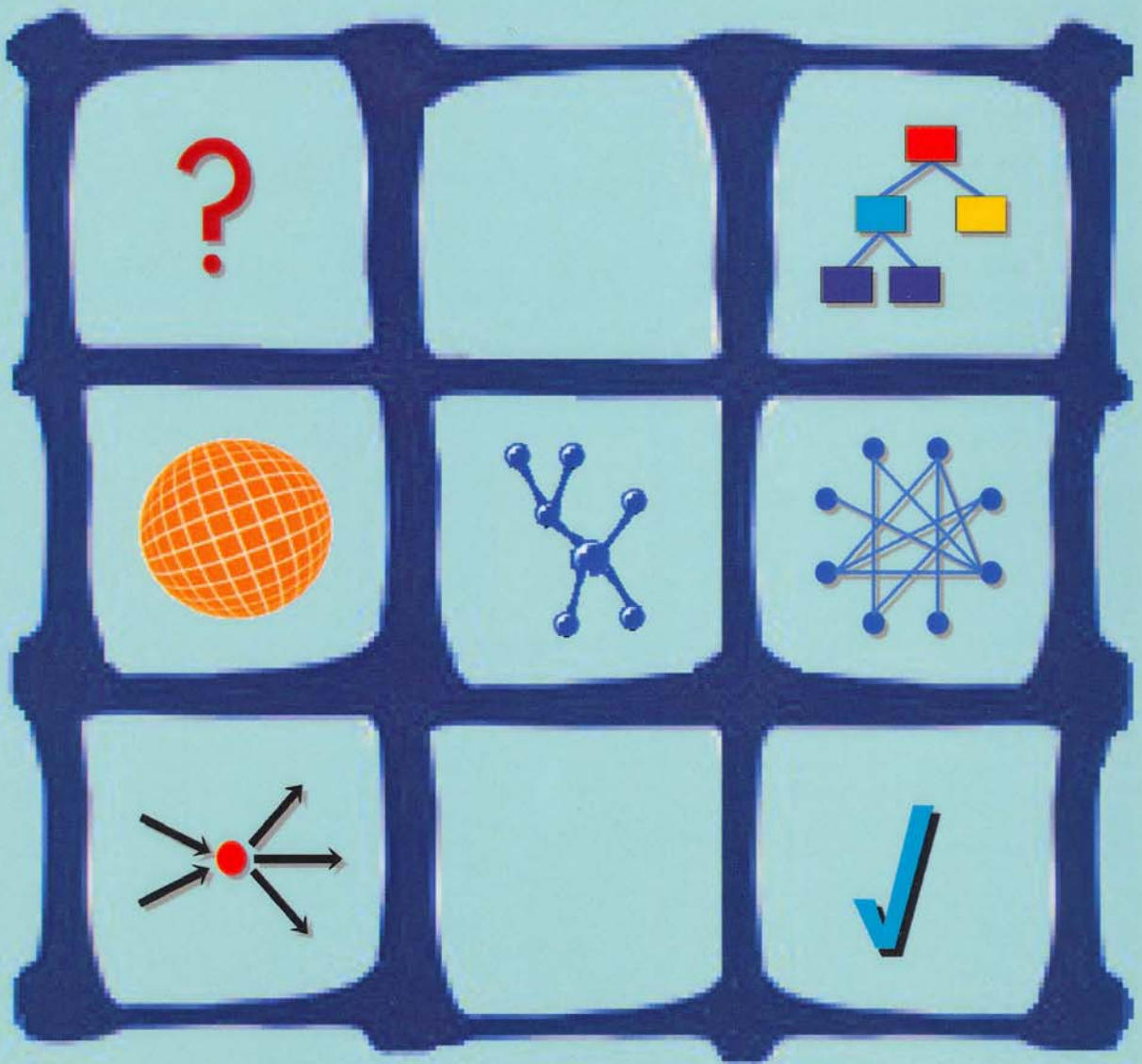
Zita Vale, Carlos Ramos and Luiz Faria (eds.)

Knowledge and Decision Technologies



A HYBRID ALGORITHM FOR LOGIC CIRCUIT SYNTHESIS

Cecília Reis, J. A. T. Machado and L. Figueiredo
GECAD - Knowledge Engineering
and Decision Support Group
Institute of Engineering - Polytechnic of Porto
Porto, Portugal
{cmr,jtm,lbf}@isep.ipp.pt

J. Boaventura Cunha
Engineering Department
Univ. of Trás-os-Montes and Alto Douro
Vila Real, Portugal
jboavent@utad.pt

***Abstract*—In view of the fact that Genetic Algorithms (GAs) are not well suited for fine-tuning structures that are close to optimal solutions [1], this paper suggests the incorporation of local improvement operators into the GA recombination phase. This study presents a hybrid genetic algorithm, also known as Memetic Algorithm (MA), applied to the design of combinational logic circuits. MAs are evolutionary algorithms (EAs) that apply a separate local search process to refine individuals (*i.e.* that improve their fitness by hill-climbing). Under different contexts and situations, MAs are also known as hybrid EAs or genetic local searchers. The proposed MA associates a GA with the gate type local search (GTLS). Combining global and local search is a strategy used by many successful global optimization approaches, and MAs have in fact been recognized as a powerful algorithmic paradigm for evolutionary computing. We also modify the calculation of the fitness function by including a discontinuity evaluation that measures the error variability of the Boolean table. The results show an improvement of the final fitness function followed by a reduction of the average number and the standard deviation of generations required to reach the solutions, for all the tested circuits.**

keywords - Evolutionary algorithms, logic circuits, memetic algorithms

I. 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) [2].

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 [3].

One decade ago Sushil and Rawlins (1991) applied GAs to the combinational circuit design problem. They combined knowledge-based systems with the GA and defined a genetic operator called masked crossover. The scheme produced new types of children that were not possible to achieve with classical crossover operators [4].

John Koza (1992) adopted Genetic Programming (GP) for the design of combinational circuits through AND, OR and NOT logic gates [5].

Coello, Christiansen and Aguirre (1996) presented a computer program capable of generating high-quality circuit designs [6]. 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).

Miller, Thompson and Fogarty (1997) applied evolutionary algorithms for the design of arithmetic circuits. The technique was based on evolving the functionality and connectivity of a rectangular array of logic cells, with a model of the resources available on the Xilinx 6216 FPGA device [7].

Kalganova, Miller and Lipnitskaya (1998) proposed another technique for designing multiple-valued circuits. The EH was easily adapted to distinct types of multiple-valued gates, associated with operations corresponding to different types of algebra, by taking advantage of the capability of including other logical expressions [8]. This approach is, in fact, an extension of EH method for binary logic circuits proposed in [7].

In order to solve complex systems, Torresen (1998) proposed the method of increased complexity evolution. The idea consisted in evolving a system gradually as a kind of divide-and-conquer method. The scheme was first undertaken individually on a large number of simple cells. The resulting functions were the basic blocks adopted in further evolution or assembly of larger and more complex systems [9].

More recently, Hollingworth, Smith and Tyrrell (2000) described the first attempts to evolve circuits using the Virtex Family of devices. They implemented a simple 2-bit adder, where the inputs to the circuit are the two 2-bit numbers and the expected output is the sum of the two input values [10].

A major bottleneck in the evolutionary design of electronic circuits is the problem of scale. This refers to the very fast growth of the number of gates, used in the target circuit, as the number of inputs of the evolved logic function increases. This results in a huge search space that

is difficult to explore even with evolutionary techniques. Another related obstacle is the time required to calculate the fitness value of a circuit [11]. A possible method to solve this problem is to use building blocks either than simple gates. Nevertheless, this technique leads to another difficulty, which is how to define building blocks that are suitable for evolution.

Timothy Gordon (2002) suggested an approach that allows evolution to search good inductive bases for solving large-scale complex problems. This scheme generated, inherently, modular and iterative structures, that exist in many real-world circuit designs but, at the same time, allowed evolution to search innovative areas of space [12].

The approaches described so far, based on pure methods, have poor scalability in logic circuit design, so hybrid methods have been applied in order to try to solve this problem. As it is known EAs are restricted to relatively small circuits (with small truth tables) [13]. However, the most interesting aspect of evolutionary design is the possibility of studying the emergent patterns [13, 14].

Following this line of research, this paper proposes a hybrid algorithm, namely a Memetic Algorithm (MA) [15] 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 for two cases, namely for a fitness function with and without the error discontinuity measure. Section 5 studies the MA convergence while section 6 outlines the scalability problem in logic circuit synthesis. Finally, section 7 summarizes the main conclusions.

II. BACKGROUND AND RELATED WORK

In our previous work, we have developed a GA for combinational logic circuits design [16]. 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 [1]. In our case, the individuals are digital circuits, which can evolve until the solution is reached (in terms of functionality and complexity).

TABLE I
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 [17].

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

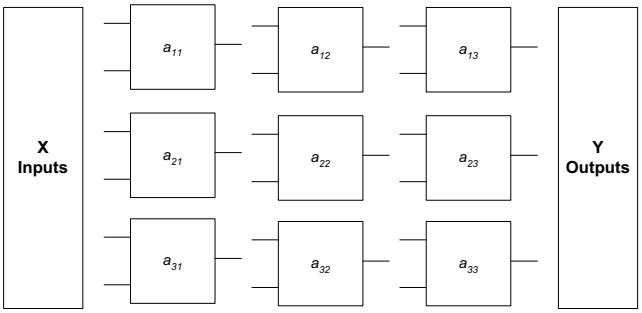


Fig. 1. A 3 × 3 matrix **A** representing a circuit with input **X** and output **Y**

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.

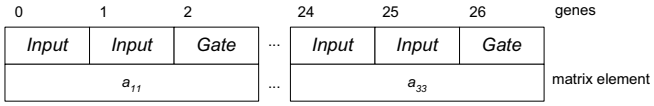


Fig. 2. Chromosome for the 3 × 3 matrix **A**

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, tour-

nament selection [1] 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 F in (1) is divided in two parts, f_1 and f_2, where f_1 measures the circuit functionality and f_2 measures the circuit simplicity. In a first phase, we compare the output $\mathbf{Y}$ produced by the GA-generated circuit with the required values $\mathbf{Y}_R$, according with 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 (eq. 1a and 1b).

In order to measure the output error variability (subsections 3.1 and 3.3) f_{11} is decremented by δ for each $\mathbf{Y}_R - \mathbf{Y}$ error discontinuity (*i.e.*, when passing from $\mathbf{Y}_R - \mathbf{Y} = 0$ to $\mathbf{Y}_R - \mathbf{Y} = 1$, or *vice-versa*) by comparing two consecutive levels of the truth table (eq. 1c).

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 (eq. 1d), yielding:

- First phase, circuit functionality:

$$f_{10} = 2^{ni} \times no \qquad ((1)a)$$

$$\begin{array}{c} f_{11} = f_{11} + 1, \\ if\ \{bit\ i\ of\ \mathbf{Y}\} = \{bit\ i\ of\ \mathbf{Y_R}\}, \\ i = 1, ..., f_{10} \end{array} \qquad ((1)b)$$

$$\begin{array}{c} f_1 = f_{11} - \delta, \\ if\ error_i \neq error_{i-1}, i = 1, ..., f_{10} \end{array} \qquad ((1)c)$$

(when measuring discontinuity)

- Second phase, circuit simplicity:

$$f_2 = f_2 + 1,\ if\ gate\ type = wire \qquad ((1)d)$$

$$F = \begin{cases} f_1, & F < f_{10} \\ f_1 + f_2, & F \geq f_{10} \end{cases} \qquad ((1)e)$$

where $i = 1, \ldots, f_{10}$, 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.

III. THE MEMETIC ALGORITHM

Our previous experiments in GA-based logic circuit design illustrated that the individuals of the population tend to be similar, leading to difficult or to premature convergence. Therefore, 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 [18]. 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 [19]. Additionally, MAs are inspired by Richard Dawkins' concept of a meme, which represents a unit of cultural evolution that can exhibit local refinement [20]. 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.

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 [21].

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.

```
Generate initial population
Evaluate the population
While 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 ALGORITHM
(Solution improvement algorithm through stepwise changes of the initial solutions)

Fig. 4. GA and a local search algorithm

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

```
For all population
  For the entire chromossome
    Substitute de gene gate type with a neighbour
      If the new solution has better fitness function
        New solution replaces old solution
  End for
End
```

Fig. 5. The local search algorithm

IV. 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 four-bit parity checker ($PC4$) and a two-bit multiplier ($MUL2$), as follows:

- 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 $\mathbf{A}$ 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 $F \geq 24$,
- 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 $\mathbf{A}$ 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 $FA1$ circuit, has 3 inputs $\mathbf{X} = \{\mathrm{A},\ \mathrm{B},\ C_{in}\}$ and 2 outputs $\mathbf{Y}_R = \{\mathrm{S},\ C_{out}\}$. The matrix $\mathbf{A}$ size is 3×3, and $CL = 27$. Since the one-bit full adder has $ni = 3$ and $no = 2$, it results $f_{10} = 16$ and $F \geq 20$,
- the $MUL2$ circuit, has 4 inputs $\mathbf{X} = \{A_1,\ A_0,\ B_1,\ B_0\}$ and 4 outputs $\mathbf{Y}_R = \{C_3,\ C_2,\ C_1,\ C_0\}$. The matrix $\mathbf{A}$ size is 4×4, and $CL = 48$ and it yields $ni = 4$ and $no = 4$, leading to $f_{10} = 64$ and $F \geq 72$.

The input bits are grouped according with the Gray Code and Table 2 presents the Boolean truth Tables for the circuits under study.

A. Fitness without discontinuity measure ($\delta = 0$)

In this sub-section we analyze the MA when adopting the fitness function without including the discontinuity measure δ error. The GA results are also included allowing the comparison between the 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 presents the solution with the higher final fitness function F requiring the smaller number of generations N and the smaller standard deviation S.

Analyzing the $PC4$, the $M2-1$ and the $FA1$ circuits 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}.

In respect to the $MUL2$ circuit, the best results are obtained applying the MA algorithm, but in this case Gset 6 is the best in terms of N_{av} being the 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.

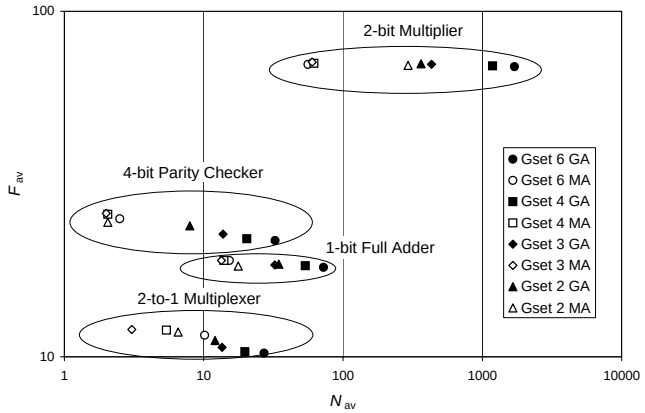


Fig. 6. Average fitness function F_{av} *versus* the average number of generations N_{av} to achieve the solution for $P = 3000$ with $\delta = 0$

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

$PC4$				
A_3	A_2	A_1	A_0	P
0	0	0	0	0
0	0	0	1	1
0	0	1	1	0
0	0	1	0	1
0	1	1	0	0
0	1	1	1	1
0	1	0	1	0
0	1	0	0	1
1	1	0	0	0
1	1	0	1	1
1	1	1	1	0
1	1	1	0	1
1	0	1	0	0
1	0	1	1	1
1	0	0	1	0
1	0	0	0	1

$MUL2$							
A_1	A_0	B_1	B_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	1	0	0	0	0
0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
0	1	0	1	0	0	0	1
0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	1	1	0	0	1
1	1	1	0	0	1	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	0	0	1	0	0	1	0
1	0	0	0	0	0	0	0

$M2-1$			
S_0	I_1	I_0	O
0	0	0	0
0	0	1	1
0	1	1	1
0	1	0	0
1	1	0	1
1	1	1	1
1	0	0	0
1	0	0	0

$FA1$				
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	1	0	1
0	1	0	1	0
1	1	0	0	1
1	1	1	1	1
1	0	0	0	1
1	0	0	1	0

Table II: CIRCUIT TRUTH TABLES

B. Fitness with discontinuity measure (δ = 0.5)

In this sub-section we analyze the MA improvement when adopting the fitness function including the discontinuity measure δ error.

The experiments were done for all the gate sets and all the circuits under study in this paper. Figure 7 presents the results for the $PC4$, the $M2-1$ and the $FA1$ circuits, using a population size $P = 10$. Since the $MUL2$ circuit has a higher complexity it is required to increase the population size to $P = 1000$. The results, are shown in figure 8.

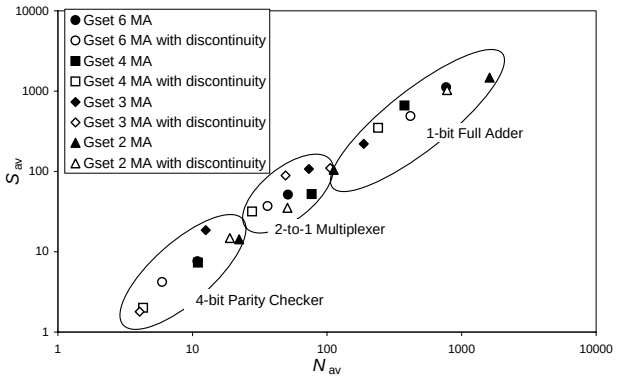


Fig. 7. Average standard deviation S_{av} *versus* the average number of generations N_{av} to achieve the solution for the $PC4$, the $M2-1$ and the $FA1$ circuits with $P = 10$ and using $\delta = 0.5$ (for the MA with discontinuity cases)

The improvement of the MA algorithm through application of the fitness function with the error discontinuity measure δ is more evident in terms of the standard deviation. Moreover, the number of generations to achieve a solution is also slightly improved.

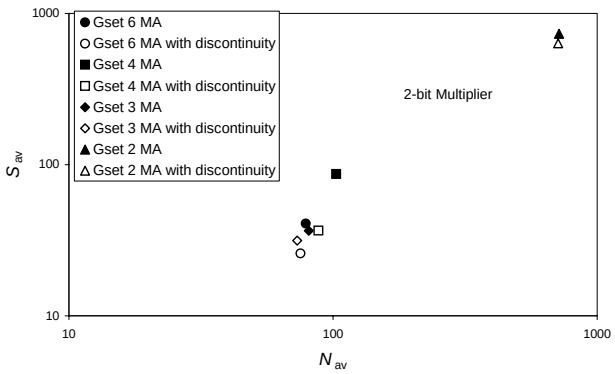


Fig. 8. Average standard deviation S_{av} *versus* the average number of generations N_{av} to achieve the solution for the $MUL2$ circuit with $P = 1000$ using $\delta = 0.5$ (for the MA with discontinuity cases)

V. 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. The logic circuit in study here is the $PC4$ circuit with the gate set 2.

Figure 9 presents the S_{av} *versus* N_{av} for the $PC4$ and the Gset 2, with the processing time PT as parameter. Ignoring the PT [22], we obtain the better results (a low number of generations to achieve the solution with a low standard deviation) the higher the population P. However, in terms of PT the best results occur for $P = 100$ and $P = 20$ for the GA and MA cases, respectively. Similar conclusions result for the other gate sets and rest of the circuits.

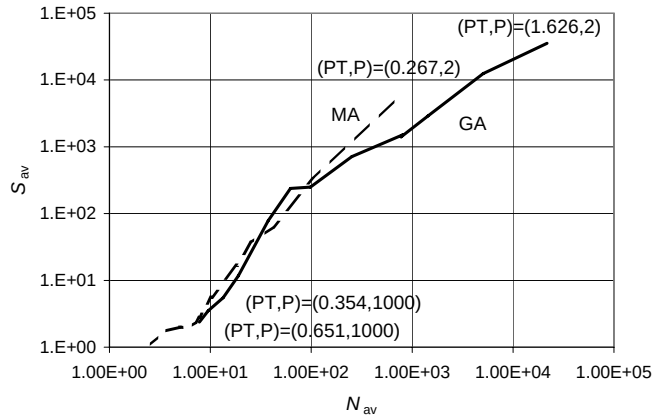


Fig. 9. Processing time PT versus the standard deviation of the number of generations to achieve the solution S_{av} and the average number of generations to achieve the solution N_{av}, for the MA and the GA algorithms, Gset 2 and the four-bit parity checker circuit, with $\delta = 0$

VI. 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 [13]. We have developed this study using the parity checker family circuits of {2, 3, 4, 5 and 6 bit}.

Figure 10 shows the evolution of F_{av} *versus* N_{av} for the parity checker family of circuits, for an increasing number of bits.

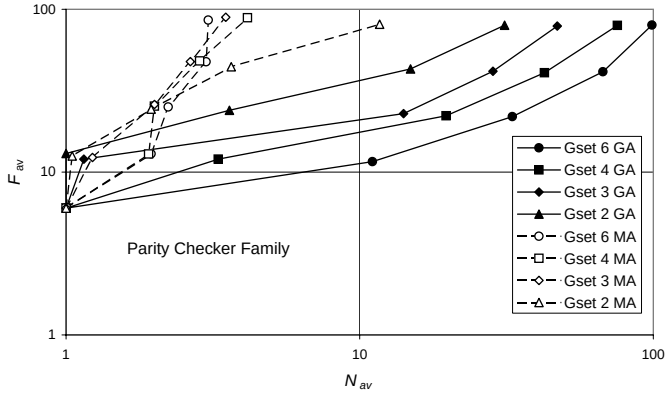


Fig. 10. F_{av} versus N_{av} for the parity checker family, for the GA and the MA algorithms and for the Gsets under evaluation for $P = 3000$, with $\delta = 0$

VII. Conclusions

As it is known, 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 experiments have shown that the MA proposed in this paper is more efficient in combinational logic circuit design when compared with classical GA approaches. On the other hand, the incorporation of the error discontinuity measure in the fitness function calculation leads also to an algorithm improvement.

VIII. Acknowledgements

The authors would like to acknowledge FCT, FEDER, POCTI, POSI, POCI and POSC for their support to R&D Projects and GECAD Unit.

IX. References

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

[2] 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.

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

[4] Louis, S.J. and Rawlins, G. J. Designer Genetic Algorithms: Genetic Algorithms in Structure Design. In *Proc. of the Fourth International Conference on Genetic Algorithms*, 1991.

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

[6] Coello, C. A., Christiansen, A. D. and Aguirre, A. H. Using Genetic Algorithms to Design Combinational Logic Circuits. In *Intelligent Engineering through Artificial Neural Networks*, Vol. 6, pages 391–396, 1996.

[7] Miller, J. F., Thompson, P. and Fogarty, T. *Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications.* Chapter 6, Wiley, 1997.

[8] Kalganova, T., Miller, J. F. and Lipnitskaya, N. Multiple-Valued Combinational Circuits Synthesised using Evolvable Hardware. In *Proc. of the 7th Workshop on Post-Binary Ultra Large Scale Integration Systems*, 1998.

[9] Torresen, J. A Divide-and-Conquer Approach to Evolvable Hardware. In *Proc. of the Second International Conference on Evolvable Hardware*, Vol. 1478, pp. 57–65.1998.

[10] Hollingworth, G. S., Smith, S. L. and Tyrrell, A. M. The Intrinsic Evolution of Virtex Devices Through Internet Reconfigurable Logic. In *Proc. of the Third International Conference on Evolvable Systems*, Vol. 1801, pp. 72–79, 2000.

[11] Vassilev, V. K. and Miller, J. F. Scalability Problems of Digital Circuit Evolution. In *Proc. of the Second NASA/DOD Workshop on Evolvable Hardware*, pp. 55–64, 2000.

[12] Gordon, T. G. and Bentley, P. Towards Development in Evolvable Hardware. In *Proc. of the 2002 NASA/DOD Conference on Evolvable Hardware*, pp. 241–250, 2002.

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

[14] Coello, C. A., Christiansen, A. D. and Aguirre, A. H. Use of Evolutionary Techniques to Automate the Design of Combinational Circuits. In *International Journal of Smart Engineering System Design*, Vol. 2, No. 4, pages 299–314, 2001.

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

[16] Ceclia Reis, J. A. Tenreiro Machado, and J. Boaventura Cunha. Evolutionary Design of Combinational Logic Circuits. In *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 8, No. 5, pages 507–513, Fuji Technology Press, September 2004.

[17] Hounsell, B. and Arslan, T. A Novel Evolvable Hardware Framework for the Evolution of High Performance Digital Circuits. In *Proc. of the Genetic and Evolutionary Computation Conference*, pages 525–532, 2000.

[18] Krasnogor, N. *Studies on the Theory and Design Space of Memetic Algorithms*. PhD thesis, University of the West of England, Bristol, June, 2002.

[19] P. Merz and B. Freisleben. A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem. In *Proc. of the 1999 Congress on Evolutionary Computation*, pages 2063–2070, IEEE Press, 1999.

[20] Dawkins, R. *The Selfish Gene*. Oxford University Press, New York, 1976.

[21] Y. S. Ong and A.J. Keane. Meta-Lamarckian in Memetic Algorithm. In *IEEE Transactions On Evolutionary Computation*, Vol. 8, No. 2, pages 99–110, April, 2004.

[22] Ceclia Reis, J. A. Tenreiro Machado, and J. Boaventura Cunha. Population Size and Processing Time in a Genetic Algorithm. In *Proc. of the IEEE International Conference on Computational Cybernetics*, Vienna University of Technology, 2004.