Particle Swarm Optimization with Discrete Recombination: An Online Optimizer for Evolvable Hardware

Jorge Peña
Advanced Learning and Research Institute - ALaRI
Università della Svizzera Italiana - USI, Switzerland
jorge.pena@alari.ch

Andres Upegui, Eduardo Sanchez
Reconfigurable Digital Systems Group RDSG
Ecole Polytechnique Fédérale de Lausanne - EPFL, Switzerland
andres.upegui@epfl.ch, eduardo.sanchez@epfl.ch

Abstract

Self-reconfigurable adaptive systems have the possibility of adapting their own hardware configuration. This feature provides enhanced performance and flexibility, reflected in computational cost reductions. Self-reconfigurable adaptation requires powerful optimization algorithms in order to search in a space of possible hardware configurations. If such algorithms are to be implemented on chip, they must also be as simple as possible, so the best performance can be achieved with the less cost in terms of logic resources, convergence speed, and power consumption. This paper presents an hybrid bio-inspired optimization technique that introduces the concept of discrete recombination in a particle swarm optimizer, obtaining a simple and powerful algorithm, well suited for embedded applications. The proposed algorithm is validated using standard benchmark functions and used for training a neural network-based adaptive equalizer for communications systems.

1 Introduction

On-line and on-chip adaptation in self-reconfigurable hardware systems provide architectural flexibility, allowing the chip to adapt dynamically and autonomously to changes in its environment [16, 18]. A popular approach for building adaptive circuits is by means of bio-inspired techniques. Evolvable Hardware (EHW) tackles this problem by using Evolutionary Algorithms (EAs): inspired in the process of natural evolution, a population of circuits is incrementally improved through the application of genetic operators (selection, recombination, and mutation).

From an algorithmical point of view, EAs are nothing but *stochastic population-based* optimization techniques. A population-based optimization algorithm is characterized for keeping a population or set of solutions in memory. The algorithm produces populations of solutions sequentially from an initial population P_0 to a final population P_G, deriving the new population from the current one through the use of a *manipulation* function [7]. If, additionally, the manipulation function is non deterministic, the algorithm is said to be stochastic. In the case of EAs, the manipulation function consists in the merged application of the genetic operators. As they generally include randomness, the result is a stochastic manipulation function.

EAs have been largely used in adaptive hardware mainly because of the analogy between the genome representation in a genetic algorithm and the configuration bit-string in a reconfigurable logic device. However, different algorithms can also be used as long as they provide the necessary search capabilities.

Particle swarm optimizers constitute another group of *stochastic population-based* optimization algorithms. Particle Swarm Optimization (PSO) is a bio-inspired technique, founded on the social behavior of bird flocking and the idea of culture as an emergent process [6]. In PSO, a swarm or population of solutions "fly" through the search space according to certain stochastic velocity update rules, producing new sets of solutions in subsequents time steps. The population of solutions is thus "evolved" through the application of a certain non deterministic manipulation function, as EAs do.

Given the similarities between the two optimization approaches, it is natural to think of PSO as an alternative to EAs for carrying out adaptation in EHW. Some prelimi-

Proceedings of the First NASA/ESA Conference on Adaptive Hardware and Systems (AHS'06)
0-7695-2614-4/06 $20.00 © 2006 IEEE

nary steps have already been taken in this direction. PSO has been used, for instance, in the context of evolutionary circuit design [3, 7, 10] and in the problem of placement and routing in FPGAs [4, 17]. However, to the best of our knowledge, PSO has not already been used for *on-line on-chip* hardware adaptation or evolution.

When intending to implement *on-line* and *on-chip* hardware adaptation, one must consider the computational complexity of the involved search algorithms. This paper presents a simple, hybrid algorithm that takes the concept of recombination of EAs to incorporate it into the original scheme of PSO. The proposed algorithm is *hardware friendly*, being suitable for efficient implementation either in an embedded processor or dedicated hardware: it does not use multiplications and requires a minimal random number generator (RNG).

The new algorithm has been conceived targeting the adaptation of a society of agents embedded in a self-reconfigurable adaptive platform. Each of these agents can be, for instance, a channel equalizer in a communication system. Our hardware setup consists of a population of neural networks with *material existence* in an FPGA, being evaluated and adapted on-line by the proposed algorithm running in an embedded microprocessor. This contrasts with existing solutions to adaptive channel equalization, where a single equalizer is adapted through supervised learning [13] or a genetic algorithm [11, 12].

Our approach is bio-inspired at two levels: at the *computing engine* level and at the *adaptation mechanism* level. The computing engine constitutes the problem solver of the system. The problem at hand, being in this case the channel equalizer, is implemented as a population of *Binary Radial Basis Functions* artificial neural networks, to be described in section 4. The adaptation mechanism provides the possibility of modifying the function described by the computing engine. This adaptation is performed by the Particle Swarm Optimizer with Discrete Recombination algorithm described in section 3.

In this paper, we compare the new algorithm against the standard PSO in a rather theoretical experimental setting involving the minimization of four mathematical functions. Then, we present some preliminary results in the use of the method for evolving simple neural networks with binary activation functions for channel equalization.

2 Particle Swarm Optimization

In PSO, a n-dimensional search space is explored using a swarm of M particles, seeking to minimize an objective function f. The particles are connected according to a given topology. The neighborhood $N(j)$ of the j-th particle is defined as the set of particles connected to it. Two topologies have been traditionally used in the literature: the *lbest* topology and the *gbest* topology. In the *lbest* topology particles are organized in a circular array, the neighborhood of a particle comprising its adjacent neighbors with or without the particle itself. In the *gbest* topology, all the particles are connected together, so that the neighborhood of every particle is the whole swarm. The type of the topology defines the way information will be exchanged among the particles and the robustness of the algorithm [8].

Three kinds of information characterize each particle of the swarm in a given time step t: its position $\mathbf{x}_j^t$, its velocity $\mathbf{v}_j^t$and its *personal best* (*pbest*) $\mathbf{p}_j^t$, the best position it has found so long.

At time step $t+1$, each particle calculates its new velocity using a given *velocity update rule*. Traditionally, this update rule takes into account: (a) the particle's velocity at time step t, (b) its personal best, $\mathbf{p}_j^t$ and (c) its *neighborhood best*, the best position found so far by the particle's neighbors . The neighborhood best is defined for each particle j as

$$\mathbf{p}_{b(j)}^t = \underset{l \in N(j)}{\arg\min} \left(f(\mathbf{p}_l^t) \right). \tag{1}$$

In the case of a *gbest* topology, where the neighborhood of each particle is the population itself ($N(j) = \{1, 2, \ldots, M\}$), the neighborhood best is the same for all the particles. It is called *global best* (*gbest*), and is represented by $\mathbf{p}_g^t$.

The *inertia weight* update rule [15] modifies the particle's velocity according to:

$$\begin{aligned} \mathbf{v}_j^{t+1} = w \cdot \mathbf{v}_j^t + \mathbf{U}\left[0, \varphi_1\right] \cdot (\mathbf{p}_j^t - \mathbf{x}_j^t) \\ + \mathbf{U}\left[0, \varphi_2\right] \cdot (\mathbf{p}_{b(j)}^t - \mathbf{x}_j^t), \end{aligned} \tag{2}$$

where w is the inertia weight, $\mathbf{U}\left[lower, upper\right]$ is a vector of uniformly distributed random values between $lower$ and $upper$, and φ_1 and φ_2 are acceleration constants usually set to 2. The velocities are normally clamped by means of a damping function $\Gamma(\cdot)$, implemented component-wise as follows:

$$\Gamma\left(v_{ji}\right) = \begin{cases} v_{max} & \text{if } v_{ji} > v_{max} \\ -v_{max} & \text{if } v_{ji} < -v_{max} \\ v_{ji} & \text{otherwise} \end{cases} \tag{3}$$

After having calculated its new velocity applying Eq. 2 and 3, each particle updates its position by applying:

$$\mathbf{x}_j^{t+1} = \mathbf{x}_j^t + \mathbf{v}_j^{t+1}. \tag{4}$$

3 PSODR: Particle Swarm Optimization with Discrete Recombination

The standard PSO algorithm has a number of features that make it suitable for embedded applications. It is simple

enough to be implemented in software, or directly in hardware. Still, it requires 3 multiplications and the generation of $2B$ random bits per particle and per dimension, where B is the bit resolution of φ_1 and φ_2. Thus, a total amount of $3Mn$ multiplications and $2BMn$ randomly generated bits are required per iteration of the algorithm. For some critical applications, this can be prohibitive in terms of area, power consumption and/or performance.

PSODR was designed bearing in mind a complete avoidance of multiplications and a reduction of the number of randomly generated bits to a minimum. The proposed model incorporates the notion of discrete recombination as used in Evolution Strategies [2], to the personal bests, proposing novel velocity update rules and a blend of *lbest* and *gbest* topological models.

The idea is to consider *lbest* neighborhoods without the self, so that the neighborhood of the j-th particle is comprised only of his left and right neighbors in the circular array:

$$N(j) = \{left(j), right(j)\} \tag{5}$$

Within these neighborhoods, a recombinant $\mathbf{r}_j^t$is generated by coordinate-wise random selection from the corresponding coordinate values of the two neighbors:

$$r_{ji} = \begin{cases} p_{left(j)i} & \text{if } RAND() = 0 \\ p_{right(j)i} & \text{otherwise} \end{cases} \tag{6}$$

where $RAND()$ is a 1-bit (0 or 1) random number.

One can think of at least two modifications to the *inertia weight* update rule of Eq. 2 using this *recombinant target*. The first one replaces the neighborhood best by the recombinant while keeping the personal best:

$$\mathbf{v}_j^{t+1} = w \cdot \mathbf{v}_j^t + \varphi_1 \cdot (\mathbf{p}_j^t - \mathbf{x}_j^t) + \varphi_2 \cdot (\mathbf{r}_j^t - \mathbf{x}_j^t). \tag{7}$$

The second one replaces the personal best by the recombinant while keeping the neighborhood best:

$$\mathbf{v}_j^{t+1} = w \cdot \mathbf{v}_j^t + \varphi_1 \cdot (\mathbf{r}_j^t - \mathbf{x}_j^t) + \varphi_2 \cdot (\mathbf{p}_{b(j)}^t - \mathbf{x}_j^t). \tag{8}$$

Notice the replacement of the random variables $\mathbf{U}[0, \varphi_1]$ and $\mathbf{U}[0, \varphi_2]$ of the original algorithm by the fixed constants φ_1 and φ_2. This fact allows an important simplification of the necessary RNG, given that only Mn random bits need to be generated per iteration (to produce the recombinants).

Taking into account the typical choices of $\varphi_1 = \varphi_2 = 2$ in the standard PSO and the fact that the expected value of $\mathbf{U}[0, 2]$ is equal to 1, $\varphi_1 = \varphi_2 = 1$ reveals as a natural choice for the update rules of Eq. 7 and Eq. 8. This choice eliminates two of the multiplications required in the original algorithm. If, in addition to that, a constant inertia weight of 0.5 is assumed, PSODR can be implemented without the need of any multiplier: a multiplication by 0.5 is just a right shift operation.

We consider two PSODR models: the *lbest* model and the *gbest* model. The first one uses a *lbest* topology and a velocity update rule given by Eq. 7. The second one uses a *lbest* topology to calculate the recombinant $\mathbf{r}_j^t$, but a *gbest* topology to calculate the neighborhood best. The pseudocode of the algorithm is shown is shown in the Alg. 1.

Algorithm 1 PSODR

```
procedure PSODR(METHOD)
    Initialize positions, velocities and personal bests
    repeat
        for each particle j in the population do
            if f(x_j) < f(p_j) then
                p_j = x_j
                if method is gbest then
                    if f(p_j) < f(p_g) then
                        g = j
                    end if
                end if
            end if
            for each dimension i do
                r = RAND()
                if r = 0 then
                    k = left(j)
                else
                    k = right(j)
                end if
                if method is gbest then
                    v_ji = w·v_ji + (p_ki − x_ji) + (p_gi − x_ji)
                else(method is lbest)
                    v_id = w·v_ji + (p_ki − x_ji) + (p_ji − x_ji)
                end if
                v_ji ∈ (−V_max, V_max)
                x_ji = x_ji + v_ji
            end for
        end for
    until termination condition is reached
end procedure
```

4 Binary Radial Basis Functions

Artificial neural networks (ANNs) are structures of densely interconnected neurons. Each of these neurons receives an input vector and processes it by passing its inner product with a *weight vector* through an *activation function* [5]. In practice, ANNs allow to efficiently design any function by setting the correct parameters (weights). This efficiency is provided thanks to their cellular architecture and the possibility of applying optimization algorithms (learning or evolution) to find the correct set of parameters. By

selecting the correct weights, using the PSODR algorithm for instance, an ANN can be used as channel equalizer.

Given their cellular nature, ANNs lend themselves to hardware implementation. However, several practical problems are faced when implementing them in hardware. Most neuron models, such as perceptron or radial basis functions, use logistics, gaussians or other continuous functions as activation functions. Additionally, each network connection (synapses) requires a multiplier for weighting the inputs to each neuron. Hardware implementation of these functions results expensive in terms of logic resources. An example showing the complexity of such hardware systems is, for instance, the GRD chip, a neural network hardware implementation in which each neuron of the net is implemented in a dedicated DSP [12]. That is the reason why several simplified hardware-oriented neuron and network models have been proposed in the literature [9].

We are mainly interested in simplified low-cost ANNs, so that an entire population of them could be easily embedded in a commercial FPGA. Given that no supervised learning is intended, the constraint of having differential activation functions can be eliminated, and binary activation functions can be considered. One such neural net with binary activation functions is, for instance, a feedforward neural network with Heaviside activation functions. Nonetheless, training this kind of nets could be time consuming, so we consider instead a *binary radial basis function* (BRBF) as it will be described in the following.

In a radial basis function (RBF) net the inputs are applied to a single layer of neurons, whose outputs feed an output adder. The j-th neuron calculates a response $\phi_j(\mathbf{x})$ based on the distance between the center $\mathbf{c}_j$ of its receptive field and the input vector $\mathbf{x}$. The closer the input vector to the center of the receptive field, the higher the output (activation) of the neuron. Standard RBFs use gaussian activation functions, so that the output of the j-th neuron is given by

$$\phi(\mathbf{x}) = \exp\left(-\frac{D(\mathbf{x}, \mathbf{c}_j)}{2\sigma_j^2}\right) \tag{9}$$

where $D(\mathbf{x}, \mathbf{c}_j)$ is the euclidean distance between $\mathbf{x}$ and $\mathbf{c}_j$, and σ_j is the width of the receptive field.

The output y of the net is typically given by

$$y = \sum_{j=1}^{J} \phi_j(\mathbf{x}) \cdot w_j, \tag{10}$$

where w_j is the weight of the output connection corresponding to the j-th neuron, and J is the total number of neurons in the net. An RBF net as the one previously described is badly suited for resource-optimal hardware implementation, because of the required multiplications and the necessity of calculating an exponential function.

To have a truly hardware friendly neural net, we propose a BRBF net in which the activation function is described by

$$\phi_j(\mathbf{x}) = \begin{cases} 1 & \text{if } D(\mathbf{x}, \mathbf{c}_j) \leq \sigma_j \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

and the distance function $D(\mathbf{x}, \mathbf{c}_j)$ is the Manhattan distance function:

$$D(\mathbf{x}, \mathbf{c}_j) = \sum_i |x_i - c_{ji}| \tag{12}$$

With these definitions, the BRBF net is, basically, a *linear approximator with binary features* that does not require any multiplication nor in the distance calculation nor in the output part (the multiplication by 1 or 0 is trivial) and can thus be easily implemented in hardware. A schematic view of the proposed ANN is shown in Fig. 1.

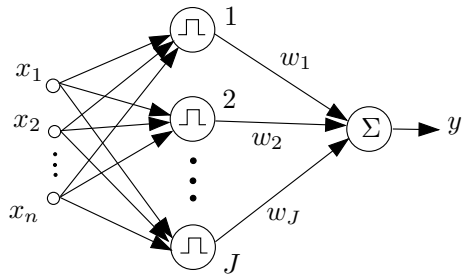

Figure 1. Binary Radial Basis Function

5 Experimental Settings and Results

In order to verify the computational capabilities of the proposed algorithm and neural network, two sets of experiments were devised. First, the two variants (*lbest* and *gbest*) of the proposed PSODR model were tested against their standard PSO counterparts in four benchmark minimization problems. Then, a population of BRBF neural nets was evolved using one variant of PSODR to solve the problem of static channel equalization in a simple communication system model.

5.1 Benchmark Function Minimization

5.1.1 Experimental Settings

Four functions were used in this experiment. The first function is the Sphere function, described by

$$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2. \tag{13}$$

The second function is the Rosenbrock function:

$$f_2(\mathbf{x}) = \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2. \tag{14}$$

The third test function is the Rastrigin function:

$$f_3(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 + 10[1 - \cos(2\pi x_i)]. \quad (15)$$

The fourth test function is the Griewank function:

$$f_4(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1. \quad (16)$$

For all the four functions, the minimum value is 0.

These functions have been widely used in the literature to test different EA and PSO-based algorithms (see, for instance, [1]). The Sphere function is an easy, unimodal function that any optimization technique should be able to solve with a good degree of resolution. It helps to identify good local optimizers. The Rosenbrock function is also unimodal, but generally difficult to optimize even for gradient-based algorithms. The last two functions are multimodal, having many local minima. They help to test the global optimization capabilities of the tested algorithms.

The proposed *gbest* and *lbest* versions of the PSODR models were tested against their standard counterparts, using swarm sizes of 20, 40 and 80 and dimensions of 10, 20 and 30. For the four algorithms a inertia weight of 0.5 was used. For the standard models the acceleration constants were set to $\varphi_1 = \varphi_2 = 2$, while for the PSODR models values of $\varphi_1 = \varphi_2 = 1$ were used. Observe that with this choice of parameters, the PSODR models are hardware friendly, in the sense pointed out in section 3.

The maximum number of iterations was set to 1000, 1500 and 2000, corresponding respectively to dimension sizes of 10, 20 and 30.

The positions of the particles were initialized according to the asymmetric initialization method proposed in [1]. Table 1 shows the the initialization ranges and the values of v_{max} for each function.

Table 1. Initialization ranges and v_{max} for each function

Function	Initialization Range	v_{max}
Sphere	$(50, 100)^n$	100
Rosenbrock	$(-100, 100)^n$	100
Griewank	$(-600, 600)^n$	10
Rastrigin	$(-5.12, 5.12)^n$	600

5.1.2 Results and Discussion

Tables 2, 3, 4 and 5 list the values of the best solution found at the final iteration by each method. The following nomenclature was used: SPSOG (standard *gbest* PSO), SPSOL (standard *lbest* PSO without the self), PSODRG (*gbest* PSODR) and PSODRL (*lbest* PSODR). The results are averaged over 50 independent runs.

Table 2. Mean fitness values for the Sphere function

M	D	SPSOG	SPSOL	PSODRG	PSODRL
20	10	9.03E-39	6.04E-17	**1.18E-51**	2.77E-38
	20	6.07E-24	1.69E-9	**1.95E-44**	3.06E-30
	30	2.24E-18	1.69E-6	**1.72E-39**	4.65E-27
40	10	2.61E-46	5.75E-18	**1.16E-54**	1.44E-38
	20	3.69E-30	6.95E-10	**1.54E-49**	1.62E-30
	30	1.37E-23	8.96E-7	**5.82E-48**	3.40E-27
80	10	2.98E-52	2.93E-18	**6.38E-57**	6.52E-39
	20	1.22E-36	3.97E-10	**1.83E-52**	9.56E-31
	30	1.94E-28	4.59E-7	**1.35E-52**	2.13E-27

Table 3. Mean fitness values for the Rosenbrock function

M	D	SPSOG	SPSOL	PSODRG	PSODRL
20	10	31.14	12.70	17.75	**9.11**
	20	80.62	73.84	**28.78**	30.42
	30	157.90	163.18	**59.05**	88.24
40	10	18.78	6.42	**3.79**	4.44
	20	61.32	27.18	**16.68**	23.68
	30	80.04	74.68	**40.84**	60.30
80	10	10.40	2.02	**1.65**	1.93
	20	80.82	14.37	**3.26**	14.53
	30	76.98	63.82	**19.42**	39.86

As it can be seen, one of the two proposed methods always performed better than the standard methods for the two unimodal functions. The best method was PSODRG, obtaining the minimum values in all the cases, except for Rosenbrock with $M = 20$ and $D = 10$, where it was beaten by PSODRL. The proposed *lbest* method also performed better than the standard *lbest* model.

For the multimodal functions, the best proposed method performed better than the best standard method in 11 of the 18 cases. In the Rastrigin function, PSODRG was the best when dealing with low dimensionalities of the problem, but SPSOG performed better in higher dimensionalities. PSODRL was always outperformed by SPSOL. For the Griewank function, the *lbest* models performed better than the *gbest* models. Here, the standard version was better when dealing with dimension sizes of 10, while the proposed version was better when dealing with dimension sizes of 20 and 30. PSODRG performed better than SPSOG in all the cases for this function.

The proposed *gbest* PSODR always performed better than its standard counterpart for the Sphere, Rosenbrock and Griewank functions and in 5 out of 9 cases in the Rastrigin function (low dimensionality of the search space). On the other hand, the proposed *lbest* PSODR always performed better than its standard counterpart for the Sphere function, in the majority of the cases (10 out of 11) for the Rosenbrock function and the Griewank function (6 out of 9), but worst in all the cases for the Rastrigin function.

Figures 2, 3, 4 and 5 show the learning performance of

Table 4. Mean fitness values for the Rastrigin function

M	D	SPSOG	SPSOL	PSODRG	PSODRL
20	10	8.26	8.68	**7.34**	15.22
	20	**35.26**	37.24	39.96	75.22
	30	**77.55**	76.43	98.90	148.95
40	10	4.22	5.63	**2.99**	13.63
	20	22.76	30.16	**22.60**	68.52
	30	**48.20**	64.84	54.66	144.14
80	10	3.02	4.70	**1.93**	11.54
	20	15.90	26.94	**11.57**	60.80
	30	**34.50**	59.42	35.26	132.58

Table 5. Mean fitness values for the Griewank function

M	D	SPSOG	SPSOL	PSODRG	PSODRL
20	10	7.54E-2	**4.51E-2**	7.16E-2	1.00E-1
	20	2.28E-2	6.50E-3	1.26E-2	**4.96E-3**
	30	1.32E-2	5.05E-3	1.29E-2	**1.84E-3**
40	10	7.60E-2	**3.61E-2**	5.69E-2	7.35E-2
	20	2.25E-2	2.42E-3	1.55E-2	**7.94E-4**
	30	1.69E-2	1.82E-3	5.71E-3	**2.84E-4**
80	10	6.95E-2	**2.12E-2**	4.29E-2	4.02E-2
	20	2.66E-2	4.12E-4	1.35E-2	**6.69E-5**
	30	1.49E-2	5.52E-4	5.02E-3	**7.75E-7**

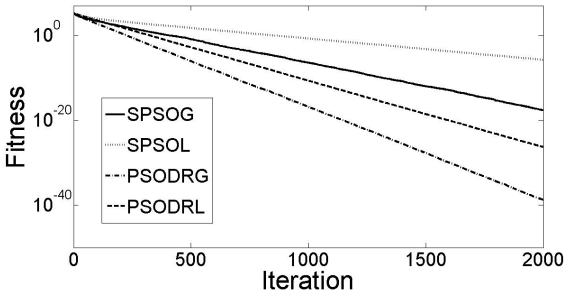

Figure 2. Learning performance for the Sphere function with $M = 20$ and $D = 30$

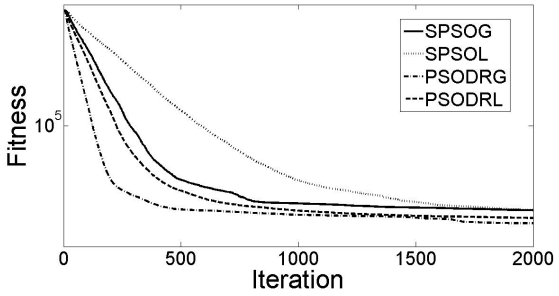

Figure 3. Learning performance for the Rosenbrock function with $M = 20$ and $D = 30$

the different algorithms for the four tested functions with $M = 20$ and $D = 30$. The curves represent the fitness of the best solution ($f(\mathbf{p}_g^t)$) averaged over 50 runs. Observe how the PSODR models are characterized by a faster

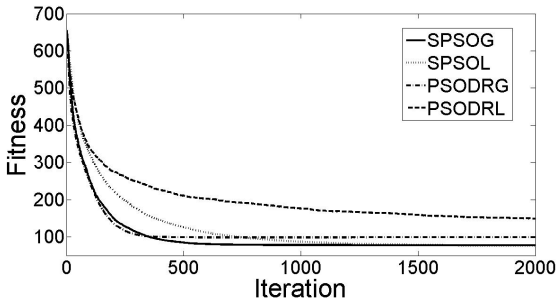

Figure 4. Learning performance for the Rastrigin function with $M = 20$ and $D = 30$

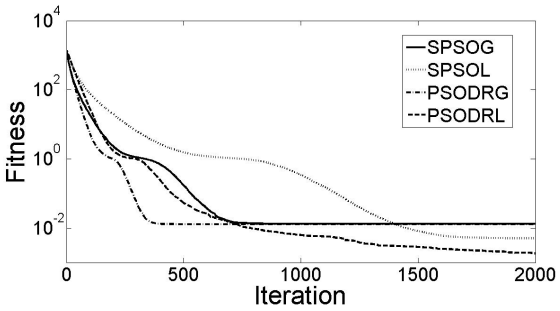

Figure 5. Learning performance for the Griewank function with $M = 20$ and $D = 30$

convergence than the SPSO models (except for Rastrigin function and the *lbest* case, in which SPSOL converges faster than PSODRL). In particular, the PSODRG model converges faster than any of the other methods. It can also be easily seen that the *lbest* models converge slower than the *gbest* methods. This slower convergence does not pay off in the unimodal functions, but allows the discovery of better solutions in one of the multimodal functions (Griewank).

5.2 Channel Equalization of a Digital Communications System

In a digital communication system, a series of symbols $s(t)$ is generated in a source and transmitted over a channel to a receiver. In practice, the channel is not ideal and data is corrupted with nonlinear distortion, intersymbolic interference (ISI) and noise.

One way to alleviate these problems and obtain reliable data transmission is to use a channel equalizer in the receiver [14]. The task of the equalizer is to reconstruct the original signal $s(t)$ from the received signal $r(t)$ or, in other words, to generate a reconstructed version $\hat{s}(t)$ of $s(t)$ as close as possible to it (Fig. 6). The addition of an equalizer usually reduces the bit error rate (BER): the ratio of received bits in error to total transmitted bits.

Traditional adaptive equalization relies on the use of a linear transversal filter. This filter is generally adjusted using a known training sequence at the beginning of the trans-

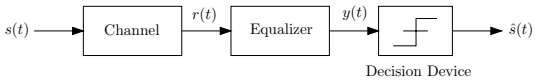

Figure 6. Communication System Model

mission and LSE or gradient descent to determine the optimal set of coefficients for the filter. However, when nonlinear distortion and intersymbolic interference are severe, nonlinear equalizers such as neural nets can give a better performance [13]. Nonetheless, the training of these structures generally involves the use of backpropagation or other related, computational expensive, supervised techniques.

In [12], Murakawa et al. presented the GRD chip and used it for adaptive channel equalization. The GRD chip is a group of 15 DSPs connected in a binary-tree network that implement a feed forward neural network. The net is reconfigured and trained by a genetic algorithm and steepest gradient descent running in an embedded RISC processor. The population of solutions does not have material existence: only one physical net is implemented by the network of DSPs, with each individual being downloaded for evaluation. From our point of view, the proposed solution presents two main drawbacks. First, even if a genetic algorithm is used, the type of learning is essentially supervised, needing a training sequence to be transmitted from the source. Second, the solution is rather expensive, since a single neuron is implemented in a dedicated DSP.

Nowadays, commercially available FPGAs benefit from large amounts of configurable resources, allowing the implementation of very complex circuits. In our approach, we will consider the FPGA implementation of a whole population of very simple neural networks (e.g, BRBF nets), along with an embedded soft-processor responsible for running the adaptation mechanism (e.g, the proposed PSODR) and the reconfiguration of the population of nets. The setup of the complete system will consist of a self-reconfigurable platform as the one described in [16].

Based on hardware synthesis reports, a single 15-neuron BRBF-network with 8-bit data resolution, implemented in a Virtex-II 2v4000 FPGA from Xilinx, requires 2% of the FPGA's logic resources. Therefore, it is reasonable to imagine a self-reconfigurable platform with a MicroBlaze soft-processor reconfiguring a population of until 30 BRBF networks embedded in a single Virtex-II 2v4000 FPGA.

We plan to use this platform for the solution of adaptive channel equalization. In order to do this without the need of a training sequence, the BER of each neural network-based equalizer will be estimated by means of an error detection code. Using these measures, the PSODR will adapt the different parameters of the nets in the population, finding incrementally a good solution in the search space, and decreasing the BER of the whole system. The best solution found so far will be always physically present, giving the actual output of the equalizer (Fig. 7).

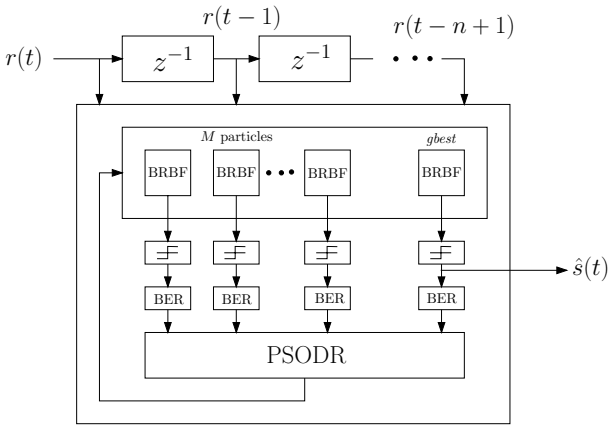

Figure 7. The Proposed Equalizer

In this paper we focus on a simulation of the system for the stationary channel case, leaving both the non-stationary case and the actual hardware implementation for the future work.

For the sake of comparison, we used the communication system proposed in [12]. The source transmits a randomly generated sequence of bipolar symbols (-1 and $+1$) through a linear channel with additive, zero-mean Gaussian noise. The transfer function of the channel is $H(z) = 1 + 1.5z^{-1}$. The order of the equalizer (number of delay elements at the input of the equalizer) was thus set to $n = 2$ (see Fig. 7).

Using this setup, populations of 10, 20 and 30 BRBF nets with $J = 15$ neurons each were evolved using the *gbest* version of PSODR with $w = 0.5$ and $\varphi_1 = \varphi_2 = 1$. The PSODR algorithm was responsible for adapting not only the output weights w_j of the net, but also the centers $\mathbf{c}_j$ and the widths σ_j of the neurons: a $2J + nJ = 60$-dimensional search space.

A generation comprises the reception of 10^4 symbols by every BRBF net. For the sake of simplicity, we assumed an ideal BER estimator, that estimates the BER of each particle as the ratio of misclassified to total number of symbols in the output of the decision device. The decision device uses the threshold function:

$$\hat{s}(t) = \begin{cases} -1 & \text{if } y(t) < 0 \\ 1 & \text{otherwise} \end{cases} \tag{17}$$

The learning performance of this simulation for a signal-to-noise-ratio (SNR) of 15dB is shown in the Fig. 8. For each population size, each curve shows the measured BER of the *gbest* solution for each generation, averaged over 100 independent runs. As it can be seen, the BER is improved over the generations showing a satisfactory learning process. When compared with the results given in [12] for this experiment, the final averaged BER obtained by our solution is much lower than both the one provided by a traditional linear transversal filter and by the GRD system. For

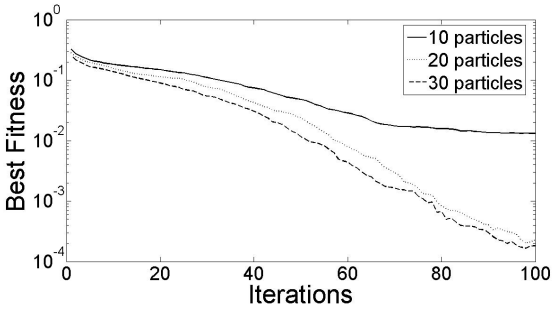

Figure 8. Learning performance of the proposed equalizer for a SNR of 15 dB

the population size of 10, the improvement is of 5 times, whereas for 20 and 30 particles the improvement is of about 2 orders of magnitude. This is a significant result, specially when comparing the population sizes (80 in the referenced work) and the computational complexity of the two approaches.

6 Conclusions and Future Work

We have presented PSODR, a simple, efficient model for stochastic optimization, that takes the concept of recombination from the evolutionary computation field and incorporates it into the general framework of particle swarm optimizers. When tested in benchmark optimization problems, the *gbest* and *lbest* PSODR variants show a better performance than the standard PSO algorithms. Most importantly, this improvement *is not achieved by computationally complicating the algorithm, but by making it simpler*.

The proposed adaptive platform, in which a population of simple neural nets with material existence are evaluated and reconfigured by means of a coprocessor running the proposed optimization algorithm, seems to be well suited to cope with the problem of channel equalization in a communication system, at least for the stationary case. The non stationary case must be investigated, and modifications to PSODR to make it suitable for dynamic optimization have to be explored. More important, the hardware implementation of the proposed solution is to be realized in a commercially available FPGA.

Regarding the optimization algorithm itself, it has to be more deeply tested in other theoretical benchmark functions as well as in more practical problems.

References

[1] P. J. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In *Evolutionary Programming*, pages 601–610, 1998.

[2] H.-G. Beyer and H.-P. Schwefel. Evolution strategies - a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[3] C. A. C. Coello, E. H. Luna, and A. H. Aguirre. Use of particle swarm optimization to design combinational logic circuits. In *ICES*, pages 398–409, 2003.

[4] V. G. Gudise and G. K. Venayagamoorthy. Fpga placement and routing using particle swarm optimization. In *ISVLSI*, pages 307–308, 2004.

[5] S. S. Haykin. *Neural networks : a comprehensive foundation*. Prentice Hall, Upper Saddle River, N.J., 2nd edition, 1999. Simon Haykin. ill. ; 25 cm.

[6] J. Kennedy, R. Eberhart, and Y. Shi. *Swarm intelligence*. Morgan Kaufmann, 2001.

[7] E. H. Luna, A. H. Aguirre, and C. A. C. Coello. On the use of a population-based particle swarm optimizer to design combinational logic circuits. In *2004 NASA/DoD Conference on Evolvable Hardware (EH'04)*, page 183, 2004.

[8] R. Mendes. *Population Topologies and Their Influence in Particle Swarm Performance*. PhD thesis, Escola de Engenharia, Universidade do Minho, May 2004.

[9] P. D. Moerland and E. Fiesler. Neural network adaptations to hardware implementations. In E. Fiesler and R. Beale, editors, *Handbook of Neural Computation*, pages E1.2:1–13. Institute of Physics Publishing and Oxford University Publishing, New York, 1997. IDIAP-RR 97-17.

[10] P. Moore and G. K. Venayagamoorthy. Evolving combinational logic circuits using a hybrid quantum evolution and particle swarm inspired algorithm. In *Evolvable Hardware*, pages 97–102, 2005.

[11] M. Murakawa, S. Yoshizawa, and T. Higuchi. Adaptive equalization of digital communication channels using evolvable hardware. In *ICES*, pages 379–389, 1996.

[12] M. Murakawa, S. Yoshizawa, I. Kajitani, X. Yao, N. Kajihara, M. Iwata, and T. Higuchi. The grd chip: Genetic reconfiguration of dsps for neural network processing. *IEEE Trans. Computers*, 48(6):628–639, 1999.

[13] N. C. Peng, M. and J. Proakis. Adaptive equalization for pam and qam signals with neural networks. *Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems and Computers*, 1991.

[14] S. Qureshi. Adaptive equalization. *IEEE Communications Magazine*, pages 9–16, March 1992.

[15] Y. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *IEEE International Conference on Evolutionary Computation*, May 1998, Anchorage, Alaska, USA.

[16] A. Upegui and E. Sanchez. On-chip and on-line self-reconfigurable adaptable platform: the non-uniform cellular automata case. *Proceedings of the 13th Reconfigurable Architectures Workshop (RAW06) at the IEEE International Parallel Distributed Processing Symposium (IPDPS06)*, 2006.

[17] G. K. Venayagamoorthy and V. G. Gudise. Swarm intelligence for digital circuits implementation on field programmable gate arrays platforms. In *Evolvable Hardware*, pages 83–86, 2004.

[18] X. Yao and T. Higuchi. Promises and challenges of evolvable hardware. In *ICES*, pages 55–78, 1996.