Chapter 3
Evolutionary Algorithm Training of Higher Order Neural Networks

M. G. Epitropakis
University of Patras, Greece

V. P. Plagianakos
University of Patras, Greece

M. N. Vrahatis
University of Patras, Greece

ABSTRACT

This chapter aims to further explore the capabilities of the Higher Order Neural Networks class and especially the Pi-Sigma Neural Networks. The performance of Pi-Sigma Networks is evaluated through several well known neural network training benchmarks. In the experiments reported here, Distributed Evolutionary Algorithms are implemented for Pi-Sigma neural networks training. More specifically, the distributed versions of the Differential Evolution and the Particle Swarm Optimization algorithms have been employed. To this end, each processor of a distributed computing environment is assigned a subpopulation of potential solutions. The subpopulations are independently evolved in parallel and occasional migration is allowed to facilitate the cooperation between them. The novelty of the proposed approach is that it is applied to train Pi-Sigma networks using threshold activation functions, while the weights and biases were confined in a narrow band of integers (constrained in the range [-32, 32]). Thus, the trained Pi-Sigma neural networks can be represented by using only 6 bits. Such networks are better suited for hardware implementation than the real weight ones and this property is very important in real-life applications. Experimental results suggest that the proposed training process is fast, stable and reliable and the distributed trained Pi-Sigma networks exhibit good generalization capabilities.

DOI: 10.4018/978-1-61520-711-4.ch003

INTRODUCTION

Evolutionary Algorithms (EAs) are nature inspired methods solving optimization problems, which employ computational models of evolutionary processes. Various evolutionary algorithms have been proposed in the literature. The most important ones are: Genetic Algorithms (Goldberg, 1989; Holland, 1975), Evolutionary Programming (Fogel, 1996; Fogel et al., 1966), Evolution Strategies (Hansen & Ostermeier, 2001; Rechenberg, 1994), Genetic Programming (Koza 1992), Particle Swarm Optimization (Kennedy & Eberhart, 1995) and Differential Evolution algorithms (Storn & Price, 1997). The algorithms mentioned above share the common conceptual base of simulating the evolution of a population of individuals using a predefined set of operators. Generally, the operators utilized belong to one of the following categories: the *selection* and the *search* operators. The most commonly used search operators are the *mutation* and the *recombination*.

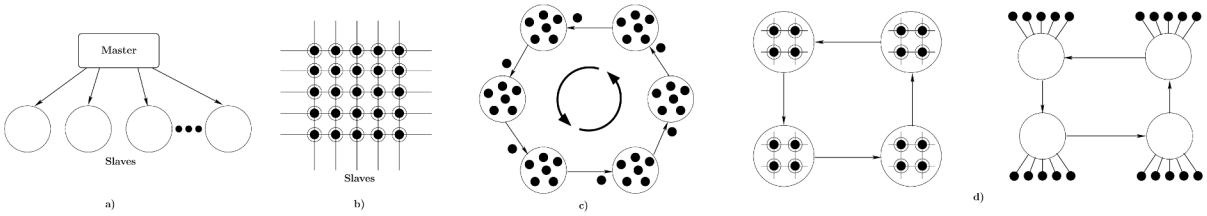
In general, EA's are parallel and distributed implementations and they are inspired by niche formation. Niche formation is a common biological phenomenon (Baeck et al., 1997). Niches could aid the differentiation of the species by imposing reproduction restrictions. Many natural environments can lead to niche formation. For example, remote islands, high mountains and isolated valleys, restrict the species and therefore the evolution process. Although diversity tends to be low in each subpopulation, overall population diversity is maintained through isolation. However, occasionally an individual escapes and reaches nearby niches, further increasing the diversity of their populations (Baeck et al., 1997).

In this chapter, we study the class of Higher Order Neural Networks (HONNs) and in particular Pi-Sigma Networks (PSNs), which were introduced by Shin and Ghosh (1991a). Although, in general, PSNs employ fewer weights and processing units than HONNs, and manage to indirectly incorporate many of HONNs' capabilities and strengths. PSNs have effectively addressed several difficult tasks, where traditional Feedforward Neural Networks (FNNs) are having difficulties, such as zeroing polynomials (Huang et al., 2005) and polynomial factorization (Perantonis et al., 1998). Here, we study PSN's performance on several well known neural network training problems.

The novelty of the proposed approach is that the PSNs were trained with small integer weights and threshold activation functions, utilizing distributed Evolutionary Algorithms. More specifically, modified distributed versions of the Differential Evolution (DE) (Plagianakos & Vrahatis, 2002; Storn & Price 1997) and the Particle Swarm Optimization (PSO) (Clerck, 2006; Kennedy, & Eberhart, 1995) algorithms have been used. DE and PSO have proved to be effective and efficient optimization methods on numerous hard real-life problems (see for example Branke, 1995; Clerc, 2006; Jagadeesan et al., 2005; Magoulas et al., 2004; Parsopoulos & Vrahatis, 2004, 2002; Plagianakos et al., 2005, 1998; Plagianakos & Vrahatis, 1999, 2002; Storn, 1999; Tasoulis et al., 2004, 2005). The distributed EAs has been designed keeping in mind that the resulting integer weights and biases require less bits to be stored and the digital arithmetic operations between them are easier to be implemented in hardware. An additional advantage of the proposed approach is that no gradient information is required; thus (in contrast to classical methods) no backward passes were performed.

Hardware implemented PSNs, with integer weights and threshold activation functions, can continue training if the input data are changing (on-chip training) (Plagianakos et al., 2005; Plagianakos & Vrahatis, 2002). Another advantage of neural networks with integer weights and threshold activation functions is that the trained neural network is to some extend immune to noise in the training data. Such networks only capture the main feature of the training data. Low amplitude noise that possibly contaminates the

Figure 1. Parallel and distributed evolutionary algorithms: a) single-population master-slave algorithms, b) single-population fine-grained algorithms, c) multiple-population coarse-grained algorithms, and d) hybrid approaches



training set cannot perturb the discrete weights, because those networks require relatively large variations to "jump" from one integer weight value to another (Plagianakos & Vrahatis, 2002).

If the network is trained in a constrained weight space, smaller weights are found and less memory is required. On the other hand, the network training procedure can be more effective and efficient when larger integer weights are allowed. Thus, for a given application a trade off between effectiveness and memory consumption has to be considered.

The remaining of this chapter is organized as follows. The first Section reviews various parallel Evolutionary Algorithm implementations, while the next section briefly describes the mathematical model of HONNs and PSNs. The next section is devoted to the presentation of the distributed DE and PSO optimization algorithms. Finally, this chapter ends with extensive experimental results in well-known and widely used benchmark problems, followed by discussion and concluding remarks.

BACKGROUND

Parallel And Distributed Evolutionary Algorithms

Following the biological niche formation many parallel and distributed Evolutionary Algorithm implementations exist (Alba & Tomassini, 2002; Cantu-Paz, 1998, 2000; Gorges-Schleuter, 1991; Gustafson & Burke, 2006; Sprave, 1999). The most widely known are (Alba & Tomassini, 2002; Cantu-Paz, 1998; Sprave, 1999):

1. single-population (global) master-slave algorithms
2. single-population fine-grained algorithms
3. multiple-population coarse-grained algorithms
4. hierarchical parallel algorithms (hybrid approach)

In EA literature *single-population fine-grained algorithms* are also called *cellular EAs* (cEAs). The *multiple-population coarse-grained algorithms* are also known as *island models* or *distributed EAs* (dEAs). These two approaches are most popular among EA researchers and seem to provide a better sampling of the search space. Additionally, they improve the numerical and runtime behavior of the basic algorithm (Alba & Tomassini, 2002; Baeck et al., 1997;Cantu-Paz, 1998; Sprave, 1999).

In a *master-slave* implementation there exists a single panmictic population (selection takes place globally and any individual can potentially mate with any other), but the evaluation of the fitness of each individual is performed in parallel among many processors. This approach does not affect the behavior of the EA algorithm; the execution is identical to a basic sequential EA, but performed much faster (depending on the number of available processors).

According to the cEA approach each individual is assigned to a single processor and the selection and reproduction operators are limited to a small local neighborhood. Neighborhood overlap is permitting some interaction among all the individuals and allows a smooth diffusion of good solutions across the population.

We must note that one could use a uniprocessor machine to run cEAs and dEAs and still get better results than with sequential panmictic EAs. The main difference between cEAs and dEAs is the separation of individuals into distinct subpopulations (islands). In biological terms, dEAs resembles distinct semi-isolated populations in which evolution takes place independently. dEAs are more sophisticated as they occasionally exchange individuals between subpopulations, utilizing the migration operator. The migration operator defines the topology, the migration rate, the migration interval, and the migration policy (Cantu-Paz, 1998, 2000; Skolicki, 2005; Skolicki & De Jong, 2005).

The migration topology determines island interconnections. The migration rate is the number of individuals exchanged during the migration. The migration interval is the number of generations between two consecutive calls of the operator, while the migration policy defines the exchanged individuals and their integration within the target subpopulations. The migration rate and migration interval are the two most important parameters, controlling the quantitative aspects of migration (Alba & Tomassini, 2002; Cantu-Paz, 1998). In the case where the genetic material, as well as the selection and recombination operators, is the same for all the individuals and all subpopulations, we call these algorithms *uniform* dEA. On the other hand, when different subpopulations evolve with different parameters and/or with different individual representations, the resulting algorithm is called *nonuniform* dEA (Alba, 2002; Tanese, 1987). For the rest of the chapter we focus on uniform dEAs.

Hierarchical parallel algorithms combine at least two different methods of EA parallelization to form a hybrid algorithm. At the higher level, there exists a multiple-population EA algorithm, while at the lower levels any kind of parallel EA implementation can be utilized.

To conclude, the use of parallel and distributed EA implementation has many advantages (Alba, 2002), such as:

1. finding alternative solutions to the same problem
2. parallel search from multiple points in the space
3. easy parallelization
4. more efficient search, even without parallel hardware
5. higher efficiency than sequential EAs
6. speedup due to the use of multiple CPUs

For more information regarding parallel EA implementations, software tools, and theory advances the interested reader could refer to the following review papers and books (Alba, 2002; Alba & Troya, 1999; Cantu-Paz, 1998, 2000; Zomaya, 1996).

HIGHER ORDER NEURAL NETWORKS AND PI-SIGMA NETWORKS

In this section we will briefly review the Higher order Neural Networks. HONNs expand the capabilities of standard Feedforward Neural Networks by including input nodes that provide the network with a more complete understanding of the input patterns and their relations. Basically, the inputs are transformed so that the network does not have to learn some basic mathematical functions, such as squares, cubes, or sines. The inclusion of these functions enhances the network's understanding of a given problem and has been shown to accelerate training on some applications. However, typically only second order networks are considered in practice.

Moreover, higher order terms in HONNs can increase the information capacity of neural networks in comparison to neural networks that utilize summation units only. The larger capacity means that the same function or problem can be solved by a network having fewer units. As a result, the representational power of higher order terms can help solving complex problems with construction of significantly smaller networks, while maintaining fast learning capabilities (Leerink et al., 1995).

Several neural network models have been developed to gain an advantage in using higher order terms (Gurney, 1992; Leerink et al., 1995; Redding et al., 1993). Examples of these higher order neural networks are: product unit neural networks (Ismail, 2000), Pi-Sigma network (Ghosh & Shin, 1992; Shin & Ghosh, 1991a,b), Sigma-Pi networks (Lee Giles, 1987), second-order neural networks (Milenkovic, 1996) and functional link neural networks (Ghosh et al., 1992; Hussain et al., 1997; Zurada, 1992). The main disadvantage of the majority of the HONNs is that the required number of weights increases exponentially with the dimensionality of the input patterns.

On the other hand, the PSNs were developed to maintain the fast learning property and powerful mapping capability of single layer HONNs, whilst avoiding the combinatorial explosion in the number of free parameters when the input dimension is increased. In contrast to a single layer HONNs, the number of free parameters in PSNs increases linearly to the order of the network. For that reason, PSNs can overcome the problem of weight explosion that occurs in single layer HONNs and rises exponentially to the number of inputs.

Specifically, PSN is a multilayer feedforward network that outputs products of sums of the input components. It consists of an input layer, a single "hidden" (or middle) layer of summing units, and an output layer of product units. The weights connecting the input neurons to the neurons of the middle layer are adapted during the learning process by the training algorithm, while those connecting the neurons of the middle layer to the product units of the output layer are fixed. For this reason the middle layer is not actually hidden and the training process is significantly simplified and accelerated (Ghosh & Shin, 1992; Shin & Ghosh, 1991a,b).

Let the input $x = (1, x_1, x_2, ..., x_N)^T$, be an *(N+1)*-dimensional vector, where 1 is the input of the bias unit and $x_k, k = 1,2,...,N$ denotes the *k*-th component of the input vector. Each neuron in the middle layer computes the sum of the products of each input with the corresponding weight. Thus, the output of the *j*-th neuron in the middle layer is given by the sum:

$$h_j = w_j^T x = \sum_{k=1}^{N} w_{kj} x_k + w_{0j},$$

where $j = 1,2,...,K$ and w_{oj} denotes a bias term. Output neurons compute the product of the aforementioned sums and apply an activation function on this product. An output neuron returns:

$$y = \sigma\left(\prod_{j=1}^{K} h_j\right),$$

Where $\sigma(\cdot)$ denotes the activation function. The number of neurons in the middle layer defines the order of the PSN. This type of networks are based on the idea that the input of a *K*-th order processing unit can be represented by a product of *K* linear combinations of the input components. Assuming that *(N+1)* weights are associated with each summing unit, there is a total of *(N+1)K* weights and biases for each output unit. If multiple outputs are required (for example, in a classification problem), an independent summing layer is required for each one. Thus, for an *M*-dimensional output vector *y*, a total of $\sum_{i=1}^{M}(N+1)K_i$ adjustable weight connections are needed, where K_i is the number of summing units for the *i*-th output. This allows great flexibility as the output layer indirectly incorporates some of the capabilities of HONNs utilizing a smaller number of weights and processing units. Furthermore, the network can be either regular or can be easily incrementally expandable, since the order of the network can be increased by adding another summing unit in the middle layer without disturbing the already established connections.

A further advantage of PSNs is that we do not have to pre-compute the higher order terms and incorporate them into the network, as is necessary for a single layer HONN. PSNs are able to learn in a stable manner even with fairly large learning rates (Ghosh & Shin, 1992; Shin & Ghosh, 1991a,b). The use of linear summing units makes the convergence analysis of the learning rules for PSN more accurate and tractable. The price to be paid is that the PSNs are not universal approximators. Despite that, PSNs demonstrated competent ability to solve many scientific and engineering problems, such image compression (Hussain & Liatsis, 2002), pattern recognition (Shin et al., 1992), and financial time series prediction (Ming Zhang, 2006). Figure 2, exhibits the architecture of a *K*-th order Pi-Sigma network having *N* input components and *M* output product units.

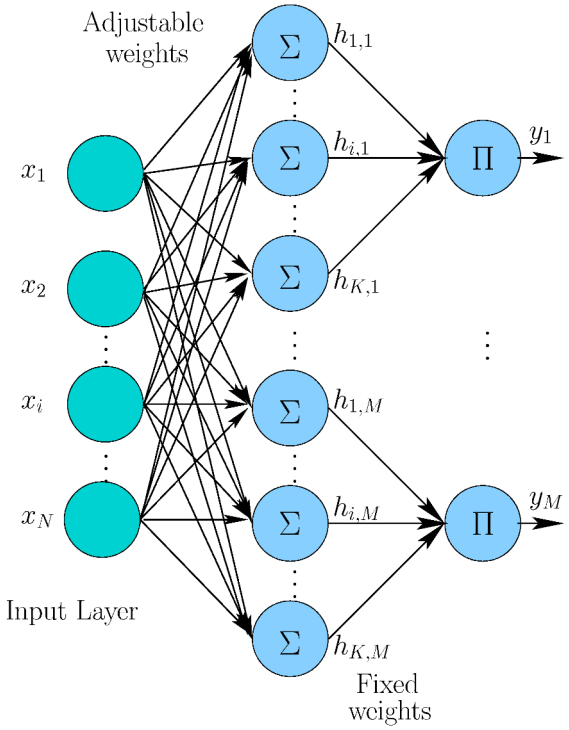
The Training Algorithm of the PSNs

The PSNs can be trained using the gradient descent learning algorithm (in the neural network literature also known as the BackPropagation learning algorithm) on the estimated sum of the squared error. Thus, the training process can be realized by minimizing the error function E, defined as:

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{j=1}^{M}\left(y_{j,p} - t_{j,p}\right)^2,$$

where *p* is an index over the training input-output pairs, *P* is the number of these pairs, *M* is the number of neurons in the output layer and $\left(y_{j,p} - t_{j,p}\right)^2$ is the squared difference between the output $y_{j,p}$ of the *j*-th neuron in the output layer, for pattern *p*, and the corresponding target value $t_{j,p}$. Each pass through the entire training set is called an epoch. Notice that, the error function is the objective (fitness) function.

Figure 2. Pi-Sigma network with multiple outputs



The weights of a PSN are updated according to the following equation:

$$\Delta w_{ij} = \eta(t_{j,p} - y_{j,p})\sigma'(\prod_{L=1}^{K} h_L)\prod_{L\neq i} h_L x_j,$$

where η is the learning rate, σ' is the derivative of the activation function, h_L is the output of the L-th neuron in the middle layer, and x_j is the j-th component of the current input pattern. Appropriate learning rate values help to avoid convergence to a saddle point or a maximum. In practice, a small constant learning rate is chosen ($0 < \eta < 1$) to secure the convergence of the BackPropagation training algorithm and avoid oscillations in a steep direction of the error surface. However, it is well known that this approach tends to be inefficient. The interested reader may refer to (Ghosh & Shin, 1992; Shin & Ghosh, 1991a,b).

Although FNNs and HONNs can be simulated in software, hardware implementation is sometimes required in real life applications, where high speed of execution is necessary. Thus, the natural implementation of FNNs or HONNs (because of their modularity) is a distributed (or parallel) one (Plagianakos & Vrahatis, 2002). In the next section we present the distributed EAs used in this study.

NEURAL NETWORK TRAINING USING DISTRIBUTED EVOLUTIONARY ALGORITHMS

For completeness purposes let us briefly present the Distributed Differential Evolution and Particle Swarm Optimization algorithms for higher order neural network training. Our distributed implementations are based on the Message Passing Interface standard, which facilitates the execution of parallel applications.

The Distributed DE Algorithm

Differential Evolution (DE) is an optimization method, capable of handling non-differentiable, discontinuous and multimodal objective functions. The method requires few, easily chosen, control parameters. Extensive experimental results have shown that DE has good convergence properties and in many cases outperforms other well known evolutionary algorithms. The original DE algorithm, as well as its distributed implementation, has been successfully applied to FNN training (Magoulas et al., 2004; Plagianakos & Vrahatis, 1999, 2002). Distributed Differential Evolution (DDE) for Pi-Sigma networks training is presented here.

The modified DDE maintains a population of potential integer solutions, *individuals*, to probe the search space. The population of individuals is randomly initialized in the optimization domain. At each iteration, called *generation*, new individuals are generated through the combination of randomly chosen individuals of the current population. Starting with a population of *NP* integer weight vectors, $w_g^i, i = 1,2,...,NP,$ where *g* denotes the current generation, each weight vector undergoes mutation to yield a mutant vector, u_{g+1}^i. The mutant vector that is considered here (for alternatives see (Chakraborty, 2008; Price et al., 2005; Storn & Price, 1997)), is obtained through one of the following equations:

$$u_{g+1}^i = w_g^{\text{best}} + F \cdot \left(w_g^{r_1} - w_g^{r_2}\right), \tag{1}$$

$$u_{g+1}^i = w_g^{r_1} + F \cdot \left(w_g^{r_2} - w_g^{r_3}\right), \tag{2}$$

where w_g^{best} denotes the best member of the current generation and $F > 0$ is a real parameter, called *mutation constant* that controls the amplification of the difference between the two weight vectors. Moreover, $r_1, r_2, r_3 \in \left\{1,2,...,i-1,i+1,...,NP\right\}$ are random integers mutually different and different from the running index *i*. Obviously, the mutation operator results in a real weight vector. As our aim is to maintain an integer weight population at each generation, each component of the mutant weight vector is rounded to the nearest integer. Additionally, if the mutant vector is not in the hypercube [-32, 32]^D, we calculate u_{g+1}^i using the following formula:

$$u_{g+1}^i = \text{sign}\left(u_{g+1}^i\right) \times \left(\left|u_{g+1}^i\right| \bmod 32\right), \tag{3}$$

where sign is the well known three valued signum function. During recombination, for each component *j* of the integer mutant vector, u_{g+1}^i, a random real number, *r*, in the interval [0,1] is obtained and compared

with the *crossover constant*, *CR*. If $r \leq CR$ we select as the *j*-th component of the trial vector, v_{g+1}^{i}, the corresponding component of the mutant vector, u_{g+1}^{i}. Otherwise, we choose the *j*-th component of the target vector, w_{g}^{i}. It must be noted that the result of this operation is also a 6-bit integer vector. Next, we briefly describe the distributed PSO algorithm utilized in this study.

The Distributed PSO Algorithm

The Particle Swarm Optimization (PSO) algorithm is an Evolutionary Computation technique, which belongs to the general category of Swarm Intelligence methods. It was introduced by Eberhart and Kennedy (1995), PSO is inspired by the social behavior of bird flocking and fish schooling, and is based on a social-psychological model of social influence and social learning. The fundamental hypothesis to the development of PSO is that an evolutionary advantage is gained through the social sharing of information among members of the same species. Furthermore, the behavior of the individuals of a flock corresponds to fundamental rules, such as nearest-neighbor velocity matching and acceleration by distance (Eberhart et al., 1996; Kennedy & Eberhart, 1995). Like DE, PSO is capable of handling non-differentiable, discontinuous and multimodal objective functions and has shown great promise in several real-world applications (Clerc, 2006; Parsopoulos & Vrahatis, 2004, 2002).

More specifically, PSO is a population-based stochastic algorithm that exploits a population of individuals, to effectively probe promising regions of the search space. Thus, each individual (particle) of the population (swarm) moves with an adaptable velocity within the search space and retains in its memory the best position it ever encountered. There are two variants of PSO, namely the *global* and the *local*. In the *global* variant, the best position ever attained by all individuals of the swarm is communicated to all the particles, while in the *local* variant, for each particle it is assigned a neighborhood consisting of a pre-specified number of particles and the best position ever attained by the particles in their neighborhood is communicated among them (Eberhart et al., 1996).

More specifically, each particle is a *D*-dimensional integer vector, and the swarm consists of *NP* individuals (particles). Thus, the position the *i*-th particle of the swarm can be represented as: $X_i = \left(x_{i1}, x_{i2}, ..., x_{iD}\right)$. The velocity of each particle is also an *D*-dimensional vector, and for the *i*-th particle is denoted as: $V_i = \left(u_{i1}, u_{i2}, ..., u_{iD}\right)$. The best previous position of the *i*-th particle can be recorded as: $P_i = \left(p_{i1}, p_{i2}, ..., p_{iD}\right)$, and the best particle in the swarm, the particle with the smallest fitness function value, is indicated by the index *g*. Furthermore, the neighborhood of each particle is usually defined through the particles' indices. The most common topology is the *ring* topology, where the neighborhood of each particle consists of particles with neighboring indices (Mendes et al., 2004).

Clerc and Kennedy (2002), proposed a version of PSO which incorporates a new parameter χ, known as the *constriction factor*. The main role of the constriction factor is to control the magnitude of the velocities and alleviate the ``swarm explosion'' effect that prevented the convergence of the original PSO algorithm (Angeline, 1998). According to Clerc & Kennedy, (2002), the dynamic behavior of the particles in the swarm is manipulated using the following equations:

$$V_i(t+1) = \chi\left(V_i(t) + c_1 r_1\left(P_i(t) - X_1(t)\right) + c_2 r_2\left(P_g(t) - X_1(t)\right)\right), \tag{4}$$

$$X_i(t+1) = X_i(t) + V_i(t+1), \tag{5}$$

where $i = 1,2,...,NP$, c_1 and c_2 are positive constants referred to as *cognitive* and *social* parameters respectively, and r_1 and r_2 are randomly chosen numbers uniformly distributed in [0,1].

In a stability analysis provided in Clerc & Kennedy, (2002) it was implied that the constriction factor is typically calculated according to the formula:

$$\chi = \frac{2k}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|}, \tag{6}$$

for $\phi > 4$, where $\phi = c_1 + c_2$, and $k = 1$.

The resulting position of the *i*-th particle $\left(X_i(t+1)\right)$ is a real weight vector. To this end, similarly to the DDE implementation, we round $\left(X_i(t+1)\right)$ to the nearest integer and subsequently utilize equation (3) to constrain it in the range [-32, 32]. Next, we briefly describe the operator controlling the migration of the best individuals.

The Migration Operator

The distributed versions of the DE and PSO algorithms have been employed according to the dEA paradigm. To this end, each processor is assigned a subpopulation of potential solutions. The subpopulations are independently evolved in parallel and occasional migration is employed to allow cooperation between them. The migration of the best individuals is controlled by the migration constant ϕ. A good choice for the migration constant is one that allows each subpopulation to evolve independently for some iterations, before the migration phase actually occur. There is a critical migration constant value below which the DDE and DPSO performance is hindered by the isolation of the subpopulations, and above which the subpopulations are able to locate solutions of the same quality as the panmictic implementations. Detailed description of the DDE algorithm and experimental results on difficult optimization problems can be found in (Plagianakos & Vrahatis, 2002; Tasoulis et al., 2004). A parallel implementation of the PSO algorithm can be found in (Nedjah et al., 2006).

Tools for Implementing a Parallel Evolutionary Algorithm

There are several communications tools or libraries for implementing a Parallel Evolutionary Algorithm. The majority of these toolkits are based on the message passing model of communication, since it allows to take advantage of the current hardware technology i.e. multi core computers having shared or distributed memory. In the message passing model, processes in the same or on physically different processors communicate by sending messages to each other through a communication medium, such as a standard or specialized network connection.

Among the most well-known and widely used tools for implementing parallel or distributed computations are: Sockets (Comer & Stevens, 1996), OpenMP[1], the Parallel Virtual Machine (PVM) (Sunderam, 1990), the Message Passing Interface (MPI) (MPI Forum, 1994), Java Remote Method Invocation (Sun), and the Globus toolkit for grid computing (Foster & Kesselman, 1997). Here, the implementation of the parallel Evolutionary Algorithms is based on the Message Passing Interface, which is briefly described in the next section.

The Message Passing Interface

The Message Passing Interface (MPI) is a portable message-passing standard that facilitates the development of parallel applications and libraries. MPI is the specification resulting from the MPI-Forum (1994), which involved several organizations designing a portable system which can allow programs to work on a heterogeneous network. MPI implementations for executing parallel applications run on both tightly-coupled massively-parallel machines and on networks of distributed workstations with separate memory. With this system, each executing process will communicate and share its data with others by sending and receiving messages. The MPI functions support process-to-process communication, group communication, setting up and managing communication groups, and interacting with the environment. Thus, MPI can be incorporated for dEA and/or cEA implementation.

A large number of MPI implementations are currently available, each of which emphasizes different aspects of high-performance computing or is intended to solve a specific research problem. In this study, the OpenMPI implementation of the MPI standard has been utilized. OpenMPI is open source, peer-reviewed, production-quality complete MPI implementation, which provides extremely high performance (Gabriel et al., 2004).

EXPERIMENTAL RESULTS

In this study, the sequential, as well as the distributed versions of the DE and PSO algorithms are applied to train PSNs with integer weights and threshold activation functions. Here, we report results from the following well known and widely used neural network training problems:

1. *N*-bit Parity check problems (Hohil et al., 1999; Rumelhart et al., 1987)
2. the Iris classification problem (Iris) (Asuncion & Newman, 2007)
3. the numeric font classification problem (NumFont) (Magoulas et al., 1997)
4. the MONK's classification problems (MONK1, MONK2, and MONK3) (Thrun et al., 1991)
5. the handwritten digits classification problem (PenDigits) (Asuncion & Newman, 2007)
6. the rock vs. mine sonar problem (Sonar) (Gorman & Sejnowski, 1988)
7. the Breast Cancer Wisconsin Diagnostic problem (BCWD) (Asuncion & Newman, 2007)

For all the training problems, we have used the fixed values of $F = 0.5$ and $CR = 0.7$ as the DE mutation and crossover constants respectively. Similarly, for the PSO algorithm, fixed values for the cognitive and social parameters $c_1 = c_2 = 2.05$ have been used, and the constriction factor $\chi = 0.729$ has been calculated using Eq. (6).

Regarding the number of hidden neurons, we tried to minimize the degrees of freedom of the PSN. Thus, the simpler network topology, which is capable to solve each problem, has been chosen. Below we exhibit the experimental results from the sequential and the distributed DE and PSO implementations. For all the experiments reported below, unless clearly stated, we utilize threshold activation functions and 6-bit integer weights.

Table 1. Simulation results (epochs) for the N-bit parity check problem

			Generations			
N	Topology	Algorithm	*Min*	*Mean*	*Max*	*St.D.*
2	2-2-1	DE_1	1	1.70	5	1.36
2	2-2-1	DE_2	1	5.04	12	4.92
2	2-2-1	PSO_1	1	1.92	10	1.26
2	2-2-1	PSO_2	1	2.07	13	1.71
3	3-3-1	DE_1	1	13.93	50	10.16
3	3-3-1	DE_2	1	17.98	77	13.95
3	3-3-1	PSO_1	1	23.21	177	21.91
3	3-3-1	PSO_2	1	29.06	281	35.28
4	4-4-1	DE_1	1	9.09	47	8.29
4	4-4-1	DE_2	1	9.66	34	8.55
4	4-4-1	PSO_1	1	2.02	10	1.42
4	4-4-1	PSO_2	1	2.20	17	1.74
5	5-5-1	DE_1	1	36.14	100	21.21
5	5-5-1	DE_2	1	35.98	100	21.76
5	5-5-1	PSO_1	1	27.01	200	28.51
5	5-5-1	PSO_2	1	28.53	210	29.74

Sequential DE and PSO Implementation

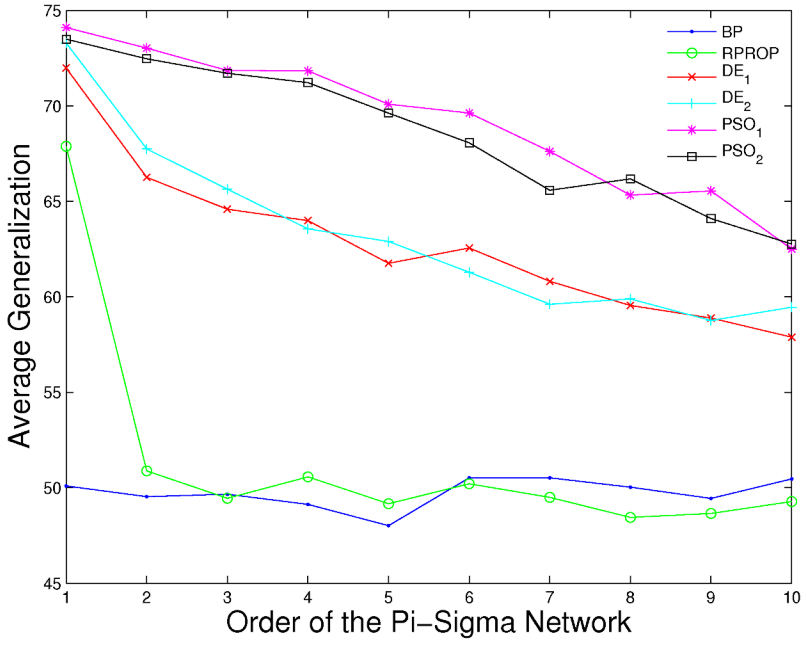
Here, we exhibit experimental results from the sequential DE and PSO algorithms. We call DE_1 and DE_2 the DE algorithms that use the mutation operators defined in equations (1) and (2), respectively. We call PSO_1 and PSO_2 the local and the global PSO variant, respectively. The neighborhood of each particle had a radius of one, i.e. the neighborhood of the *i*-th particle consists of the *(i-1)*-th and the *(i+1)*-th particles. Notice that the software used in this section does not contain calls to the MPI library. To this end, this implementation is marginally faster than the distributed implementation executed in just one computer node.

The first set of experiments consists of the *N*-bit parity check problems. These problems are well known and widely used benchmarks and are suitable for testing the non-linear mapping and "memorization" capabilities of neural networks. Although these problems are easily defined they are hard to solve, because of their sensitivity to initial weights and their multitude of local minima. Each *N*-bit problem has 2^N patterns with *N* attributes in each pattern. All patterns have been used for training and testing. For each *N*-bit problem we have used an *N* degree Pi-Sigma network (resulting *N* neurons in the middle layer). Here, we report results for *N*=2, 3, 4, and 5.

For each problem and each algorithm, we have used 10 individuals in each population and have conducted 1000 independent simulations. The termination criterion applied to the learning algorithm was the mean square training error (*MSE*) and it was different for each *N*-bit parity problem (0.05, 0.025, 0.125, and 0.125, respectively), following the experimental setup of (Ghosh & Shin, 1992). Notice that the PSNs trained here have threshold activation functions.

Table 1 shows the experimental results for the parity check problems. The reported parameters for the simulations that have reached solution are: *Min* the minimum number, *Mean* the mean value, *Max*

Figure 3. Average generalization results for training several PSNs with different order by BP, RProp, DE_1, DE_2, PSO_1 and PSO_2 algorithms (Sonar problem)



the maximum number, and *St.D.* the standard deviation of the number of training generations. All trained networks gave perfect classification capabilities for all problems. The results of PSNs having threshold activation functions reported below are equivalent or better than the results of PSNs trained using the classical back-propagation algorithm (Ghosh & Shin, 1992). An additional advantage of the proposed approach is that no gradient information is required, thus no backward passes were performed.

Furthermore, for comparison reasons with the classical Pi-Sigma training methods, 100 PSNs of various architectures were trained for the Sonar problem (see description below), using the classical BackPropgation (BP) method, the RProp method (Riedmiller & Braun, 1994), and the DE_1, DE_2, PSO_1

Table 2. Generalization results for the Iris classification problem

Network Topology	Mutation Strategy	Generalization (%)			
		Min	*Mean*	*Max*	*St.D.*
4-2-1	DE_1	50	93.5	100	6.639
4-2-1	DE_2	80	95.4	100	3.115
4-2-1	PSO_1	78	95.1	100	3.163
4-2-1	PSO_2	82	95.3	100	3.013
4-3-1	DE_1	80	94.9	100	3.115
4-3-1	DE_2	80	95.1	100	3.224
4-3-1	PSO_1	82	95.1	100	3.067
4-3-1	PSO_2	80	95.2	100	3.062

Table 3. Generalization results for the NumFont problem

Network Topology	Mutation Strategy	Generalization (%)			
		Min	*Mean*	*Max*	*St.D.*
64-1-1	DE_1	80	99.4	100	2.50
64-1-1	DE_2	100	100	100	0.00
64-1-1	PSO_1	80	95.9	100	5.70
64-1-1	PSO_2	90	99.8	100	1.21

and PSO_2 algorithms. For fair comparison, real value weights and logistic activation functions have been used. All network weights have been initialized in the range [-32, 32] with real values. The learning rate for the BP algorithm has been set to 0.01, while default parameters values have been employed for the RPROP algorithm (Riedmiller & Braun, 1994). Additionally, the termination criterion applied to all learning algorithms was the mean square training error to reach the fixed value of 0.05. The average classification accuracy of the trained PSNs is illustrated in Figure 3.

Firstly, we must note that the generalization capabilities of the algorithms decrease as the dimension of the weight space increases. It is evident that both BP and RPROP have not achieved satisfactory generalization. On the other hand, the evolutionary approaches have obtained good generalization results for all training networks and it was always superior comparing against the BP and RProp algorithms. Additionally, the BP and the RProp algorithms occasionally converged to the predefined error goal and most of the times they converged to an undesired local minimum. As a result the trained PSNs exhibited bad classification accuracy.

The performance for all versions of DE and PSO algorithms was more reliable, stable and in the examined cases, always converged to the desired training error goal. The PSO_1 and PSO_2 trained PSNs exhibited classification accuracy comparable to the accuracy of DE_1, and DE_2 trained PSNs for the first architecture, while for the remaining architectures PSO trained PSNs were clearly better. The incorporation of global

Table 4. Generalization results for the MONK's problems

			Generalization (%)			
Problem	Topology	Algorithm	*Min*	*Mean*	*Max*	*St.D.*
MONK1	17-2-1	DE_1	86	96.68	100	2.43
MONK1	17-2-1	DE_2	86	96.74	100	2.38
MONK1	17-2-1	PSO_1	80	95.16	100	3.30
MONK1	17-2-1	PSO_2	83	96.02	100	2.66
MONK2	17-2-1	DE_1	79	97.36	100	2.38
MONK2	17-2-1	DE_2	91	97.66	100	1.45
MONK2	17-2-1	PSO_1	90	96.86	100	1.69
MONK2	17-2-1	PSO_2	91	97.31	100	1.64
MONK3	17-2-1	DE_1	82	91.57	97	2.37
MONK3	17-2-1	DE_2	81	90.77	97	3.10
MONK3	17-2-1	PSO_1	80	92.02	99	2.97
MONK3	17-2-1	PSO_2	81	93.14	99	2.46

Table 5. Generalization results for the PenDigits problem

Network	Mutation	Generalization (%)			
Topology	Strategy	*Min*	*Mean*	*Max*	*St.D.*
16-2-1	DE_1	83.91	86.20	88.74	1.08
16-2-1	DE_2	81.53	84.60	87.71	1.16
16-2-1	PSO_1	82.38	84.76	87.19	1.20
16-2-1	PSO_2	82.59	85.16	87.70	1.17

optimization methods (such as EAs) instead of classical local optimization methods (such as BP and RProp) is recommended. Global optimization methods incorporate efficient and effective searching mechanisms that avoid the convergence to local minima and thus enhance the neural network training procedure, as well as the classification accuracy of the trained networks.

Below we report experimental results from the sequential DE and PSO implementations on (a) the numeric font, (b) the MONK's, (c) the handwritten digits, (d) the rock vs. mine sonar, and (e) the Breast Cancer Wisconsin Diagnostic classification problems. To present the generalization results the following notation is used in the following Tables: *Min* indicates the minimum generalization capability of the trained PSNs; *Max* is the maximum generalization capability; *Mean* is the average generalization capability; and *St.D.* is the standard deviation of the generalization capability. In all cases, average performance presented was validated using the well known test for statistical hypotheses, named *t-test* (see for example (Law & Kelton, 2000)), using the SPSS 15 statistical software package.

It must be noted that PSNs trained for the MONK1, MONK2, MONK3, the Sonar, and the Breast Wisconsin Cancer Diagnostic training problems have only one output unit, since all the samples of those datasets belong to one of the two available classes. On the other hand, the networks trained for the Num-Font and the PenDigits classification problems have ten output units (one for each digit), while for the Iris classification problem the network has three output units. To implement a PSN having multiple output units is equivalent to constructing PSNs having common input units and different middle layer units (thus, different sets of weights), each having one output unit. Thus, a PSN should be trained to discriminate samples from each problem class.

The Iris Classification Problem

The Iris classification problem is perhaps the best known problem to be found in the pattern recognition literature. The data set contains three classes of 50 instances each, where each class refers to a type of an

Table 6. Generalization results for the Sonar problem

Network	Mutation	Generalization (%)			
Topology	Strategy	*Min*	*Mean*	*Max*	*St.D.*
60-1-1	DE_1	58	73.81	87	4.24
60-1-1	DE_2	57	73.35	87	4.34
60-1-1	PSO_1	64	73.89	85	3.92
60-1-1	PSO_2	61	74.44	90	3.85

Table 7. Generalization results for the Breast Cancer Wisconsin Diagnostic problem

Network	**Mutation**	**Generalization (%)**			
Topology	Strategy	*Min*	*Mean*	*Max*	*St.D.*
30-1-1	DE_1	91	95.8	100	1.293
30-1-1	DE_2	90	95.7	100	1.272
30-1-1	PSO_1	91	95.6	99	1.250
30-1-1	PSO_2	91	95.7	99	1.289

iris plant. One class is linearly separable from the other two, while the latter are not linearly separable from each other.

For the Iris classification problem the aim is to train a PSN to predict the class of an iris plant (Asuncion & Newman, 2007). We trained three distinct PSNs, one for each iris plant class, each one having 4 input units and one output unit. We have conducted 1000 independent simulations for each network. The termination criterion applied to the learning algorithm was either a training error less than 0.001 or 2000 iterations. After being trained, each PSN was tested for its average generalization performance, using the *max* rule. The experimental results are presented in Table 2. The training procedure was very

Table 8. Generalization results for the MONK1 and MONK2 benchmark problems

Computer	**Mutation**	**MONK1 Generalization (%)**				**MONK2 Generalization (%)**			
Nodes	Strategy	*Min*	*Mean*	*Max*	*St.D.*	*Min*	*Mean*	*Max*	*St.D.*
1	DDE_1	90	97.26	100	2.18	93	98.00	100	1.42
1	DDE_2	89	97.44	100	2.06	94	98.12	100	1.39
1	$DPSO_1$	85	96.16	100	2.54	91	97.14	100	1.62
1	$DPSO_2$	91	97.59	100	1.71	93	98.19	100	1.23
2	DDE_1	90	97.81	100	2.18	94	97.88	100	1.36
2	DDE_2	89	97.84	100	1.73	93	97.74	100	1.56
2	$DPSO_1$	88	96.75	100	2.47	93	97.59	100	1.50
2	$DPSO_2$	90	97.59	100	1.92	96	98.35	100	1.19
4	DDE_1	93	98.13	100	1.79	93	98.12	100	1.14
4	DDE_2	91	97.90	100	1.93	93	98.00	100	1.46
4	$DPSO_1$	93	97.27	100	1.88	93	97.90	100	1.46
4	$DPSO_2$	93	97.87	100	1.49	95	98.21	100	1.12
6	DDE_1	91	97.86	100	1.91	94	98.12	100	1.25
6	DDE_2	92	97.73	100	1.85	94	97.79	100	1.28
6	$DPSO_1$	92	96.78	100	1.69	93	97.61	100	1.49
6	$DPSO_2$	92	97.80	100	1.68	95	98.18	100	1.22
8	DDE_1	92	98.22	100	1.59	95	97.96	100	1.33
8	DDE_2	93	97.77	100	1.59	95	98.24	100	1.15
8	$DPSO_1$	90	97.05	100	2.03	92	97.48	100	1.71
8	$DPSO_2$	94	97.91	100	1.26	95	98.19	100	1.08

fast and robust, while all trained PSNs exhibited high generalization accuracy for all classes of the Iris classification problem.

The Numeric Font Classification Problem

For the numeric font classification problem the aim is to train a PSN to recognize 8x8 pixel machine printed numerals from zero to nine in standard Helvetica font. After being trained, the PSN was tested for its generalization capability using Helvetica italic font (Magoulas et al., 1997). Note that the test patterns in the italic font have 6 to 14 bits reversed from the training patterns. To evaluate the average generalization performance the *max* rule was employed.

For the NumFont problem we trained 10 distinct PSNs, each one having 16 input units and one output unit. Thus, one PSN for each digit has been trained and we have conducted 1000 independent simulations for each network. The termination criterion applied to the learning algorithm was either a training error less than 0.001 or 1000 iterations. The experimental results are presented in Table 3. All algorithms exhibited high generalization accuracy. DE_1 in particular achieved 100% generalization success, followed closely by PSO_2. This indicates that the global variants exhibited better results for this problem.

Table 9. Generalization results for the MONK3 and SONAR benchmark problems

Computer	**Mutation**	**MONK3 Generalization (%)**				**SONAR Generalization (%)**			
Nodes	Strategy	*Min*	*Mean*	*Max*	*St.D.*	*Min*	*Mean*	*Max*	*St.D.*
1	DDE_1	83	92.69	97	2.19	61	76.62	88	5.87
1	DDE_2	81	91.06	97	2.98	60	76.49	90	6.08
1	$DPSO_1$	84	92.16	97	2.79	62	76.37	91	5.81
1	$DPSO_2$	86	92.90	98	2.42	60	77.16	90	5.46
2	DDE_1	87	92.49	97	2.14	54	76.22	90	5.87
2	DDE_2	82	90.99	96	2.62	62	76.40	90	5.63
2	$DPSO_1$	82	91.55	96	2.61	64	76.97	88	5.25
2	$DPSO_2$	83	91.99	98	3.04	62	77.55	90	5.82
4	DDE_1	83	92.37	97	2.47	61	75.90	90	6.23
4	DDE_2	82	90.40	97	3.14	64	76.05	88	5.32
4	$DPSO_1$	83	90.91	97	3.00	58	76.77	91	6.42
4	$DPSO_2$	85	92.80	98	2.48	62	76.90	88	5.54
6	DDE_1	84	92.71	96	2.11	58	76.59	87	5.92
6	DDE_2	83	90.45	96	3.22	58	75.89	87	6.20
6	$DPSO_1$	84	91.27	96	2.85	58	76.47	90	6.04
6	$DPSO_2$	82	91.67	98	3.33	61	75.48	90	5.66
8	DDE_1	85	92.34	96	1.93	61	76.53	87	5.60
8	DDE_2	82	90.64	95	2.66	60	76.06	91	5.83
8	$DPSO_1$	84	90.94	96	2.79	61	77.00	90	5.83
8	$DPSO_2$	80	92.51	97	2.67	60	77.16	90	6.26

Table 10. Generalization results for the Breast Cancer Wisconsin Diagnostic benchmark problem

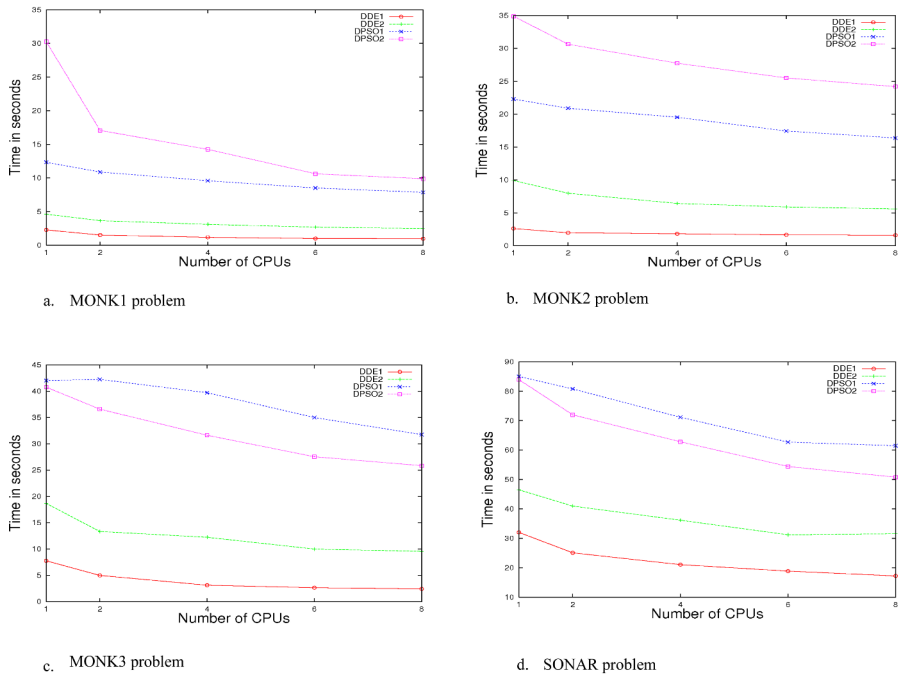
Computer	Mutation	BCWD Generalization (%)			
Nodes	Strategy	*Min*	*Mean*	*Max*	*St.D.*
1	DDE_1	92	95.43	98	1.38
1	DDE_2	91	95.54	98	1.23
1	$DPSO_1$	93	95.83	98	1.16
1	$DPSO_2$	92	95.86	99	1.26
2	DDE_1	92	95.32	98	1.27
2	DDE_2	91	95.47	98	1.38
2	$DPSO_1$	92	95.68	99	1.35
2	$DPSO_2$	92	95.76	99	1.29
4	DDE_1	90	95.35	98	1.44
4	DDE_2	91	95.52	98	1.29
4	$DPSO_1$	91	95.67	99	1.38
4	$DPSO_2$	92	95.66	99	1.31
6	DDE_1	92	95.50	98	1.28
6	DDE_2	92	95.53	98	1.20
6	$DPSO_1$	92	95.82	98	1.25
6	$DPSO_2$	92	95.65	99	1.40
8	DDE_1	92	95.35	98	1.33
8	DDE_2	91	95.46	98	1.41
8	$DPSO_1$	92	95.55	98	1.27
8	$DPSO_2$	91	95.64	99	1.40

The MONK's Classification Problems

The MONK's classification problems are three binary classification tasks, which have been used for comparing the generalization performance of learning algorithms (Thrun et al., 1991). These problems rely on the artificial robot domain, in which robots are described by six different attributes. Each one of the six attributes can have one of 3, 3, 2, 3, 4, and 2 values, respectively, which results in 432 possible combinations that constitute the total data set (see (Thrun et al., 1991), for details). Each possible value for every attribute is assigned a single bipolar input, resulting 17 inputs.

For the MONK's problems we have tested PSNs having two units in the middle layer (i.e. 17-2-1 PSN architecture). Table 4 illustrates the average generalization results (1000 runs were performed). The termination criterion applied to the learning algorithm was either a training error less than 0.01 or 5000 iterations. Once again the DE and PSO trained PSNs exhibited high classification success rates. Notice that it has been theoretically proved that PSNs are capable to learn perfectly any Boolean Conjunctive Normal Form (CNF) expression (Shin & Ghosh, 1991b) and that the MONK's problems can be described in CNF.

Figure 4. Average elapsed wall-clock times for training PSNs by DDE_1, DDE_2, $DPSO_1$ and $DPSO_2$, for the MONK's and the Sonar problems



a. MONK1 problem

b. MONK2 problem

c. MONK3 problem

d. SONAR problem

The Handwritten Digits Classification Problem

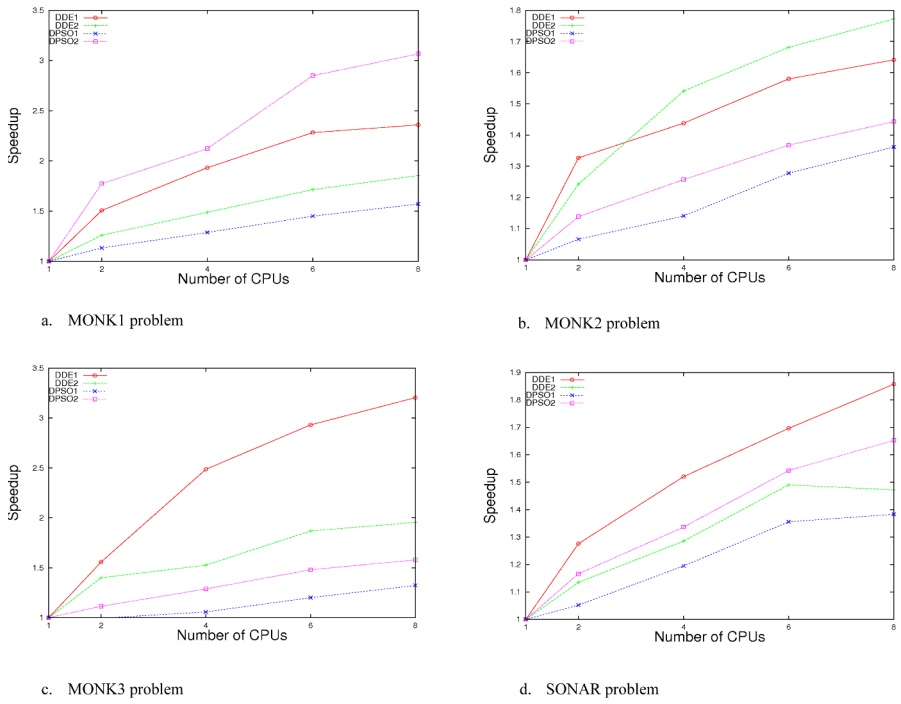
The PenDigits problem is part of the UCI Repository of Machine Learning Databases (Asuncion & Newman, 2007) and is characterized by a real-valued training set of approximately 7500 patterns. In this experiment, a digit database has been assembled by collecting 250 samples from 44 independent writers. The samples written by 30 writers are used for training, and the rest are used for testing. The training set consists of 7494 real valued samples and the test set of 3498 samples.

For the PenDigits problem we trained 10 different PSNs, one PSN for each digit. We have conducted 100 independent simulations for each network and the termination criterion applied to the learning algorithm was either a training error less than 0.001 or 1000 iterations. Table 5 exhibits the average generalization results. The average classification accuracy of the trained PSNs for the PenDigits problem is about 85% for all algorithms.

The Sonar Problem

For the Sonar problem the task is to train a PSN to discriminate between sonar signals bounced off a metal cylinder (mine) and those bounced off a roughly cylindrical rock. In this experiment the dataset contains 208 samples obtained by bouncing sonar signals off a metal cylinder and a rock at various angles and under various conditions (Gorman & Sejnowski, 1988). There exist 111 samples obtained from mines and 97 samples obtained from rocks. Each pattern consists of 60 real numbers in the range [0, 1]. Each number represents the energy within a particular frequency band, integrated over a certain period of time. The trained PSNs have one unit in the middle layer (i.e. 60-1-1 PSN architecture).

Figure 5. Speedup of training PSNs by DDE_1, DDE_2, $DPSO_1$ and $DPSO_2$, for the MONK's and the Sonar problems



a. MONK1 problem

b. MONK2 problem

c. MONK3 problem

d. SONAR problem

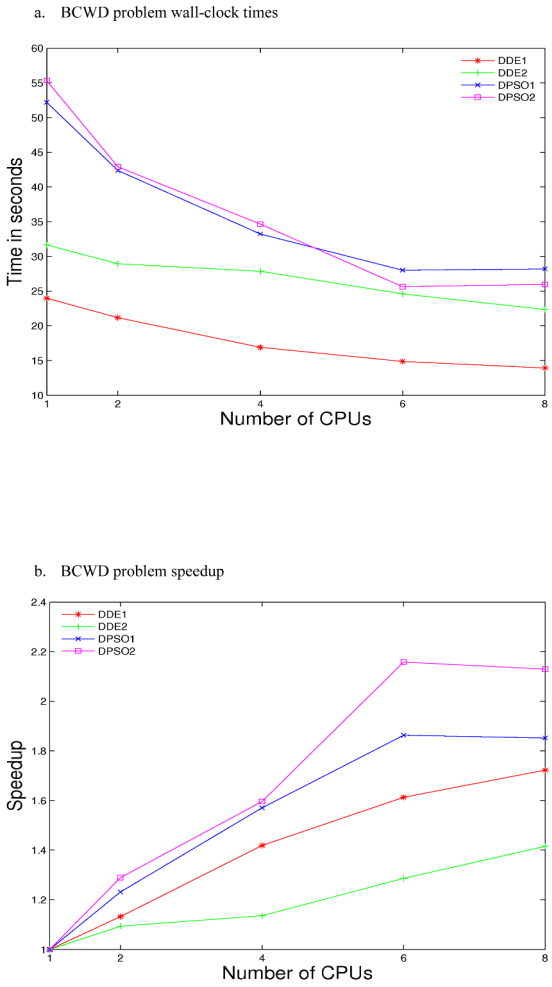
The classification accuracy of the trained PSNs is exhibited in Table 6. The average classification accuracy obtained by the EA trained PSNs is comparable to the classification accuracy of FNNs.

The Breast Cancer Wisconsin Diagnostic Problem

For the Breast Cancer Wisconsin Diagnostic (BCWD) problem the task is to train a PSN to classify breast images into two classes: malignant and benign. The dataset contains 569 samples. The features of each pattern were computed from a digitized image of a fine needle aspirate (FNA) of a breast mass and they describe characteristics of the cell nuclei present in the image (Asuncion & Newman, 2007). There exist 357 samples in the benign and 212 malignant class, each one consisting of 30 real numbers. In this set of simulations it has been used a trained PSNs with one unit in the middle layer (i.e. 30-1-1 PSN architecture), while the termination criterion applied to the learning algorithm was either a training error less than 0.005 or 2000 iterations.

The classification accuracy of the trained PSNs is exhibited in Table 7. Once again the learning process was robust, fast and reliable, and the DE and PSO trained PSNs exhibited high classification success rates.

Figure 6. Average elapsed wall-clock and speedup times for training PSNs by DDE_1, DDE_2, $DPSO_1$ and $DPSO_2$, for the Breast Cancer Wisconsin Diagnostic problem



Distributed DE and PSO Implementations

In this section the DDE and the DPSO algorithms are applied to train PSNs with integer weights and threshold activation functions. Here, we report results on the MONK's (Thrun et al., 1991), on the Sonar (Gorman & Sejnowski, 1988), as well as on the Breast Cancer Wisconsin Diagnostic (Asuncion & Newman, 2007) benchmark problems.

For this set of experiments, we have conducted 1000 independent simulations for each algorithm, using a distributed computation environment consisting of 1, 2, 4, 6, and 8 nodes. For the DDE and DPSO algorithms, we have used the same values for the algorithms' parameters. The migration constant was $\varphi = 0.1$. The termination criterion applied to the learning algorithm was either a training error less than 0.01 or 5000 iterations.

As for the choice of the communication topology, islands with many neighbors are more effective than sparsely connected ones. However, this brings forth a tradeoff between computation and communication

cost. The estimation of the optimal degree of the topology that minimizes the total computational cost is expensive and difficult. For the DDE and the DPSO implementation we have used the ring topology (each node communicates only with the next node on a ring).

In the distributed implementation, each processor evolves a subpopulation of potential solutions. To allow cooperation between the subpopulations, migration is employed. When a higher number of CPUs is utilized (i.e. higher number of subpopulations) the average generalization accuracy is slightly improved. This is probably due to the island model for the migration of the best individuals (Plagianakos & Vrahatis, 2002; Tasoulis et al., 2004).

The experimental generalization results of problems MONK1 and MONK2 are exhibited in Table 8, while the results of problems MONK3 and SONAR are presented in Table 9. Table 10 illustrates the generalization results for the Breast Cancer Wisconsin Diagnostic problem.

Overall, the results indicate that the training of PSNs with integer weights and thresholds, using the modified DDE and DPSO algorithms is efficient and promising. The learning process was robust, fast and reliable, and the performance of the distributed algorithms stable. Additionally, the trained PSNs utilizing DDE and DPSO exhibited good generalization capabilities.

The four methods considered here, exhibit similar performance. To better compare them, we have performed ANOVA tests and post hoc analysis (Tukey). For the 8 computer node case, the statistical results indicate that in the MONK problems the four methods exhibit different behavior, while they are equivalent in the SONAR and BCWD problems. More specifically, in the MONK1 problem the two PSO variants are equivalent, while in the MONK2 the global methods (i.e. DDE_2 and $DPSO_2$) are equivalent.

In addition to the generalization accuracy tests, we have also compared the four methods by means of the time needed to train the PSNs.

Distributed DE and PSO Algorithms: Time and Speedup Measurements

To better understand the efficiency of the proposed methods we have measured the time needed to converge to a solution. Figure 4 and Figure 6a illustrates average elapsed wall-clock times. For every experiment, the MPI timer (MPI Wtime) was used. This procedure is a high-resolution timer, which calculates the elapsed wall-clock time between two successive function calls. From the results, it is evident that the DDE algorithms are faster and trained the PSNs efficiently. Among the DDE algorithms DDE_1 is marginally better, while $DPSO_1$ and $DPSO_2$ seem equivalent.

Notice that DDE_1 mutation operator uses the best individual of the current generation for computation the mutant vector. On the other hand, DDE_2 computes the mutant vector from randomly chosen individuals. To this end, DDE_1 converges faster to a single minimum, while DDE_2 better explores the search space. Similarly, $DPSO_1$ is the local version of PSO, while $DPSO_2$ is the global version. Thus, to train a HONN for a new application, where the balance between exploration and exploitation is unknown, both local and global algorithms can be tried. Furthermore, one can start the training process using the DDE_2 or the $DPSO_2$ for better exploration and consequently switch to DDE_1 or $DPSO_1$ for faster convergence (Tasoulis et al., 2005). If only one algorithm must be utilized, in the case of an unknown problem, we recommend the use of the DDE_1 algorithm.

In addition to time measurements, we also calculated the speedup achieved by assigning each subpopulation to a different processor relative to assigning all subpopulations to a single processor. The speedup is illustrated in Figure 5 and in Figure 6b. In the literature various speedup measurement methods

have been proposed. However, to perform fair comparison between the sequential and the parallel (or distributed) code, the following conditions must be met (Alba, 2002; Punch, 1998):

1) average and not absolute times must be used,
2) the uni- and multi-processor implementations should be exactly the same, and
3) the parallel (or distributed) code must be run until a solution for the problem is found.

To obtain the plotted values, we conducted 1000 independent simulations for 1, 2, 4, 6, 8 computer nodes and the average speedup is shown. For every simulation the training error goal was met and the migration constant was equal to 0.1.

Several factors can influence the speedup, such as the local area network load and the CPU load due to system or other users' tasks (Alba, 2002; He & Yao, 2006; Hidalgo et al., 2007). Nevertheless, the speedup results indicate that the combined processing power overbalances the overhead due to process communication and speedup is achievable. It must be noted that the DDE_1 and $DPSO_2$ generally exhibit higher speedup results, with DDE_1 being the most efficiently parallelized algorithm. Overall, the best speedup was achieved by DDE_1 on the MONK3 problem, when 8 computer nodes were utilized (approximately 3.2 times faster than the simulation utilizing one computer node). Once again the use of DDE_1 is recommended for large distributed systems.

CONCLUSION

In this chapter, we study the Pi-Sigma Networks and propose the utilization of the sequential as well as the parallel (or distributed) Evolutionary Algorithms for PSN training. For the proposed distributed DE and PSO training algorithms each processor of a distributed computing environment is assigned a subpopulation of potential solutions. The subpopulations are independently evolved in parallel and occasional migration of the best individuals is employed to allow subpopulation cooperation. Furthermore, the trained PSNs have threshold activation functions and small integer weights. Thus, their hardware implementation is significantly simplified.

The performance of PSNs is evaluated through well known neural network training problems and the experimental results suggest that the proposed training approach using DE or PSO is robust, reliable, and efficient. By assigning each subpopulation to a different processor significant training speedup was achieved. The PSNs were able to effectively address several difficult classification tasks. Moreover, the EA trained PSNs exhibited good generalization capabilities, comparable with the best generalization capability of PSNs trained using other well-known batch training algorithms, such as the BP and the RProp (Epitropakis et al., 2006).

The EA-trained PSNs exhibited high classification accuracy. However the local variant of the DE algorithm (DDE_1) was clearly the fastest one. Thus, the use of DDE_1, in an unknown optimization task, is recommended.

In a future communication we intend to rigorously compare the classification capability of PSNs against other soft computing approaches. Additionally, we will give experimental results of PSNs trained using hierarchical parallel Evolutionary Algorithms.

ACKNOWLEDGMENT

This work was partially supported by an “Empirikion Foundation” award that helped the acquisition and the setup of the distributed computer cluster.

REFERENCES

Alba, E. (2002). Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters*, *82*(1), 7–13. doi:10.1016/S0020-0190(01)00281-2

Alba, E., & Tomassini, M. (2002). Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *6*(5), 443–462. doi:10.1109/TEVC.2002.800880

Alba, E., & Troya, J. M. (1999). A survey of parallel distributed genetic algorithms. *Complex.*, *4*(4), 31–52. doi:10.1002/(SICI)1099-0526(199903/04)4:4<31::AID-CPLX5>3.0.CO;2-4

Angeline, P. J. (1998). Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In *EP ‘98: Proceedings of the 7th International Conference on Evolutionary Programming VII*, (pp. 601-610). London, UK: Springer-Verlag.

Asuncion, A., & Newman, D. (2007). UCI machine learning repository.

Baeck, T., Fogel, D. B., & Michalewicz, Z. (Eds.). (1997). *Handbook of Evolutionary Computation*. Oxford, UK: Oxford University Press.

Branke, J. (1995). Evolutionary Algorithms for Neural Network Design and Training. In *Proceedings of the First Nordic Workshop on Genetic Algorithms and its Applications*, (pp. 145-163).

Cantù-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs Paralèlèles . Réseaux et Systèmes Répartis*, *10*(2), 141–171.

Cantù-Paz, E. (2000). *Efficient and Accurate Parallel Genetic Algorithms*. Norwell, MA: Kluwer Academic Publishers

Chakraborty, U. K. (2008). *Advances in Differential Evolution*. Springer Publishing.

Clerc, M. (2006). *Particle Swarm Optimization*. ISTE Publishing Company.

Clerc, M., & Kennedy, J. (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation . IEEE Transactions on*, *6*(1), 58–73.

Comer, D. E., & Stevens, D. L. (1996). *Internetworking with TCP/IP vol III (2nd ed.): client-server programming and applications BSD socket version*}. Upper Saddle River, NJ: Prentice-Hall.

Eberhart, R., Simpson, P., & Dobbins, R. (1996). *Computational intelligence PC tools*.San Diego, CA: Academic Press Professional

Eberhart, R. C., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proceedings of 6th Symposium on Micro Machine and Human Science*, (pp. 39-43). Nagoya, Japan.

Epitropakis, M. G., Plagianakos, V. P., & Vrahatis, M. N. (2006). Higher-order neural networks training using differential evolution. In *International Conference of Numerical Analysis and Applied Mathematics*, (pp. 376-379). Crete, Greece: Wiley-VCH.

Fogel, D. (1996). *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution*. West Sussex, Uk: Wiley.

Forum, M. P. I. (1994). MPI: A message-passing interface standard. (Technical Report UT-CS-94-230).

Foster, I., & Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications*, *11*, 115–128. doi:10.1177/109434209701100205

Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., et al. (2004). Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, (pp. 97-104). Budapest, Hungary.

Ghosh, J., & Shin, Y. (1992). Efficient higher-order neural networks for classification and function approximation. *International Journal of Neural Systems*, *3*, 323–350. doi:10.1142/S0129065792000255

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison Wesley

Gorges-Schleuter, M. (1991). Explicit parallelism of genetic algorithms through population structures. In *PPSN I: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, (pp 150-159). London: Springer-Verlag.

Gorman, R. P., & Sejnowski, T. J. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, *1*, 75–89. doi:10.1016/0893-6080(88)90023-8

Gurney, K. N. (1992). Training Nets of Hardware Realizable Sigma-Pi Units . *Neural Networks*, *5*(2), 289–303. doi:10.1016/S0893-6080(05)80027-9

Gustafson, S., & Burke, E. K. (2006). The speciating island model: an alternative parallel evolutionary algorithm. *Journal of Parallel and Distributed Computing*, *66*(8), 1025–1036. doi:10.1016/j.jpdc.2006.04.017

Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, *9*(2), 159–195. doi:10.1162/106365601750190398

He, J., & Yao, X. (2006). Analysis of scalable parallel evolutionary algorithms. In *CEC 2006: IEEE Congress on Evolutionary Computation*, (pp 120-127).

Hidalgo, J. I., Lanchares, J., de Vega, F. F., & Lombraina, D. (2007). Is the island model fault tolerant? In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, (pp. 2737-2744), New York: ACM.

Hohil, M. E., Liu, D., & Smith, S. H. (1999). Solving the n-bit parity problem using neural networks. *Neural Networks*, *12*(9), 1321–1323. doi:10.1016/S0893-6080(99)00069-6

Holland, J. H. (1975). *Adaptation in natural and artificial system*. Ann Arbor, MI:University of Michigan Press.

Huang, D. S., Ip, H. H. S., Law, K. C. K., & Chi, Z. (2005). Zeroing polynomials using modified constrained neural network approach. *IEEE Transactions on Neural Networks*, *16*(3), 721–732. doi:10.1109/TNN.2005.844912

Hussain, A. J., & Liatsis, P. (2002). Recurrent pi-sigma networks for DPCM image coding. *Neurocomputing*, *55*, 363–382. doi:10.1016/S0925-2312(02)00629-X

Hussain, A. J., Soraghan, J. J., & Durbani, T. S. (1997). A New Neural Network for Nonlinear Time-Series Modelling. *Neurovest Journal*, 16-26.

Ismail, A., & Engelbrecht, A. P. (2000). Global optimization algorithms for training product unit neuralnetworks. *In IJCNN 2000 . Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, *1*(2), 132–137.

Jagadeesan, A., Maxwell, G., & Macleod, C. (2005). Evolutionary algorithms for real-time artificial neural network training. *Lecture Notes in Computer Science*, *3697*, 73–78.

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings IEEE International Conference on Neural Networks, IV*, (pp 1942-1948). Piscataway, NJ: IEEE Service Center.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press

Law, A. M., & Kelton, W. D. (2000). *Simulation Modeling and Analysis*. New York: McGraw-Hill

Lee Giles, C. (1987). Learning, Invariance and Generalization in Higher-Order Neural Networks . *Applied Optics*, *26*(23), 4972–4978. doi:10.1364/AO.26.004972

Leerink, L. R., Giles, C. L., Horne, B. G., & Jabri, M. A. (1995). Learning with product units. In G. Tesaro, D. Touretzky, & T. Leen (Eds.), *Advances in Neural Information Processing Systems 7*, (pp 537-544). Cambridge, MA: MIT Press.

Magoulas, G. D., Plagianakos, V. P., & Vrahatis, M. N. (2004). Neural network-based colonoscopic diagnosis using on-line learning and differential evolution. *Applied Soft Computing*, *4*, 369–379. doi:10.1016/j.asoc.2004.01.005

Magoulas, G. D., Vrahatis, M. N., & Androulakis, G. S. (1997). Effective backpropagation training with variable stepsize. *Neural Networks*, *10*(1), 69–82. doi:10.1016/S0893-6080(96)00052-4

Mendes, R., Kennedy, J., & Neves, J. (2004). The fully informed particle swarm: simpler, maybe better. *Evolutionary Computation . IEEE Transactions on*, *8*(3), 204–210.

Milenkovic, S., Obradovic, Z., & Litovski, V. (1996). Annealing Based Dynamic Learning in Second-Order Neural Networks, (Technical Report), Department of Electronic Engineering, University of Nis, Yugoslavia.

Ming Zhang. (2008), *Artificial Higher Order Neural Networks for Economics and Business*, Christopher Newport University, USA.

Nedjah, N., Alba, E., & de Macedo Mourelle, L. (2006). *Parallel Evolutionary Computations (Studies in Computational Intelligence)*. New York, &Secaucus, NJ. Springer-Verlag

Parsopoulos, K. E., & Vrahatis, M. N. (2002). Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing: an international journal, 1*(2-3), 235-306.

Parsopoulos, K. E., & Vrahatis, M. N. (2004). On the computation of all global minimizers through particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, *8*(3), 211–224. doi:10.1109/TEVC.2004.826076

Perantonis, S., Ampazis, N., Varoufakis, S., & Antoniou, G. (1998). Constrained learning in neural networks: Application to stable factorization of 2-d polynomials. *Neural Processing Letters*, *7*(1), 5–14. doi:10.1023/A:1009655902122

Plagianakos, V. P., Magoulas, G. D., & Vrahatis, M. N. (2005). Evolutionary training of hardware realizable multilayer perceptrons. *Neural Computing & Applications*, *15*, 33–40. doi:10.1007/s00521-005-0005-y

Plagianakos, V. P., Sotiropoulos, D. G., & Vrahatis, M. N. (1998). Evolutionary algorithms for integer weight neural network training (Technical Report No.TR98-04). University of Patras,Greece.

Plagianakos, V. P., & Vrahatis, M. N. (1999). Neural network training with constrained integer weights. In Angeline, P., Michalewicz, Z., Schoenauer, M., Yao, X., & Zalzala, A., (eds.), *Congress of Evolutionary Computation (CEC'99)*, (pp 2007-2013). Washington DC: IEEE Press.

Plagianakos, V. P., & Vrahatis, M. N. (2002). Parallel evolutionary training algorithms for `hardware-friendly' neural networks. *Natural Computing*, *1*, 307–322. doi:10.1023/A:1016545907026

Price, K., Storn, R. M., & Lampinen, J. A. (2005). *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*}. New York: Springer-Verlag

Punch, W. (1998). How effective are multiple populations in genetic programming. In Koza, J. e. a., editor, *Proceedings of the Third Annual Conference on Genetic Programming* (pp 308-313). Madison, WI: Morgan Kaufmann.

Rechenberg, I. (1994). Evolution strategy. In Zurada, J., Marks II, R., & Robinson, C., (eds). *Computational Intelligence: Imitating Life*. Piscataway, NJ: IEEE Press.

Redding, N. J., Kowalczyk, A., & Downs, T. (1993). Constructive Higher-Order Network Algorithm that is Polynomial in Time . *Neural Networks*, *6*, 997–1010. doi:10.1016/S0893-6080(09)80009-9

Riedmiller, M., & Braun, H. (1994). RPROP -- *Description and Implementation Details* (Technical Report). Universitat Karlsruhe.

Rumelhart, D. E., McClelland, J. L., & the PDP Research Group, editors (1987). *Parallel Distributed Processing - Vol. 1*. Cambridge, MA: The MIT Press.

Shin, Y., & Ghosh, J. (1991a). The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. In *International Joint Conference on Neural Networks*.

Shin, Y., & Ghosh, J. (1991b). Realization of boolean functions using binary pi-sigma networks. In Dagli, C. H., Kumara, S. R. T., & Shin, Y. C., (eds.), *Intelligent Engineering Systems through Artificial Neural Networks*, (pp 205-210). New York: ASME Press.

Shin, Y., Ghosh, J., & Samani, D. (1992). Computationally efficient invariant pattern classification with higher-order pi-sigma networks. In Burke and Shin, (Eds.), *Intelligent engineering systems through artificial neural networks-II,* (379-384). New York: ASME Press.

Skolicki, Z. (2005). An analysis of island models in evolutionary computation. In *GECCO '05:* In *Proceedings of the 2005 workshops on Genetic and evolutionary computation*, (386-389). New York: ACM.

Skolicki, Z., & Jong, K. D. (2005). The influence of migration sizes and intervals on island models. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, (pp 1295-1302). New York: ACM.

Sprave, J. (1999). A unified model of non-panmictic population structures in evolutionary algorithms. In Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., & Zalzala, A., (eds.), *Proceedings of the Congress on Evolutionary Computation, 2*, (pp 1384-1391). IEEE Press.

Storn, R. (1999). System design by constraint adaptation and differential evolution. *IEEE Transactions on Evolutionary Computation*, *3*, 22–34. doi:10.1109/4235.752918

Storn, R., & Price, K. (1997). Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, *11*, 341–359. doi:10.1023/A:1008202821328

Sunderam, V. S. (1990). PVM: a framework for parallel distributed computing. *Concurrency: Pract. Exper.*, *2*(4), 315–339. doi:10.1002/cpe.4330020404

Tanese, R. (1987). Parallel genetic algorithms for a hypercube. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, (pp 177-183). Mahwah, NJ: Lawrence Erlbaum Associates.

Tasoulis, D. K., Pavlidis, N. G., Plagianakos, V. P., & Vrahatis, M. N. (2004). Parallel differential evolution. In *IEEE Congress on Evolutionary Computation (CEC 2004). 2*, (pp 2023-2029).

Tasoulis, D. K., Plagianakos, V. P., & Vrahatis, M. N. (2005). Clustering in evolutionary algorithms to efficiently compute simultaneously local and global minima. In *IEEE Congress on Evolutionary Computation (CEC 2005), 2*, (pp. 1847-1854).

Thrun, S. B. et.al. (1991). The MONK's problems: A performance comparison of different learning algorithms (Technical Report CS-91-197), Pittsburgh, PA.

Zomaya, A. Y. H. (1996). *Parallel and Distributed Computing Handbook.* New York: McGraw-Hill

Zurada, J. M. (1992). *Introduction to Artificial Neural Systems*. St. Paul MN:West Publishing Company.

ENDNOTE

[1] The OpenMP API specification for parallel programming: http://openmp.org/