + v_{in}(t+1) \tag{2}$$

Previous equations will modified the network weights till a stop conditions is achieved, that is, a lower mean squared error or a maximum number of iterations is reached.



a) First grammar with individual
`{2, 1, 3}`



b) Second grammar with
`{ 2 -- 1, 2 --, 1 -- 1 --, 3 -- 1 --}`

Figure 3: Neural Network obtained using grammatical swarm.

Here it is the algorithm to train the network with the particle swarm optimization model:

```
Initialize population with random values $[-1,1]$
WHILE not finish condition is satisfied
    FOR i=1 TO n DO
        Compute fitness particle Ji (MSE of the net)
        IF Ji < pid
            pid = Ji
        END IF
        IF Ji < pig
            pig = Ji
        END IF
        Compute new velocity of i (equation 2)
        Compute new position of i (equation 1)
    END FOR
END WHILE
```

This particle swarm optimization model is similar to genetic algorithms model but it uses a collaborative approach instead a competitive one.

Figure 4 shows different neural network architectures and their learning evolution according to the number of PSO iteration (maximunm 30 iterations). We can see that any multilayer perceptron, with at least 2 hidden neurons, can successfully solve the XOR problem. While a multilayer perceptron with only one hidden neuron will never achive a mean squared error lower than 2.0.

4.2 Grammatical Topology

Previous PSO model applied to a fixed neural network is a good solution to train a net, but it does not define any kind or topology properties it only obtains the best weights configuration. Next 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 just with consecutive layers, that is, a classical Multilayer Perceptron, see figure 3–a).

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

Next grammar is able to generate feed-forward connections not only with one consecutive layer but also with more than one consecutive layers, see figure 3–b). 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, could be summarize 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 appropiate.
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.

Figure 4 shows different multilayer perceptrons obtained with previous grammars. Depending on the topology that each individual codes the mean squared error will be lower or greater.

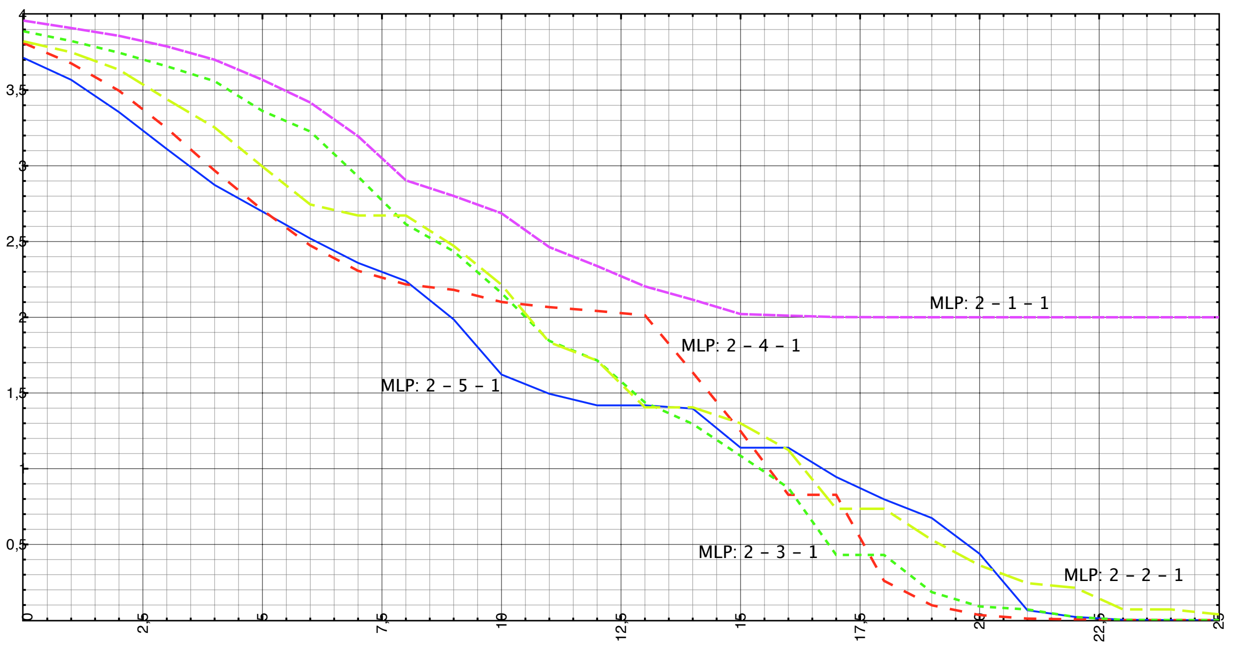


Figure 4: Neural Network Topologies and Training results obtained with a grammatical swarm for the topology and particle swarm optimization to train the net – XOR problem –.

5 R Implementation of the PSO

R is a system for statistical computation and graphics [16]. It provides, among other things, a programming language, high level graphics, interfaces to other languages and debugging facilities. The R language is a dialect of S which was designed in the 1980s and has been in widespread use in the statistical community since. Its principal designer, John M. Chambers, was awarded the 1998 ACM Software Systems Award for S. The language syntax has a superficial similarity with C, but the semantics are of the FPL (functional programming language) variety with stronger affinities with Lisp and APL. In particular, it allows computing on the language, which in turn makes it possible to write functions that take expressions as input, something that is often useful for statistical modeling and graphics.

There are many modified versions of PSO by improving convergence property to a certain problem. While, a standard PSO is defined in Kennedy [8] to give a real standard for PSO studies. Thus, a lot of studies have demonstrated the effectiveness of PSO family in optimizing various continuous and discrete optimization problems. And a plenty of applications of PSO, such as the neural network training, PID controller tuning, electric system optimisation have been studied and achieved well results. Using the satandard PSO, table 1 shows the particle swarm algorithm coded using R.

This implementation follows the classical steps in the PSO algorithm:

- initialize the population with random values,
- compute fitness of all indiviuals and choose the winner,
- modify positon and velocity of individuals according to the winner.

Fitness function to evaluate individuals (weight parameters of the neural network) in the particle swarm algorithm is shown in table 2. First of all, a mapping from individuals to neural network parameters is defined (`individualToNn`) and after that, the fitness function will build a multilayer perceptron with such parameters and compute the mean squared error of the net as the fitness of the particle swarm algorithm using the `pso_settings` data structure that will be analyzed below.

Some parameters must be defined in order to run the particle swarm algorithm (see equation (1)) among them, the $c1$ and $c2$ parameters, the initial velocity $v_{in}(0)$ and the weight inertia w, the number and length of individuals, the number of iterations and the function to compute the fitness evaluation. Fitness function depends on the problem, when dealing with neural networks such fitness function will correspond to

Table 1: Listing of the particle swarm optimization algoritm implemented in R

```
######## TOOLS MATRIX GENERATION
mRUniform <- function(m,n) {
  return(array(runif(m*n,min=-0.5,max=0.5), dim = c(m,n)))
}

mRNormal <- function(m,n) {
  return(array(rnorm(m*n,0,0.5), dim = c(m,n)))
}

############## DEFAULT FITNESS FUNCTION
fit <- function(ind) {
  sum(abs(ind))
}

############# ALGORITHM
pso <- function(settings) {
  fit_evolution <- rnorm(settings$it)
  population <- mRNormal(settings$num_ind, settings$length_ind)
  velocity <- array(settings$init_vel*rnorm(1), c(settings$num_ind, settings$length_ind))
  fitness <- array(0,c(settings$num_ind,1))
  best <- population
  best_fitness <- fitness

  best_individual <- population[which.min(fitness),]
  best_individual_fitness <- min(fitness)

  for (j in 1:settings$num_ind) {
    fitness[j] <- settings$func(population[j,])
  }
  best <- population
  best_fitness <- fitness

  for(i in 1:settings$it) {
    for (j in 1:settings$num_ind) {
      fitness[j] <- settings$func(population[j,])
      if (!is.nan(fitness[j])) {
        if (fitness[j] < best_fitness[j]) {
          best_fitness[j] <- fitness[j]
          best[j] <- population[j]
        }
      }
    }

    best_individual_fitness <- min(best_fitness)
    best_individual <- best[which.min(best_fitness),]
    fit_evolution[i] <- best_individual_fitness

    velocity_update <- settings$c1 * mRUniform(settings$num_ind,settings$length_ind) * (best - population) +
      settings$c2 * mRUniform(settings$num_ind,settings$length_ind)  * (best_individual - population)

    velocity  <- settings$inertia * velocity + (velocity_update)
    population  <- population + velocity

    cat ("Iteration: ", i, " fitness: ", best_individual_fitness, "\n")
  }

  list(best_individual=best_individual, best_fitness=best_individual_fitness, fit_evo=fit_evolution)
}
```

Table 2: Fitness function of the particle swarm optimization algorithm applied to Neural Networks. Mean Squared Error is used as the fitness function.

```
evaluate <- function(ind) {
  z <- individualToNn(ind,nn_settings$inputs,nn_settings$hiddens,nn_settings$outputs)
  neurons <- c(nn_settings$inputs,nn_settings$hiddens,nn_settings$outputs)
  output <- mlp(entrada,z[[1]],z[[2]],neurons,functions)
  mean((salida-output)*(salida-output))
}

individualToNn <- function(ind, inp, hid, out) {
  ########################### WEIGHTS
  weights <- array(0,dim=c(inp,hid))
  for (i in 1:inp)
    for (j in 1:hid)
      weights[i,j] <- ind[(i-1)*(hid)+j]
  x <- list(weights)
  weights <- array(0,dim=c(hid,out))
  for (i in 1:hid)
    for (j in 1:out)
      weights[i,j] <- ind[(hid*inp)+(i-1)*(out)+j]
  x <- list(x[[1]],weights)
  ########################### BIAS
  y <- list(NULL)
  dis <- 1:hid
  for (j in 1:hid)
    dis[j] <- ind[(inp*hid)+(hid*out)+j]
  y <- list(y[[1]],dis)
  dis <- 1:out
  for (j in 1:out)
    dis[j] <- ind[(inp*hid)+(hid*out)+(hid)+j]
  y <- list(y[[1]],y[[2]],dis)
  ##########
  list(x,y)
}
```

the mean squared error or some other measure able to evaluate the desired response of the network, see table 2.

```
pso_settings <- list(
  c1 = 1.0,
  c2 = 2.0,
  inertia = 0.75,      # inertia
  init_vel = 1.00,     # initial velocity
  num_ind = 20,        # population size
  length_ind = tamano_ind,    # individual length
  it = 10,             # number of generations
  func = evaluate      # fitness function
)
```

Main problem is the dimension of individual in the particle swarm algorithm. Figure 5 shows such problem, as the number of dimensions increases then the convergence of the algorithm is worst. This is a handicap in neural networks since a multilayer perceptron with i inputs, o outputs, h hidden neurons will have, at least, $(i + 1) * h + (h + 1) * o$ dimensions. Some PSO variants could be taken into account to avoid such problem but it is not in the scope of this paper.

Neural networks parameters can be output when the particle swarm optimization *training* is finished. Individual information is decoded as network weights and bias values, corresponding to the best fitness evaluation (mean squared error in this neural network example). Next output shows the network parameters of the XOR problem for a multilayer perceptron with 2 hidden neurons. This is one of the 9 rounds, see table 3, each round starts with a different random population in order to evaluate the convergence of algorithm.

```
Desired Output
1         1         -1         -1
Neural Network Output
0.999756 0.9485043 -0.9999969 -0.9682595

WEIGHT VALUES
[[1]]
[[1]][[1]]
         [,1]       [,2]      [,3]
[1,] 4.160567  2.8819753 -1.287110
[2,] 3.637939 -0.2741027  3.088520

[[1]][[2]]
          [,1]
[1,] -7.348808
[2,]  5.751979
[3,]  3.396366

BIAS VALUES
[[2]]
[[2]][[1]]
     [,1]
[1,]    0

[[2]][[2]]
[1]  4.3329053 -0.7112471  0.6386734

[[2]][[3]]
[1] 3.010704

MSE: 0.0009148323
```

This R implementation is focused only on the PSO *training* not in the grammatical swarm topology learning. Future works will present an implementation of the whole process (grammatical swarm and particle swarm) in R as the shown in Java (see section 4).

6 Conclusions

This paper has reviewed some natural computation strate- gies as a survey concerning optimization strategies. Some competitive and collaborative models has been exposed in order to understand the ability to extract some biological concepts and apply them in computational models as de- scribed along the paper. Such bio-inspired models have proof 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 in order to choose the better network topology using grammatical swarm and the training of such networks (to compute the fitness function) is done with the PSO approach.

Particle Swarm Optimization is often failed in searching the global optimal solution in the case of the objective function has a large number of dimensions. The reason of this phenomenon is not only existence of the local optimal solutions, the velocities of the particles sometimes lapsed into the degeneracy, so that the successive range is restricted in the sub-plain of the whole search hyper-plain [15]. The sub-plane that is defined by finite number of particle velocities is a partial space in the whole search space. The issue of local optima in PSO has been studied and proposed several modifications on the basic particle driven equation [14, 11, 4]. There used a kind of adaptation technique or randomized method (e.g. mutation in evolutionary computations) to keep particles velocities or to accelerate them. Although such improvements work well and have ability to avoid fall in the local optima, the problem of early convergence by the degeneracy of some dimensions is still remaining, even if there are no local optima. 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 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.

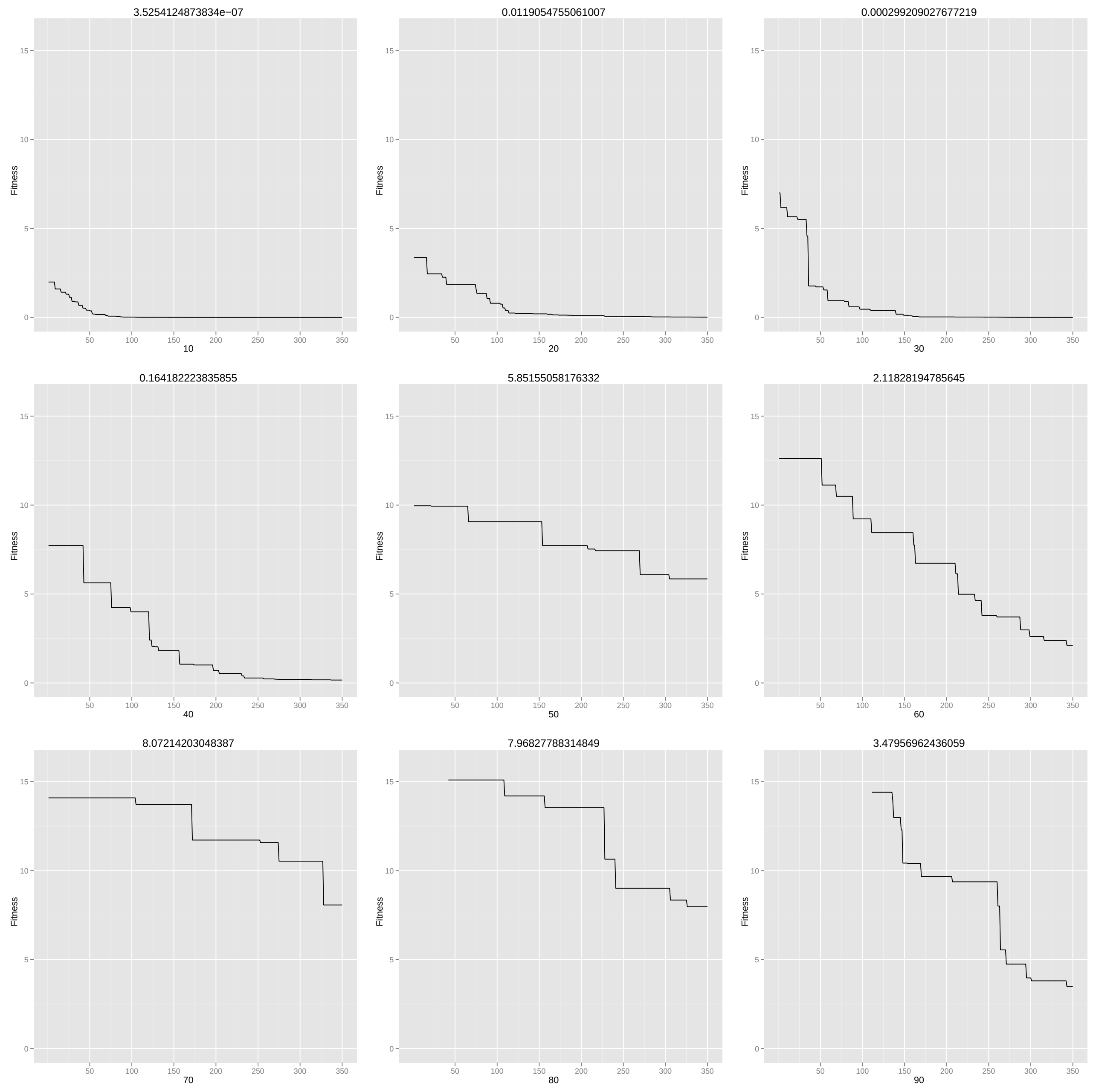


Figure 5: PSO algorithm with different dimensions (from $N = 10$ to $N = 90$) applied to function $\sum_{i=0}^{N} x_i^2$. Convergence of the pso is better with lower dimensions than with higher ones.

Table 3: Results of the training a XOR Neural Network (MLP 2-3-1) with the particle swarm algorithm.

	Output 1	Output 2	Output 3	Output 4	MSE
Round 1	0.9999917	1	-0.6773486	-1	0.02602597
Round 2	1	1	-0.9999999	-0.999962	3.594795e-10
Round 3	0.06365612	0.623899	-0.9990188	-0.994997	0.2545544
Round 4	0.4452416	0.8356855	-0.9338923	-0.9290893	0.08603868
Round 5	0.9999777	0.9999724	-0.999958	-0.9999847	8.14248e-10
Round 6	0.3620248	0.6908745	-0.9815965	-0.7450217	0.1419809
Round 7	0.6807203	0.9488	-0.9816844	-0.929081	0.02748148
Round 8	1	1	-0.9999745	-0.9830038	7.221789e-05
Round 9	0.4224904	0.7102605	-0.9966216	-0.6679007	0.1319419

Acknowledgements: The research was supported by the Spanish Research Agency projects CCG08-UAM TIC-4425-2009 and TEC2007-68065-C03-02.

References:

[1] E. Bonabeau, M. Dorigo and G. Theraulaz, "Swarm Intelligence: From Natural to Artificial Systems", Oxford University Press, Oxford, 1999.

[2] Grammatical Evolution Group. http://www.grammatical-evolution.org/

[3] Yu Hen Hu, Jeng-Neng Hwang; "Handbook of Neural Network Signal Processing VE Profiling". ISBN: 0849323592. CRC Press. September 2001.

[4] T. Hendtlass; "A particle swarm algorithm for high dimensional, multi-optima problem spaces", Proceedings of Swarm Intelligence Symposium 2005. pp. 149-154, Pasadena, June 2005

[5] Piao Haiguo, Wang Zhixin, Zhang Huaqiang, "Cooperative-PSO-Based PID Neural Network Integral Control Strategy and Simulation Research with Asynchronous Motor Controller Design", WSEAS Transactions on Circuits and Systems Volume 8, pp. 136-141, ISSN: 1109-2734, 2009.

[6] T Jayabarathi, Sandeep Chalasani, Zameer Ahmed Shaik, Nishchal Deep Kodali; "Hybrid Differential Evolution and Particle Swarm Optimization Based Solutions to Short Term Hydro Thermal Scheduling", WSEAS Transactions on Power Systems Issue 11, Volume 2, pp. , ISSN: 1790-5060, 2007.

[7] Shigeru Katagiri; "Handbook of Neural Networks for Speech Processing". Artech House. ISBN 0890069549. 2000.

[8] J. Kennedy, R. Eberhart and Y. Shi, "Swarm Intelligence", Morgan Kauff-man, San Mateo, California, 2001.

[9] J.R. Koza, D. Andre, F.H. Bennett III and M. Keane "Genetic Programming 3: Darwinian Invention and Problem Solving", Morgan Kaufmann, 1999.

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

[11] J. J. Liang, A. K. Qin, P. N. Suganthan & S. Baskar; "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions", IEEE Transactions on Evolutionary Computation, Vol.10, No.3, 281-295, 1089-778X, 2006.

[12] M. ONeill and C. Ryan, "Grammatical Evolution", *IEEE Trans. Evolutionary Computation*, Vol. 5, No.4, 2001.

[13] M. ONeill, C. Ryan, M. Keijzer and M. Cattolico, "Crossover in Grammatical Evolution", *Genetic Programming and Evolvable Machines*, Vol. 4 No. 1. Kluwer Academic Publishers, 2003.

[14] K. Parsopoulos, V. P. Plagianakos, G. D. Magoulas & M. N. Vrahatis; "Stretching technique for obtaining global minimizers through particle swarm optimization", Proceedings of the Particle Swarm Optimization Workshop, pp. 22-29, Indianapolis, 2001.

[15] Milan R. Rapaic, Zeljko Kanovic, Zoran D. Jelicic; "A Theoretical and Empirical Analysis of Convergence Related Particle Swarm Optimization", WSEAS Transactions on Systems and Control, Volume 4, pp. , ISSN: 1991-8763, 2009.

[16] R Development Core Team, "R: A Language and Environment for Statistical Computing", R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0, 2009.

[17] Lijia Ren, Xiuchen Jiang, Gehao Sheng, Wu B; "A New Study in Maintenance for Transmission Lines", WSEAS Transactions on Circuits and Systems Volume 7, pp. 53-37, ISSN: 1109-2734, 2008.