2005 WSEAS Int. Conf. on DYNAMICAL SYSTEMS and CONTROL, Venice, Italy, November 2-4, 2005 (pp598-603)

A Memetic Algorithm for Logic Circuit Design

CECÍLIA REIS, J. A. TENREIRO MACHADO
Electrical Engineering Department
Institute of Engineering of Porto
R. Dr. António Bernardino de Almeida, Porto
PORTUGAL

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

Abstract: - Memetic Algorithms (MAs) have shown to be very effective in solving many hard combinatorial optimization problems. In this perspective, this paper presents a MA for combinational logic circuits synthesis. The proposed MA combines a Genetic Algorithm (GA) for digital circuit design with the gate type local search (GTLS). The combination of a global and a local search is a strategy used by many successful hybrid optimization approaches. The main idea is to apply a local refinement to an Evolutionary Algorithm (EA) in order to improve the fitness of the individuals in the population. The obtained results indicate that the MA reduces the number of generations required to reach the solutions and its standard deviation while improves the final fitness function.

Key-Words: - Artificial intelligence, Digital circuits, Evolutionary computation, Genetic algorithms, Logic design, Memetic algorithms.

1 Introduction

In the last decade genetic algorithms (GAs) have been applied in the design of electronic circuits, 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 good designs.

One decade ago Sushil and Rawlins (1991) applied GAs to the combinational circuit design problem [2]. John Koza (1992) adopted Genetic Programming (GP) for the design of combinational circuits through AND, OR and NOT logic gates [3].

Coello, Christiansen and Aguirre (1996) presented a computer program capable of generating high-quality circuit designs [4]. They used five possible types of gates (AND, NOT, OR, XOR and WIRE) with the objective of finding a functional design minimizing the use of gates other than WIRE (essentially a logical no-operation).

Most of the approaches described so far use pure evolutionary methods. This state of affairs motivated the appearance of hybrid techniques in logic circuit design. As it is known, EAs are restricted to relatively simple circuits (with small truth tables). However, the most interesting aspect of evolutionary design is the possibility of studying the emergent patterns [5].

Following this line of research, this paper proposes a hybrid algorithm, named Memetic Algorithm (MA) [6] for the design of combinational logic circuits. Section 2 gives an overview of the background and related work. Section 3 describes the MA approach and presents the adaptation and implementation details. Section 4 compares the GA *versus* the MA results. Section 5 studies the MA convergence while section 6 outlines the scalability problem in logic circuit synthesis. Finally, section 7 summarizes the main conclusions.

2 Background and Related Work

In our previous work, we have developed a GA for combinational logic circuits design [7]. 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.

Table I shows the four gate sets defined, being Gset 2 the simplest one (i.e., a RISC-like set) and Gset 6 a more complex gate set (i.e., a CISC-like 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 [8]. 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).

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.

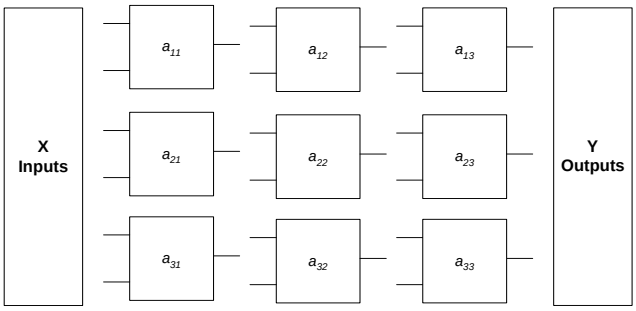

Fig. 1: A 3 × 3 matrix A representing a circuit with input X and output Y.

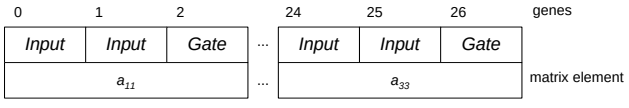

Fig. 2: Chromosome for the 3 × 3 matrix of figure 1.

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.

Successive generations of new strings are reproduced on the basis of their fitness function. In this case, tournament selection [8] 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 (meaning that the algorithm has stabilized) and *iii*) after having attained 10.000 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 of individuals 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 [9]. Bearing these ideas in mind, figure 3 presents the MA implemented in this work.

As figure 4 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.

```
Generate initial population
Evaluate the population
While the stop criteria not attended
   Selection
   Crossover
   Mutation
   Apply Local Search Algorithm
   Evaluate new population
End
```

Fig. 3: The memetic algorithm.

GENETIC ALGORITHM
(Global search algorithm
that generates the initial solutions)
+
LOCAL SEARCH
(Solution improvement algorithm
through stepwise changes of the initial solutions)

Fig. 4: GA and a local search 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.

In our case, it is implemented a gate type local search (GTLS) algorithm as shown in figure 5.

```
For all population
   For the entire chromosome, substitute the gene gate type
   with a neighbour
         If the new solution has better fitness
            New solution replaces old solution
          End For
End
```

Fig. 5: The Local Search Algorithm.

4 Computational Results for the GA and the MA Implementations

This section shows the implementation of four different combinational logic circuits, namely, a 2-to-1 multiplexer, a one-bit full adder, a four-bit parity checker and a two-bit multiplier, using the GA and the MA algorithms.

Due to the stochastic nature of the GAs in order to evaluate its performance, for each gate set we perform 20 simulations. The best gate set is the one that presents the solution with the higher final fitness function F requiring the smaller number of generations N and the smaller standard deviation S.

4.1 2-to-1 multiplexer

The first case study is a 2-to-1 multiplexer 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 $F \geq 12$.

Table 2 shows the average number of generation N_{av}, the standard deviation S_{av} and the average fitness function F_{av}, for each gate set and for the GA and the MA algorithms. We can see that, the best case occurs for Gset 3 with the MA algorithm, because it leads to the smallest N_{av} and the best F_{av}.

Table 2 GA and MA results for all circuits

2-to-1 multiplexer						
	N_{av}		S_{av}		F_{av}	
Gate Set	***GA***	***MA***	***GA***	***MA***	***GA***	***MA***
Gset 6	27.15	10.15	10.00	3.65	10.25	11.55
Gset 4	19.75	5.40	4.29	3.23	10.35	11.95
Gset 3	13.55	3.05	2.98	1.10	10.65	12.00
Gset 2	12.05	6.55	2.78	3.69	11.15	11.80
One-bit full adder						
	N_{av}		S_{av}		F_{av}	
Gate Set	***GA***	***MA***	***GA***	***MA***	***GA***	***MA***
Gset 6	72.45	15.30	52.98	1.75	18.15	19.00
Gset 4	53.65	14.00	29.11	0.86	18.35	19.00
Gset 3	32.40	13.40	10.60	0.50	18.45	19.00
Gset 2	34.86	17.70	6.44	4.59	18.57	18.30
Four-bit parity checker						
	N_{av}		S_{av}		F_{av}	
Gate Set	***GA***	***MA***	***GA***	***MA***	***GA***	***MA***
Gset 6	32.55	2.50	8.85	0.51	21.70	25.10
Gset 4	20.40	2.05	5.05	0.22	21.95	25.85
Gset 3	13.75	2.00	1.80	0.00	22.65	26.00
Gset 2	7.95	2.05	4.10	0.39	23.95	24.50
Two-bit multiplier						
	N_{av}		S_{av}		F_{av}	
Gate Set	***GA***	***MA***	***GA***	***MA***	***GA***	***MA***
Gset 6	1699	56.10	1713	11.59	69.15	70.15
Gset 4	1183	61.85	1652	20.05	69.50	70.70
Gset 3	432	60.05	595	22.85	70.25	71.25
Gset 2	362	293.05	357	225.32	70.45	69.70

4.2 One-bit full adder

The second case study is a one-bit full adder 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 length of each string representing a circuit (i.e., the chromosome length) is $CL = 27$. Since the one-bit full adder has $ni = 3$ and $no = 2$, it results $f_{10} = 16$ and $F \geq 20$.

Table 2 shows N_{av}, the standard deviation S_{av} and F_{av}, for each gate set and for the GA and MA algorithms. We conclude that, once again, the best case occurs for Gset 3 and for the MA algorithm.

4.3 Four-bit parity checker

The third case study consists on a four-bit parity (even) checker circuit, with a truth table having four inputs {A_3, A_2, A_1, A_0} and one output {P}. 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 $F \geq 24$.

Table 2 shows N_{av}, the standard deviation S_{av} and F_{av}, for each gate set, and for the GA and the MA algorithms. Once again, we conclude that Gset 3 in conjunction with the MA algorithm is the best gate set for generating the combinational logic circuit.

4.4 Two-bit multiplier

The fourth case study is a two-bit multiplier. Therefore, the truth table has four inputs {A_1, A_0, B_1, B_0} and four outputs {C_3, C_2, C_1, C_0}. The corresponding matrix is $r \times c = 4 \times 4$ dimensional, the chromosome as size $CL = 48$, and it yields $ni = 4$ and $no = 4$, leading to $f_{10} = 64$ and $F \geq 72$.

Table 2 shows N_{av}, the standard deviation S_{av} and F_{av}, for each gate set for the GA and the MA algorithms. The best results are obtained applying the MA algorithm but, in this case, Gset 6 is the best in terms of N_{av} being Gset 3 superior in respect to F_{av}.

Figure 6 depicts the average of the fitness function F_{av} *versus* the average number of generations to achieve the solution N_{av}, for the GA and the MA algorithms, for all gate sets and all circuits under analysis.

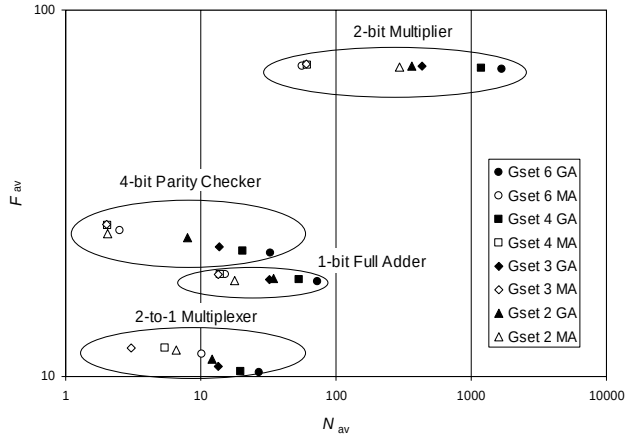

Fig. 6: Average fitness function F_{av} *versus* the average number of generations N_{av} to achieve the solution for $P = 3000$.

The superior performance of the MA algorithm is obvious for all gate sets and all circuits, particularly in the perspective of the N_{av}. Moreover, Gset 3

demonstrates to be the most efficient gate set.

5 Convergence Analysis

This section addresses an important issue of the evolutionary algorithms because in general, due to their stochastic nature, the algorithms may present convergence problems. In this line of thought, we analyze the average number of generations N_{av} to achieve the solution and the standard deviation S_{av}, for different population sizes P.

Figure 7 shows N_{av} *versus* P for the MA algorithm and the four-bit parity checker circuit. In fact, this figure illustrates also the case of the other circuits, because they present similar type of charts.

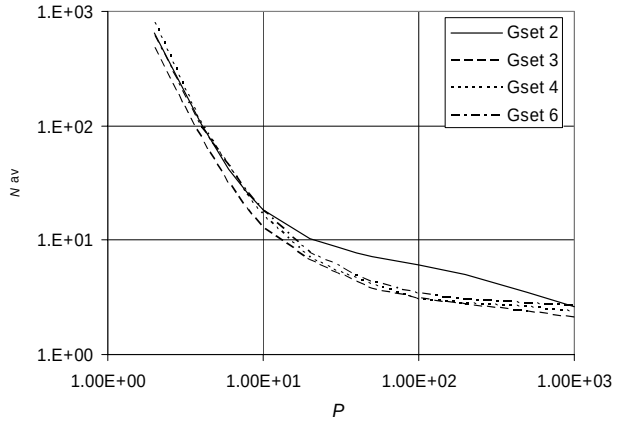

Fig. 7: Average number of generations N_{av} to achieve the solution *versus* the population size P, for the MA algorithm and for the four-bit parity checker circuit, using all gate sets.

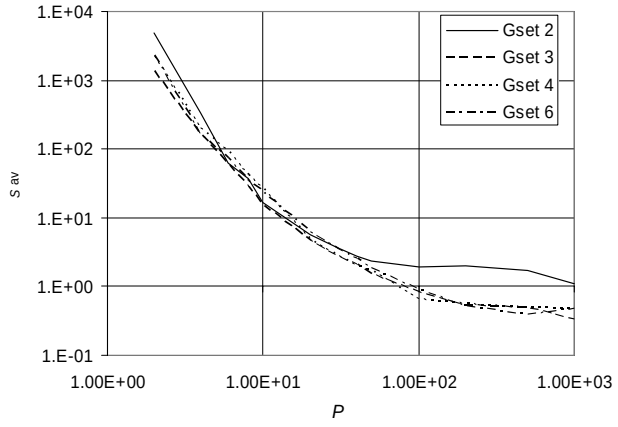

Fig. 8: Standard deviation of the number of generations to achieve the solution S_{av} *versus* the population size P, for the MA algorithm and the four-bit parity checker circuit, using all gate sets.

Figure 8 shows the standard deviation of the number of generations to achieve the solution S_{av} *versus* the population size P, for the MA algorithm and the four-bit parity checker circuit for all the gate sets.

Combining the two charts presented previously, we get the plot of S_{av} versus N_{av} depicted in figure 9. This locus demonstrates the structural behavior of the algorithm convergence. It is clear that, ignoring the processing time, we obtain the better results (a low number of generations to achieve the solution with a low standard deviation) the higher the population P. Similar conclusions result for the other gate sets and rest of the circuits.

For comparison it is also included the locus of the GA scheme for identical conditions.

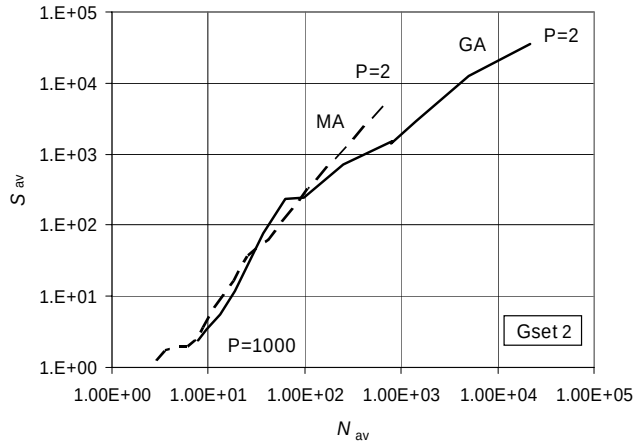

Fig. 9: Standard deviation of the number of generations to achieve the solution S_{av} *versus* the average number of generations to achieve the solution N_{av} for the MA and the GA algorithms, with Gset 2 and the four-bit parity checker circuit.

6 Scalability Analysis

Another issue that emerges with the increasing number of the circuit inputs and outputs is the scalability problem. Since the truth table grows exponentially, the computational burden to achieve the solution increases dramatically.

Figure 10 shows the evolution of F_{av} *versus* N_{av} for the parity checker family of circuits, for an increasing number of bits. The parity checker family is {2, 3, 4, 5 and 6 bit}.

Analyzing the plots for the parity checker family of figure 10 we verify numerically that, after elapsing an initial 'transient', we have an exponential law given by:

$$F_{av} = \alpha e^{\beta N_{av}} \quad \alpha, \beta \in \Re \tag{2}$$

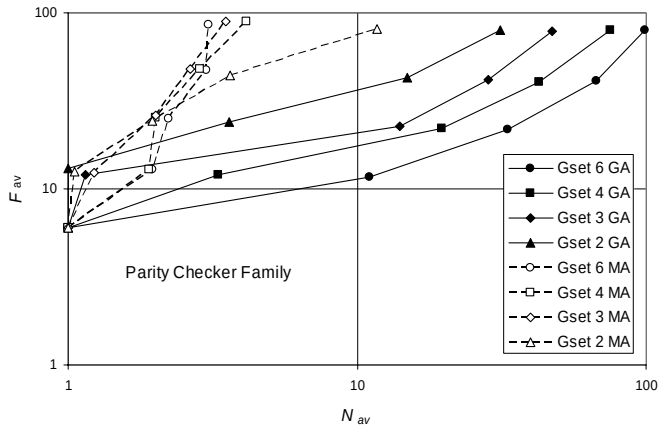

Fig. 10: F_{av} *versus* N_{av} for the parity checker family, for the GA and the MA algorithms and for the Gsets under evaluation for P = 3000.

Table 3 presents the coefficients (α, β) that result for each gate set and for each of the algorithms. For the GA algorithm and with respect to coefficient α we can say that Gset 2 is the best one, Gsets 3 and 4 are similar and that Gset 6 is the less performing. It is possible to group Gsets 6, 4 (inferior performance) and Gsets 2, 3 (superior performance) in terms of coefficient β, that captures the growth characteristics. On the other hand, for the MA algorithm, it is possible to group gate sets 6, 4, 3 for both coefficients while Gset 2 exhibits an inferior behavior.

Table 3 Coefficients of equation 2

	GA		***MA***	
Gate Set	***α***	***β***	***α***	***β***
Gset 6	9.8	0.0214	2	1.06
Gset 4	12.3	0.0257	1.92	1.14
Gset 3	12.2	0.0408	2.33	1.17
Gset 2	21.2	0.0433	8.44	0.47

7 Conclusions

In general, most real world problems are too complex for any single optimization technique to solve it in isolation. The modern trend and philosophy for constructing fast, globally convergent algorithms is to combine a simple globally convergent algorithm with a fast locally convergent heuristic, to form a more suitable and faster hybrid.

GAs are well known for exploring the solution space effectively but are unable to fine-tune the search. In order to improve the GAs search capabilities, a local search technique is often integrated with a GA to form a hybrid called Memetic Algorithms. Accordingly, the hybrid MA tends to incorporate the exploration capability of GAs with the exploitation features of local search, which we have confirm in this work.

The performed experiments have shown that the MA proposed in this paper is highly effective in combinational logic circuit design when compared with classical GA approaches. Furthermore, the local search technique was able to enhance the convergence rate of the Evolutionary Algorithm by finely tuning the search on the immediate area of the landscape considered.

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] Louis, S.J. and Rawlins, G. J., "Designer Genetic Algorithms: Genetic Algorithms in Structure Design," in Proceedings of the Fourth International Conference on Genetic Algorithms, 1991.

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

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

[5] Miller, J. F., Job, D. and Vassilev, V. K., "Principles in Evolutionary Design of Digital Circuits – Part I", Genetic Programming and Evolvable Machines 1(1/2), 7-35, 2000.

[6] P. Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms", Tech. Rep. Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

[7] Cecília Reis, J. A. Tenreiro Machado, and J. Boaventura Cunha, "Evolutionary Design of Combinational Logic Circuits", Journal of Advanced Computational Intelligence and Intelligent Informatics, Fuji Technology Press, Vol. 8, No. 5, pp. 507-513, Sep. 2004.

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

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