Proceedings of the WSEAS International Conferences:

6th WSEAS Int. Conf. on POWER SYSTEMS (PE '06)
6th WSEAS Int. Conf. on SIMULATION, MODELLING AND OPTIMIZATION (SMO '06)
6th WSEAS Int. Conf. on SIGNAL, SPEECH AND IMAGE PROCESSING (SSIP '06)
6th WSEAS Int. Conf. on MULTIMEDIA, INTERNET & VIDEO TECHNOLOGIES (MIV '06)
6th WSEAS Int. Conf. on DISTANCE LEARNING and WEB ENGINEERING (DIWEB '06)

Lisbon, Portugal, September 22-24, 2006

EDITORS:

Ana Maria Madureira (Portugal)
Jose Carlos Quadrado (Portugal)
Zoran Bojkovic (Serbia)
Sebastiano Impedovo (Italy)
Damir Kalpic (Croatia)
Zoran Stjepanovic (Slovenia)
Shahram Javadi (Iran)

ASSOCIATE EDITOR:

Zita A. Vale (Portugal)

http://www.wseas.org

ISSN: 1790-5117
ISBN: 960-8457-53-X

Evolutionary Techniques in Circuit Design and Optimization

CECÍLIA REIS, J. A. TENREIRO MACHADO
Department of Electrical Engineering
Institute of Engineering of Porto
R. Dr. António Bernardino de Almeida, Porto
PORTUGAL
{cmr,jtm}@isep.ipp.pt

J. BOAVENTURA CUNHA
Engineering Department
Univ. of Trás-os-Montes and Alto Douro
Apt. 1013, 5000-911 Vila Real
PORTUGAL
jboavent@utad.pt

Abstract: - Several Evolutionary Algorithms (EAs) are applied in the design and optimization of digital circuits, namely Genetic Algorithms (GAs), Memetic Algorithms (MAs) and swarm intelligence with Particle Swarm Optimization (PSO). GAs are optimization and search techniques based on the principles of genetics and natural selection. MAs are evolutionary algorithms that include a stage of individual optimization as part of its search strategy, being the individual optimization in the form of a local search. The combination of a global and a local search is a strategy used by many successful hybrid optimization approaches. PSO is a population-based search algorithm that starts with a population of random solutions called particles. In a PSO scheme each particle flies through the search space with a velocity that is adjusted dynamically according with its historical behavior. Therefore, the particles have a tendency to fly towards the best search area along the search process. In this line of thought, this paper presents the results for digital circuits design using the three above EAs.

Key-Words: - Digital circuits, Evolutionary computation, Genetic algorithms, Memetic algorithms, Optimization.

1 Introduction

In recent decades Evolutionary Computation (EC) techniques have been applied to the design of electronic circuits and systems, leading to a novel area of research called Evolutionary Electronics (EE) or Evolvable Hardware (EH) [1]. EE considers the concept for automatic design of electronic systems. Instead of using human conceived models, abstractions and techniques, EE employs search algorithms to develop implementations not achievable with the traditional design schemes, such as the Karnaugh or the Quine-McCluskey Boolean methods.

This paper proposes three evolutionary techniques for the design of combinational logic circuits, namely a Genetic Algorithm (GA), a Memetic Algorithm (MA) and a Particle Swarm Optimization (PSO) scheme.

Bearing these ideas in mind, the organization of this article is as follows. Section 2 presents the GA, the MA is described in section 3 and the PSO is detailed in section 4. Section 5 exhibits the computational results. Finally, section 6 outlines the main conclusions.

2 The Genetic Algorithm

In our previous work, we have developed a GA for combinational logic circuits design [2]. The circuits are specified by a truth table, can have multiple inputs and multiple outputs, and the goal is to implement a functional circuit with the least possible complexity. For that purpose, it is defined a set of logic gates and the circuits are generated with components of that specific set.

1. Initialize the population
2. Calculate the fitness of each individual in the population
3. Reproduce selected individuals to form a new population
4. Perform evolutionary operations such as crossover and mutation on the population
5. Loop to step 2 until some condition is met

Fig. 1: Evolutionary computation algorithm.

Table I shows the four gate sets defined, being Gset 2 the simplest one and Gset 6 a more complex gate set.

For each gate set the GA searches the solution space of a function through a simulated evolution aiming the survival of the fittest strategy. In general, the best individuals of any population tend to reproduce and survive, thus improving successive generations. However, inferior individuals can, by chance, survive and reproduce [3]. In our case, the individuals are digital circuits, which can evolve until the solution is reached (in terms of functionality and complexity).

Table 1 Gate sets

Gate Set	Logic gates
Gset 6	{AND,OR,XOR,NOT,NAND,NOR,WIRE}
Gset 4	{AND,OR,XOR,NOT,WIRE}
Gset 3	{AND,OR,XOR,WIRE}
Gset 2	{AND,XOR,WIRE}

In what concerns to the circuit encoding as a chromosome, EH systems develop chromosomes that encode the functional description of a given circuit. As with many GA applications, the resulting circuit is the phenotype, as it comprises several smaller logic cells or genotypes. The adopted terminology reflects the conceptual similarity between EH, natural evolution and genetics.

In the GA scheme a rectangular matrix (row × column = $r \times c$) of logic cells encodes de circuits (figure 1) [4].

Three genes represent each cell: <*input1*><*input2*><*gate type*>, where *input1* and *input2* are one of the circuit inputs, if they are in the first column, or one of the previous outputs, if they are in other columns. The *gate type* is one of the elements adopted in the gate set. As many triplets of this kind, as the matrix size demands, constitute the chromosome. For example, the chromosome that represents a 3 × 3 matrix is depicted in figure 2.

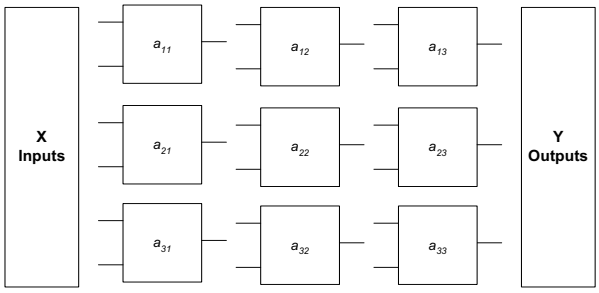

Fig. 1: A 3 × 3 matrix **A** representing a circuit with input **X** and output **Y**.

The GA starts by generating the initial population of circuits (strings) at random. The search is then carried out among this population. The three different operators used are reproduction, crossover and mutation, as described in the sequel.

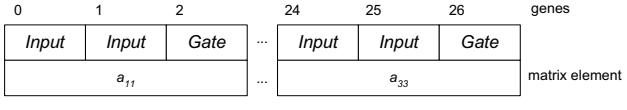

Fig. 2: Chromosome for the 3 × 3 matrix of figure 1.

Successive generations of new strings are reproduced on the basis of their fitness function. In this case, tournament selection [3] is used to select the strings from the old population, up to the new population.

For the crossover operator the strings in the new population are grouped together into pairs at random. Single point crossover is then performed among pairs. The crossover point is only allowed between cells to maintain the chromosome integrity.

The mutation operator changes the characteristics of a given cell in the matrix. Therefore, it modifies the gate type and the two inputs, meaning that a completely new cell can appear in the chromosome. An elitist algorithm is applied to retain the best solutions for the next generation.

To run the GA we have to define the number of individuals to create the initial population *P*. This population is always the same size across the generations, until the GA reaches the solution.

The crossover rate *CR* represents the percentage of the population *P* that reproduces in each generation. Likewise, *MR* is the percentage of the population *P* that mutates in each generation.

The calculation of the fitness function *F* has two parts f_1 and f_2 that measure the functionality and the simplicity, respectively. Firstly, we compare the output produced by the GA-generated circuit with the expected values, according with the truth table, on a bit-per-bit basis (*i.e.*, f_1). Once the circuit is functional, the GA tries to generate circuits with the least number of gates. Therefore, the index f_2, that measures the simplicity, is increased by *one* (*zero*) for each *wire* (*gate*) of the generated circuit, yielding:

$$f_{10} = 2^{ni} \times no \tag{1a}$$

$$f_1 = f_1 + 1 \quad \text{if } \{\text{bit } i \text{ of } \mathbf{Y}\} = \{\text{bit } i \text{ of } \mathbf{Y_R}\},\ i = 1, \ldots, f_{10} \tag{1b}$$

$$f_2 = f_2 + 1 \text{ if } \textit{gate type} = \textit{wire} \tag{1c}$$

$$F = \begin{cases} f_1, & F < f_{10} \\ f_1 + f_2, & F \geq f_{10} \end{cases} \tag{1d}$$

where *ni* and *no* represent the number of inputs and outputs of the circuit.

The GA has three stop criteria with the following hierarchy: *i*) based on the matrix size, it is reached a possible best solution; *ii*) the variation of the average fitness function, for 10 consecutive generations, is less or equal to 1 (the algorithm has stabilized) and *iii*) after having attained 10^4 generations.

3 The Memetic Algorithm

In this work we adopt a MA, that is, an evolutionary algorithm that includes a stage of individual optimization as part of its search strategy, being the individual optimization in the form of a local search. MAs are inspired by models of adaptation in natural systems that combine evolutionary adaptation of populations with individual learning within a lifetime. As it is known, MAs are metaheuristics that take advantage of the evolutionary operators in determining interesting regions of the search space. Moreover, MAs adopt a local search that rapidly finds good solutions in a small region of the search space. Additionally, MAs are inspired by Richard Dawkins' concept of a meme, which represents a unit of cultural evolution that can exhibit local refinement [5].

As figure 3 shows, the proposed MA includes a GA and a local search algorithm, where the GA corresponds to the algorithm implemented in first stage of development.

1. Initialize the population
2. Calculate the fitness of each individual in the population
3. Reproduce selected individuals to form a new population
4. Perform evolutionary operations such as crossover and mutation on the population
5. Apply a local search algorithm
5. Loop to step 2 until some condition is met

Fig. 3 Memetic algorithm.

Over the last decade MAs have relied on the use of a variety of different methods as the local improvement procedure. Some recent studies on the choice of local search method employed have shown that this choice significantly affects the efficiency of problem searches.

The local search method investigates a small area around a solution and adopts the best-found solution. By other words, the procedure tries to find a fitter solution in the neighborhood of the current solution. If the algorithm finds a better solution, then the new solution replaces the current solution, and the neighborhood restarts. Local search methods are iterative algorithms that seek to enhance the solution by stepwise improvements. The simplest form of local search attempts to swap elements in combinatorial optimization problems [6].

4 The Particle Swarm Algorithm

4.1 Introduction

In the literature about PSO the term 'swarm intelligence' appears rather often and, therefore, we begin by explaining why this is so.

Non-computer scientists (ornithologists, biologists and psychologists) did early research, which led into the theory of particle swarms. In these areas, the term 'swarm intelligence' is well known and characterizes the case when a large number of individuals are able of accomplish complex tasks. Motivated by these facts, some basic simulations of swarms were abstracted into the mathematical field. The usage of swarms for solving simple tasks in nature became an intriguing idea in algorithmic and function optimization.

Eberhart and Kennedy were the first to introduce the PSO algorithm (figure 4) [7], which is an optimization method inspired in the collective intelligence of swarms of biological populations, and was discovered through simplified social model simulation of bird flocking, fishing schooling and swarm theory.

1. Initialize population in hyperspace
2. Evaluate fitness of individual particles
3. Modify velocities based on previous best and global (or neighborhood) best
4. Terminate on some condition
5. Go to step 2

Fig. 4 Particle swarm optimization process.

4.2 Parameters

In the PSO, instead of using genetic operators, as in the case of GAs, each particle (individual) adjusts its flying according with its own and its companions experiences. Each particle is treated as a point in a *D*-dimensional space and is manipulated as described below in the original PSO algorithm:

$$v_{id} = v_{id} + c_1\, rand()(p_{id} - x_{id}) + c_2\, Rand()(p_{gd} - x_{id}) \quad (2a)$$

$$x_{id} = x_{id} + v_{id} \quad (2b)$$

where c_1 and c_2 are positive constants, *rand*() and *Rand*() are two random functions in the range [0,1], $X_i = (x_{i1}, x_{i2},\ldots, x_{iD})$ represents the ith particle, $P_i = (p_{i1}, p_{i2},\ldots, p_{iD})$ is the best previous position (the position giving the best fitness value) of the particle, the symbol g represents the index of the best particle among all particles in the population, and $V_i = (v_{i1}, v_{i2},\ldots, v_{iD})$ is the rate of the position change (velocity) for particle *i*.

Expression (2) represents the flying trajectory of a population of particles. Equation (2a) describes how the velocity is dynamically updated and equation (2b) the position update of the "flying" particles. Equation (2a) is divided in three parts, namely the momentum, the cognitive and the social parts. In the first part the velocity cannot be changed abruptly: it is adjusted based on the current velocity. The second part represents the learning from its own flying experience. The third part consists on the learning group flying experience [8].

The first new parameter added into the original PSO algorithm is the inertia weigh. The dynamic equation of PSO with inertia weigh is modified to be:

$$v_{id} = wv_{id} + c_1 rand()(p_{id} - x_{id}) + c_2 Rand()(p_{gd} - x_{id}) \quad (3a)$$

$$x_{id} = x_{id} + v_{id} \quad (3b)$$

where w constitutes the inertia weigh that introduces a balance between the global and the local search abilities. A large inertia weigh facilitates a global search while a small one facilitates a local search.

Another parameter, called constriction coefficient k, is introduced with the hope that it can insure a PSO to converge. A simplified method of incorporating it appears in equation (4), where k is function of c_1 and c_2 as it is presented in equation (4c).

$$v_{id} = k\left[v_{id} + c_1 rand()(p_{id} - x_{id}) + c_2 Rand()(p_{gd} - x_{id})\right] \quad (4a)$$

$$x_{id} = x_{id} + v_{id} \quad (4b)$$

$$k = 2\left(2 - \phi - \sqrt{\phi^2 - 4\phi}\right)^{-1} \quad where\ \phi = c_1 + c_2, \quad \phi > 4 \quad (4c)$$

4.3 Topologies

There are two different PSO topologies, namely the global version and the local version. In the global version of PSO, each particle flies through the search space with a velocity that is dynamically adjusted according to the particle's personal best performance achieved so far and the best performance achieved so far by all particles. On the other hand, in the local version of PSO, each particle's velocity is adjusted according to its personal best and the best performance achieved so far within its neighborhood. The neighborhood of each particle is generally defined as topologically nearest particles to the particle at each side.

4.4 Algorithm

PSO is an evolutionary algorithm simple in concept, easy to implement and computationally efficient. Figures 1-3 present a generic EC algorithm, a hybrid algorithm, more precisely a MA and the original procedure for implementing the PSO algorithm, respectively.

The different versions of the PSO algorithms are: the real-value PSO, which is the original version of PSO and is well suited for solving real-value problems; the binary version of PSO, which is designed to solve binary problems; and the discrete version of PSO, which is good for solving the event-based problems. To extend the real-value version of PSO to binary/discrete space, the most critical part is to understand the meaning of concepts such as trajectory and velocity in the binary/discrete space.

Kennedy and Eberhart [7] use velocity as a probability to determine whether x_{id} (a bit) will be in one state or another (zero or one). The particle swarm formula of equation (1a) remains unchanged, except that now p_{id} and x_{id} are integers in [0.0,1.0] and a logistic transformation $S(v_{id})$ is used to accomplish this modification. The resulting change in position is defined by the following rule:

$$if\ \left[rand() < S(v_{id})\right] \quad then\ x_{id} = 1; \quad else\ x_{id} = 0 \quad (5)$$

where the function $S(v)$ is a sigmoid limiting transformation and *rand*() is a random number selected from a uniform distribution in the range [0.0,1.0].

The initial population of circuits (particles) has a random generation. The initial velocity of each particle is initialized with zero. The following velocities are calculated applying equation (2a) and the new positions result from using equation (2b). In this way, each potential solution, called particle, flies through the problem space. For each gene is calculated the corresponding velocity. Therefore, the new positions are as many as the number of genes in the chromosome. If the new values of the input genes result out of range, then a re-insertion function is used. If the calculated gate gene is not allowed a new valid one is generated at random. These particles then have memory and each one keeps information of its previous best position (*pbest*) and its corresponding fitness. The swarm has the *pbest* of all the particles and the particle with the greatest fitness is called the global best (*gbest*).

The basic concept of the PSO technique lies in accelerating each particle towards its *pbest* and *gbest* locations with a random weighted acceleration. However, in our case we also use a kind of mutation operator that introduces a new cell in 10% of the population. This mutation operator changes the characteristics of a given cell in the matrix. Therefore, the mutation modifies the gate type and

the two inputs, meaning that a completely new cell can appear in the chromosome.

To run the PSO we have also to define the number P of individuals to create the initial population of particles. This population is always the same size across the generations, until reaching the solution.

5 Computational Results

This section shows the implementation of four different combinational logic circuits, namely, a 2-to-1 multiplexer (*M2-1*), a one-bit full adder (*FA1*), a one-bit full subtractor (*FS1*) and a four-bit parity checker (*PC4*), using the GA, the MA and the PSO algorithms.

Due to the stochastic nature of the EAs, in order to evaluate its performance, for each gate set we perform 20 simulations. The best gate set is the one that requires the smaller average number of generations $Av(N)$ and the smaller standard deviation $S(N)$ to reach the solution.

5.1 Circuit Implementation

The first case study is the *M2-1* circuit, with a truth table with three inputs $\{S_0, I_1, I_0\}$ and one output $\{O\}$. The matrix has a size of $r \times c = 3 \times 3$ and the length of each string representing a circuit (*i.e.*, the chromosome length) is $CL = 27$. Since the 2-to-1 multiplexer has $ni = 3$ and $no = 1$, it results $f_{10} = 8$ and $F \geq 12$.

The second case study is the *FA1* circuit, with a truth table with three inputs $\{A, B, C_{in}\}$ and two outputs $\{S, C_{out}\}$. In this case, the matrix has a size of $r \times c = 3 \times 3$, and the chromosome length is $CL = 27$. Since the one-bit full adder has $ni = 3$ and $no = 2$, it results $f_{10} = 16$ and $F \geq 20$.

The third case study is a *FS1* circuit, with a truth table with three inputs $\{A, B, B_{in}\}$ and two outputs $\{S, B_{out}\}$. In this case, the matrix has a size of $r \times c = 3 \times 3$, and the chromosome length is $CL = 27$. Since the one-bit full adder has $ni = 3$ and $no = 2$, it results $f_{10} = 16$ and $F \geq 20$.

The fourth case study consists on the *PC4* circuit, which has four inputs $\{A_3, A_2, A_1, A_0\}$ and one output $\{O\}$. The size of the matrix is $r \times c = 4 \times 4$ and the chromosome length is $CL = 48$. In this case $ni = 4$ and $no = 1$, resulting $f_{10} = 16$ and $F \geq 24$.

Figure 5 presents the results obtained in terms of Log[$S(N)$] *versus* Log[$Av(N)$] for the *M2-1*, the *FA1*, the *FS1* and the *PC4* circuits and $P = \{100, 500, 1000, 3000\}$.

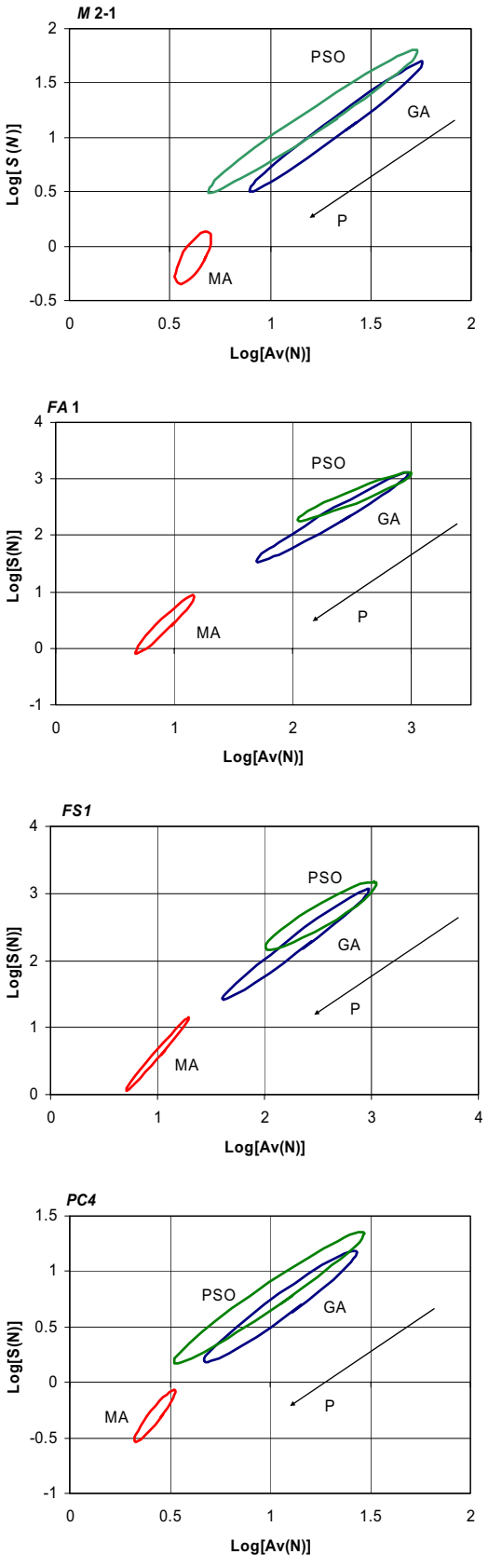

Fig. 5: Log[$S(N)$] *versus* Log[$Av(N)$] for the *M2-1*, the *FA1*, the *FS1* and the *PC4* circuits for $P = \{100, 500, 1000, 3000\}$.

The points in the space {Log[*Av*(*N*)], Log[*S*(*N*)]} are approximated by a bi-dimensional Gaussian probability distribution. The ellipses depicted in the charts represent the corresponding contour plots.

It is obvious that the MA algorithm reveals a better performance for all the combinational circuits and that both *Av*(*N*) and *S*(*N*) vary inversely with *P*. The GA and the PSO algorithms present similar results in particular for the *M2-1* and the *PC4* circuits. For the *FA1* and the *FS1* the PSO is less sensitive to *P* then the GA.

5.2 Comparison of the algorithms

Figure 6 shows Log[*S*(*N*)] *versus* Log[*Av*(*N*)] with *P* = {100, 500, 1000, 3000} for the GA, the MA and the PSO algorithms.

Analysing the charts is possible to classify the complexity of the combinational logic circuits in the perspective of each evolutionary algorithm. For the three algorithms, the sequence of increasing circuit complexity becomes {*PC4*, *M2-1, FA1, FS1*}. In the PSO algorithm, the circuit complexity is clearly divided in two zones, namely the {*FS1*, *FA1*} and the {*M2-1, PC4*} zones.

6 Conclusions

This paper studied the implementation of combinational logic circuits using three evolutionary algorithms. The results reveal that the population size has influence upon the results and that Log[*S*(*N*)] has a linear dependence with Log[*Av*(*N*)], meaning that $S(N) \sim [Av(N)]^{\alpha}$.

The superior performance of the MA algorithm is obvious for all gate sets and all circuits. Moreover, the adopted methodology leads to a classification scheme for combinational logic circuits in terms of their complexity.

References:

[1] Zebulum, R. S., Pacheco, M. A. and Vellasco, M. M., Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms, CRC Press, 2001.

[2] Cecília Reis, J. A. Tenreiro Machado, and J. Boaventura Cunha, "Evolutionary Design of Combinational Logic Circuits", JACIII, Fuji Tec. Press, Vol. 8, No. 5, pp. 507-513, Sept. 2004.

[3] Goldberg, D. E., Genetic Algorithms in Search Optimization and Machine Learning, 1989, Addison-Wesley.

[4] Louis, S.J. and Rawlins, G. J., "Designer Genetic Algorithms: Genetic Algorithms in Structure Design," in Proc. of the Fourth Int. Conf. on Genetic Algorithms, 1991.

[5] Dawkins, R., "The Selfish Gene", Oxford University Press, New York, 1976.

[6] P. Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms", Rep. 826, California Inst. of Tech., California, USA, 1989.

[7] J. Kennedy and R. C. Eberhart. Particle Swarm Optimization. In Proc. of the IEEE Int. Conf. Neural Networks, pp 1942-1948, November 1995.

[8] M. Clerc and J. Kennedy. The Particle Swarm: explosion, stability, and convergence in a multi-dimensional complex space. In IEEE Trans. on Evolutionary Comp., vol. 6, pp. 58-73, 2002.

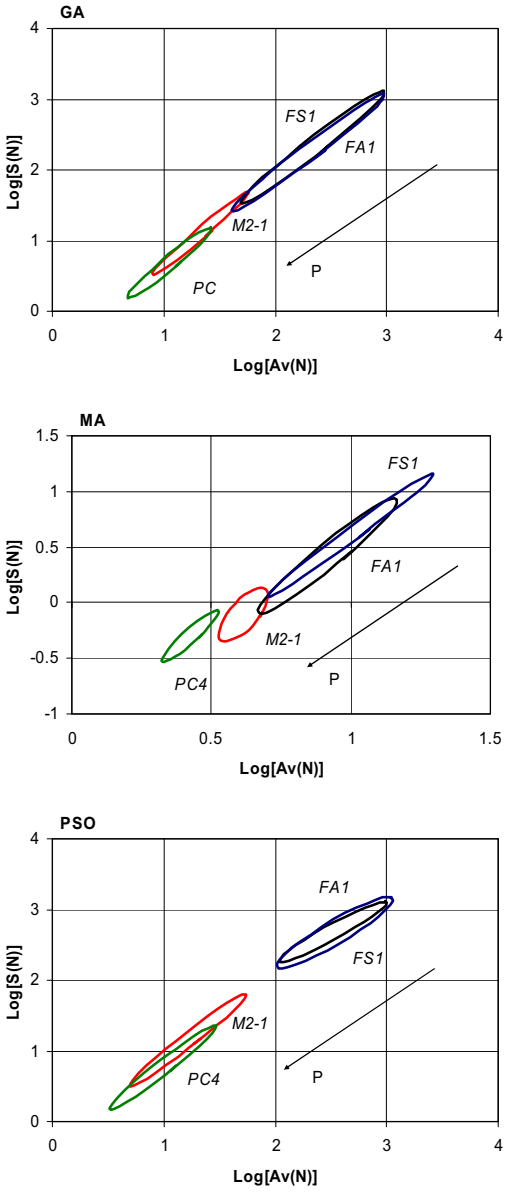

Fig. 6: Log[*S*(*N*)] *versus* Log[*Av*(*N*)] for *P* = {100, 500, 1000, 3000} for the GA, the MA and the PSO algorithms.