CEE'07 - 2nd INTERNATIONAL CONFERENCE ON ELECTRICAL ENGINEERING
ISBN: 978-972-99064-4-2

COMBINING GENETIC AND PARTICLE SWARM ALGORITHMS FOR THE DESIGN OF COMBINATIONAL CIRCUITS

Cecília Reis and J. A. Tenreiro Machado

Institute of Engineering of Porto
Electrical Engineering Department / GECAD
Rua Dr. António Bernardino de Almeida
4200-072 Porto, Portugal

ABSTRACT

Evolutionary computation (EC) is a growing research field of Artificial Intelligence (AI) and is divided in two main areas: the Evolutionary Algorithms (EA) and the Swarm Intelligence (SI). This paper presents an algorithm that combines an EA algorithm - the Genetic Algorithm (GA) with a SI algorithm - the Particle Swarm Optimization Algorithm (PSO). The new algorithm is applied to the synthesis of combinational logic circuits. With this combination is possible to take advantage of the best features of each particular algorithm.

1. INTRODUCTION

In recent decades Evolutionary Computation 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 [3]. 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 Boolean methods: Karnaugh or the Quine-McCluskey.

Several papers proposed designing combinational logic circuits using evolutionary algorithms and, in particular, genetic algorithms (GAs) [2, 1, 4, 8] and hybrid schemes such as the memetic algorithms (MAs) [11]

Another emerging area of research of Artificial Intelligence is the Swarm Intelligence. Swarm Intelligence is a new computational and behavioral paradigm for solving distributed problems based on self-organization. While its main principles are similar to those underlying the behavior of natural systems consisting of many individuals, such as ant colonies and flocks of birds, SI is continuously incorporating new ideas, algorithms, and principles from the engineering and basic science communities.

The authors thank the GECAD Unit for the financial support.

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling.

PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However the PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. The detailed information will be given in following sections.

The advantages of the PSO, relatively to the GA, is that the PSO is easier to implement and involving fewer parameters to adjust.

This paper studies the combination of these two techniques applied to combinational logic circuit synthesis. Bearing these ideas in mind, the organization of this article is as follows. Section 2 presents a brief overview of the GA. Section 3 presents the PSO, while section 4 exhibits the simulation results. Finally, section 5 outlines the main conclusions and addresses perspectives towards future developments.

2. THE GENETIC ALGORITHM

In this section we present the GA developed in the study, in terms of the circuit encoding as a chromosome, the genetic operators and fitness functions.

2.1. Problem Definition

A GA strategy is adopted to design combinational logic circuits. The circuits are specified by a truth table and the goal is to implement a functional circuit with the least possible complexity. Two sets of logic gates have been defined, as shown in Table 1, Gset 6, with six logic gates and Gset 4, with four logic gates. The WIRE means a direct connection (i. e. without any logic gate).

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

Table 1. Gate sets

For each gate set the GA searches the solution space, based on 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 also reproduce [5]. In our case, the individuals are digital circuits, which can evolve until the solution is reached (in terms of functionality and complexity).

2.2. Circuit encoding

In the GA scheme the each circuit is encoded as a rectangular matrix **A** of logic cells as represented in figure 1.

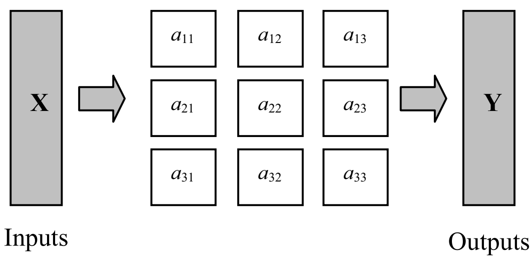

Fig. 1. A 3 $\times$ 3 matrix **A** representing a circuit with input **X** and output **Y**

The three genes: *<input1> <input2> <gate type>* represent each cell, where *input1* and *input2* are one of the circuit inputs, if the cell is in the first column of the matrix, or, one of the outputs of a previous cell, if the cell is not in the first column of the matrix. The *gate type* is one of the elements adopted in the gate set. The chromosome is formed by as many triplets of this kind as the matrix size demands. For example, the chromosome that represents a 3 $\times$ 3 matrix is depicted in figure 2.

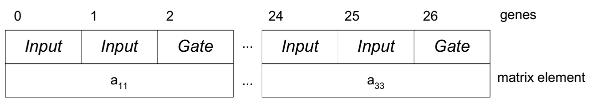

Fig. 2. Chromosome for the 3 $\times$ 3 matrix of figure 1

2.3. The genetic operators

The initial population of circuits (strings) is generated 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.

In what concern the reproduction operator, the successive generations of new strings are reproduced on the basis of their fitness function. In this case, it is used a tournament selection 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. Moreover, it is applied an elitist algorithm and, consequently, the best solutions are always kept 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 solution is reached.

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

2.4. The Fitness Function

The initial population of circuits (strings) is generated 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.

The calculation of F in (1) is divided in two parts, f_1 and f_2, where f_1 measures the functionality and f_2 measures the simplicity. In a first phase, we compare the output **Y** produced by the GA-generated circuit with the required values $\mathbf{Y}_R$, according to the truth table, on a bit-per-bit basis. By other words, f_{11} is incremented by *one* for each correct bit of the output until f_{11} reaches the maximum value f_{10}, that occurs, when we have a functional circuit. Once the circuit is functional, in a second phase, the GA tries to generate circuits with the least number of gates. This means that the resulting circuit must have as much genes *<gate type>* $\equiv$ *<wire>* as possible. Therefore, the index f_2, that measures the simplicity (the number of null operations), is increased by *one* (*zero*) for each *wire* (*gate*) of the generated circuit, yielding:

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

$$f_{11} = f_{11} + 1 \; if \; bit \; i \; of \; \mathbf{Y} = bit \; i \; of \; \mathbf{Y_R}, \quad i = 1, ..., f_{10} \tag{1b}$$

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

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

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

3. PARTICLE SWARM OPTIMIZATION

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 [6], 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.

3.1. 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:

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

$$x_{id} = x_{id} + v_{id} \tag{3}$$

where c_1 and c_2 are positive constants and *rand*() is a random function 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 (1) represents the flying trajectory of a population of particles. Equation (2) describes how the velocity is dynamically updated and equation (3) the position update of the “flying” particles. Equation (2) 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 [7, 9].

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

$$v_{id} = w v_{id} + c_1 \, rand()(p_{id} - x_{id}) + c_2 \, rand()(p_{gd} - x_{id}) \tag{4}$$

$$x_{id} = x_{id} + v_{id} \tag{5}$$

where w constitutes the inertia weight that introduces a balance between the global and the local search abilities. A large inertia facilitates a global search while a small inertia weight facilitates the 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 (3), where k is function of c_1 and c_2 as presented in equation (8).

$$v_{id} = k[v_{id} + c_1 rand p_{id} - x_{id} + c_2 rand p_{gd} - x_{id}] \tag{6}$$

$$x_{id} = x_{id} + v_{id} \tag{7}$$

$$k = 2\left(2 - \phi - \sqrt{\phi^2 - 4\phi}\right)^{-1}, \Phi = c_1 + c_2, \Phi > 4 \tag{8}$$

3.2. 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.

3.3. Algorithm

PSO is an evolutionary algorithm simple in concept, easy to implement and computationally efficient. Figures 3 and 4 present the generic genetic algorithm and the original procedure for implementing the PSO algorithm, respectively.

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. 3. Generic genetic algorithm

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. PSO algorithm

The different versions of the PSO algorithms are: the real-valued 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-valued 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 [4] 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 (2) 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:

$$if\ [rand() < S(v_{id})] \quad then\ x_{id} = 1; \quad else\ x_{id} = 0 \qquad (9)$$

where the function $S(v)$ is a sigmoid limiting transformation.

4. COMBINATION OF THE GA AND THE PSO ALGORITHMS

The algorithm developed in the present work combines a GA with a PSO. The GA is executed in first place and is followed by the PSO. The interlacing of the algorithms is repeated until the solution is found. The number of generations of each algorithm ($n1$ for the GA and $n2$ for the PSO) is initially defined at the moment of running the simulations.

4.1. Experiments and Simulation Results

Reliable execution and analysis of a EA usually requires a large number of simulations to provide a reasonable assurance that stochastic effects have been properly considered. Therefore, in this study are developed n = 20 simulations for each case under analysis.

The experiments consist on running the combination of algorithms to generate a typical combinational logic circuit, namely a 2-to-1 multiplexer ($M2-1$) and a 4-bit parity checker (*PC4*), using the fitness function described previously and the two gate sets presented in table 1.

- the $M2-1$ circuit, has 3 inputs $\mathbf{X} = \{S_0, I_1, I_0\}$ and 1 output $\mathbf{Y}_R = \{\mathrm{O}\}$. The matrix **A** size is 3 × 3, and CL = 27. Since the 2-to-1 multiplexer has *ni* = 3 and *no* = 1, it results f_{10} = 8 and $F \geq 12$,
- the $PC4$ circuit, has 4 inputs $\mathbf{X} = \{A_3, A_2, A_1\ A_0\}$ and 1 output $\mathbf{Y}_R = \{\mathrm{P}\}$. The matrix **A** size is 4 × 4, and the length of each string representing a circuit (*i.e.*, the chromosome length) is CL = 48. In this case *ni* = 4 and *no* = 1, resulting f_{10} = 16 and $F \geq 24$.

Having a superior performance means achieving solutions with a smaller average number of generations *Av*(N) and a smaller standard deviation of the number of generations $S(N)$ to achieve the solution in order to reduce the stochastic nature of the algorithm.

Figures 5 - 8 depict the average number of generations *Av*(N) and the standard deviation of the number of generations to achieve the solution $S(N)$ with $0 \leq n1, n2 \leq 6$ for the $M2-1$ circuit, using the Gsets 6 and 4, respectively.

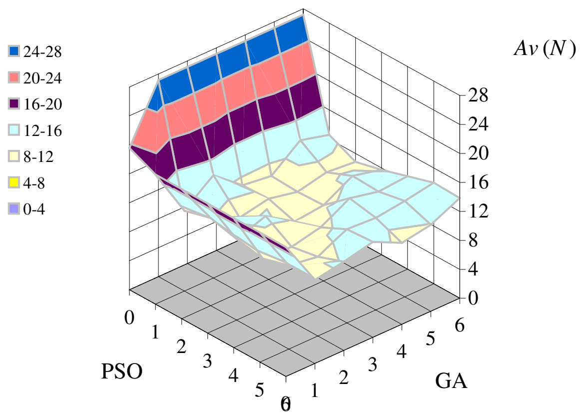

Fig. 5. Average number of generations for solution *Av*(*N*) for the $M2-1$ circuit using Gset 6

ISBN: 978-972-99064-4-2

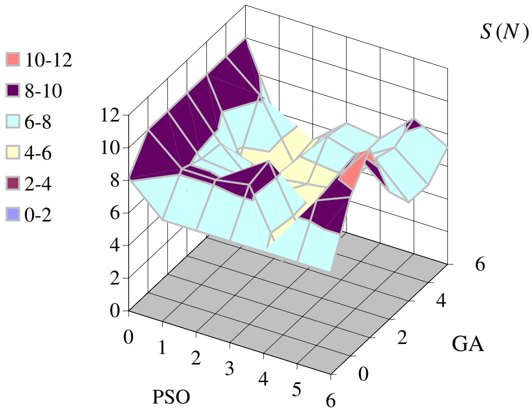

Fig. 6. Standard deviation of the number of generations for solution *S*(*N*) for the $M2-1$ circuit using Gset 6

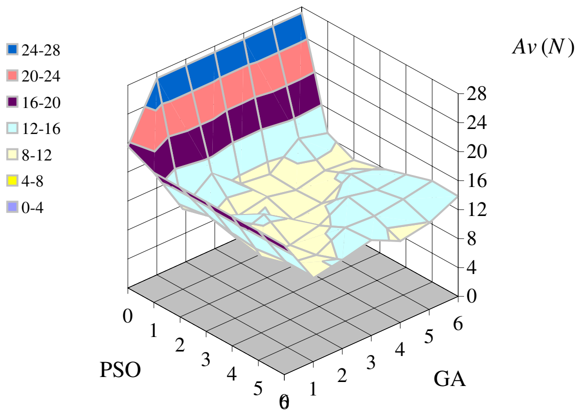

Fig. 7. Average number of generations for solution *Av*(*N*) for the $M2-1$ circuit using Gset 4

Figures 9 - 12 show the average number of generations *Av*(N) and the standard deviation of the number of generations to achieve the solution $S(N)$ with $0 \leq n1$, $n2 \leq 6$ for the $PC4$ circuit, using Gsets 6 and 4, respectively.

Analyzing the charts is possible to see the advantage of combining the two algorithms particularly in respect to the average number of generations *Av*(N).

We verify the existence of an optimal locus from ($n1$, $n2$) = (2,4) up to ($n1$, $n2$) = (4,2).

5. CONCLUSIONS

The main conclusion of this study is that the combination of the evolutionary algorithm with the swarm intelligence algorithm leads to superior results than the execution of the same algorithms individually. With this hybrid algorithm it is possible to take advantage of the benefits of each algorithm.

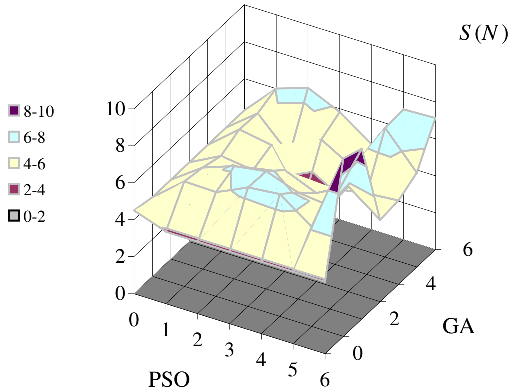

Fig. 8. Standard deviation of the number of generations for solution *S*(*N*) for the $M2-1$ circuit using Gset 4

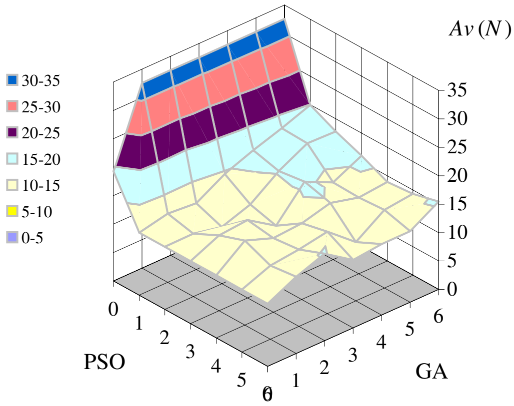

Fig. 9. Average number of generations for solution *Av*(*N*) for the $PC4$ circuit using Gset 6

Future research will address the automatic adjust, during the execution, of the number of iterations $n1$ and $n2$ of each evolutionary algorithm.

6. REFERENCES

[1] S. Louis and G. Rawlins (1991) Designer Genetic Algorithms: Genetic Algorithms in Structure Design. In: Proceedings of the Fourth International Conference on Genetic Algorithms

[2] Goldberg D E (1989) Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley

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

ISBN: 978-972-99064-4-2

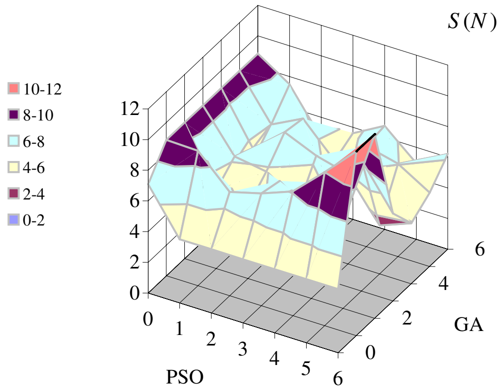

Fig. 10. Standard deviation of the number of generations for solution $S(N)$ for the $PC4$ circuit using Gset 6

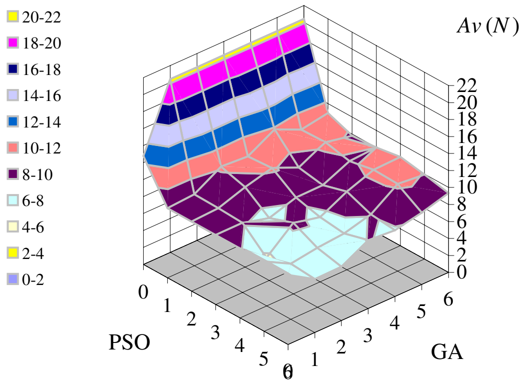

Fig. 11. Average number of generations for solution $Av(N)$ for the $PC4$ circuit using Gset 4

[4] Koza, J. R. (1992) Genetic Programming. On the Programming of Computers by means of Natural Selection, MIT Press

[5] Thompson, A. and Layzell, P. (1999) Analysis of unconventional evolved electronics. In *Comunications. of the ACM*, Vol. 42, pages 71–79

[6] Kennedy J and Eberhart R C (1995) Particle Swarm Optimization. In Proceedings of the IEEE International Conference Neural Networks, pp 1942-1948

[7] Shi Y and Eberhart R C (1998). A Modified Particle Swarm Optimizer. In Proc. of the1998 International Conf. on Evolutionary Computation, pp. 69-73

[8] Coello, C. A., Christiansen, A. D. and Aguirre, A. H. (1996) Using Genetic Algorithms to Design Combinational Logic Circuits. Intelligent Engineering through Artificial Neural Networks. vol. 6, pp. 391-396

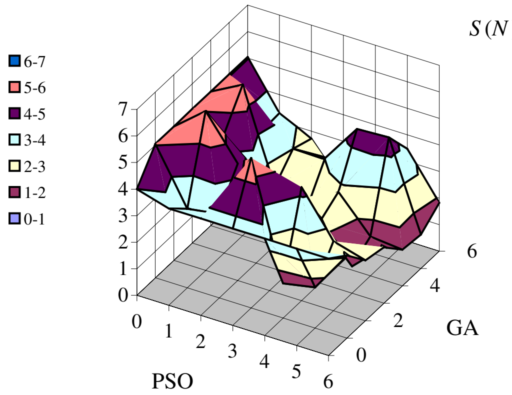

Fig. 12. Standard deviation of the number of generations for solution $S(N)$ for the $PC4$ circuit using Gset 4

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

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

[11] Reis C, Machado J A T and Cunha J B. An Evolutionary Hybrid Approach in the Design of Combinational Digital Circuits. In WSEAS Transactions on Systems, Issue 12, Vol. 4, 2338-2345