An Evolutionary Hybrid Approach in the Design of Combinational Digital Circuits

CECÍLIA REIS, J. A. TENREIRO MACHADO
Electrical Engineering Department
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: - This paper presents a hybrid genetic algorithm, also known as Memetic Algorithm (MA), applied to the design of combinational logic circuits. In view of the fact that hybrid algorithms have shown to be very effective in solving many hard combinatorial optimization problems, 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 adopted by recent 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 results show an improvement of the final fitness function followed by a reduction of the average number of generations required to reach the solutions and its standard deviation, for all the tested circuits.

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 [2].

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 [3].

John Koza (1992) adopted Genetic Programming (GP) for the design of combinational circuits through AND, OR and NOT logic gates [4].

Coello, Christiansen and Aguirre (1996) presented a computer program capable of generating high-quality circuit designs [5]. 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 [6].

Kalganova, Miller and Lipnitskaya (1998) proposed another technique for designing multiple-valued circuits. The EH was easily adapted to the 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 [7]. This approach is, in fact, an extension of EH method for binary logic circuits proposed in [6].

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 [8].

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 [9].

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 [10]. 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 for 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 [11].

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) [12]. However, the most interesting aspect of evolutionary design is the possibility of studying the emergent patterns [12, 13].

Following this line of research, this paper proposes a hybrid algorithm, named Memetic Algorithm (MA) [14] 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 [15]. 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 [16]. 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 [17].

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.

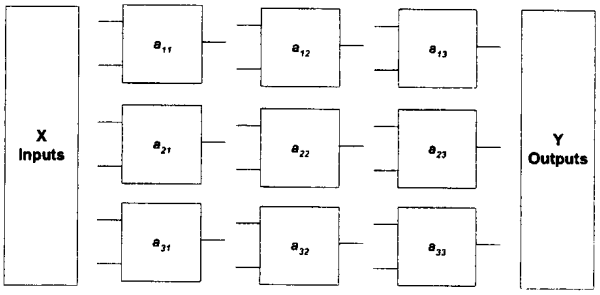

Fig. 1: A 3 × 3 matrix A representing a circuit with input X and output Y.

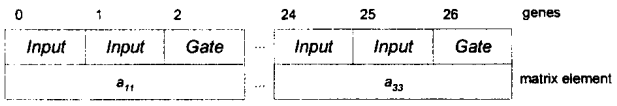

Fig. 2: Chromosome for the 3 × 3 matrix of figure 1.

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, in this case, tournament selection [16] 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:

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

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

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

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

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

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

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].

```
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.

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 the 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

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 3 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 3 GA and MA results for the 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

Table 4 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.

Table 4 GA and MA results for the 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

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 5 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 the gset 3 superior in respect to F_{av}.

Table 5 GA and MA results for the 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

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.

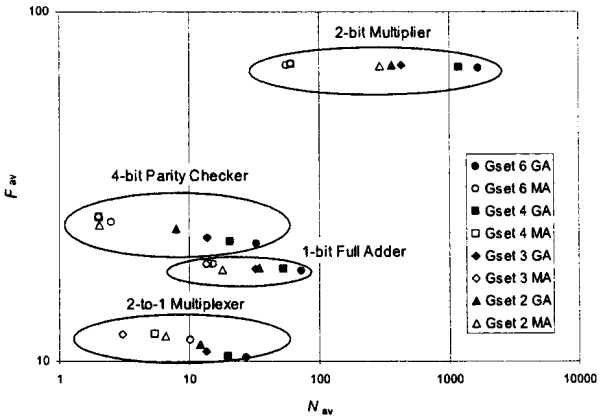

Fig. 6: Average fitness function F_{av} versus the average number of generations N_{av} to achieve the solution for $P = 3000$.

The superior performance of the MA algorithm is obvious for all gate sets and all the circuits, particularly in the perspective of the N_{av}. Moreover, Gset 3 demonstrated 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 4 bit parity checker circuit. In fact, this figure illustrates also the case of the other circuits, because they present similar type of charts.

It is possible to divide the plot in two areas, namely the low P and the high P regions, which follow a power law:

$$N_{av} \approx \alpha P^{\beta} \quad \alpha, \beta \in \Re \tag{2}$$

where the (α, β) parameters are shown in tables 6 and 7.

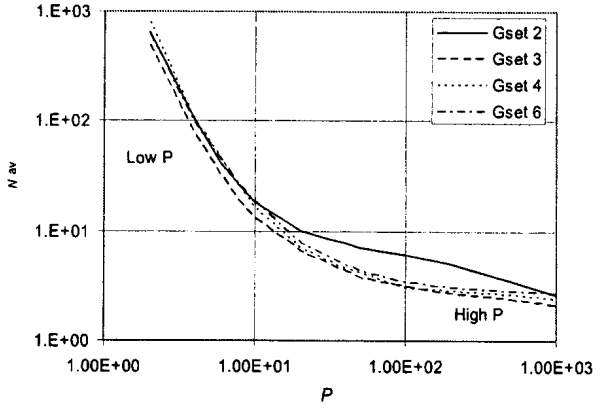

Fig. 7: Average number of generations N_{av} to achieve the solution versus the population size P, for the MA algorithm and for the four-bit parity checker circuit, using all gate sets.

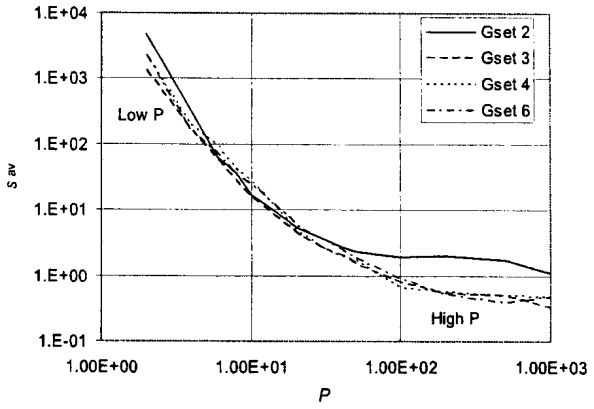

Fig. 8: Standard deviation of the number of generations to achieve the solution S_{av} versus the population size P, for the MA algorithm, gset 2 and the four-bit parity checker circuit.

Table 6 Parameters of $N_{av} \approx \alpha P^{\beta}$ for low P

	2-to-1 Multiplexer		*4 bit Parity Checker*		*1 bit Full Adder*		*2 bit Multiplier*	
Gset	α	β	α	β	α	β	α	β
6	$2\ 10^4$	-2.50	$2.8\ 10^3$	-2.31	$6.7\ 10^5$	-3.27	$1\ 10^6$	-1.43
4	$9.4\ 10^4$	-2.52	$4.2\ 10^3$	-2.39	$1\ 10^6$	-4.03	$1\ 10^6$	-1.40
3	$1.5\ 10^5$	-3.43	$2.4\ 10^3$	-2.39	$1.4\ 10^5$	-3.09	$4\ 10^7$	-2.04
2	$2.6\ 10^5$	-3.32	$3.6\ 10^3$	-2.51	$6\ 10^6$	-3.65	$5\ 10^7$	-1.62

Table 7 Parameters of $N_{av} \approx \alpha P^{\beta}$ for high P

	2-to-1 Multiplexer		*4 bit Parity Checker*		*1 bit Full Adder*		*2 bit Multiplier*	
Gset	α	β	α	β	α	β	α	β
6	24.87	-0.11	5.64	-0.11	$2\ 10^2$	-0.37	$4\ 10^2$	-0.23
4	35.88	-0.26	5.23	-0.11	$1.1\ 10^2$	-0.36	$4\ 10^2$	-0.23
3	38.22	-0.22	7.56	-0.19	$1.1\ 10^2$	-0.34	$5\ 10^2$	-0.28
2	35.44	-0.25	27.99	-0.34	$6.6\ 10^2$	-0.51	$1\ 10^8$	-1.54

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, gset 2 and the 4 bit parity checker circuit. Once again, the other cases under study follow a similar behavior and it is possible to divide the plot in two regions the low P and the high P areas, approximately given by:

$$S_{av} \approx \chi P^{\delta} \quad \chi, \delta \in \Re \tag{3}$$

where the (χ, δ) parameters are shown in tables 8 and 9.

Combining the charts presented previously, we get the plot (Fig. 9) of the processing time PT versus (S_{av}, N_{av}) and its projection in the horizontal plane S_{av} versus N_{av}. 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.

Table 8 Parameters of $S_{av} \approx \chi P^{\delta}$ for low P

	2-to-1 Multiplexer		*4 bit Parity Checker*		*1 bit Full Adder*		*2 bit Multiplier*	
Gset	χ	δ	χ	δ	χ	δ	χ	δ
6	$1\ 10^4$	-2.95	$1.8\ 10^4$	-3.20	$8\ 10^4$	-2.91	$3\ 10^6$	-1.53
4	$2.8\ 10^5$	-3.07	$1\ 10^4$	-2.60	$6\ 10^6$	-4.49	$4.2\ 10^5$	-1.02
3	$6\ 10^5$	-3.70	$8.4\ 10^3$	-2.75	$3.1\ 10^5$	-2.97	$2\ 10^7$	-1.84
2	$5.4\ 10^5$	-3.32	$7.4\ 10^4$	-3.93	$6\ 10^6$	-3.00	$7\ 10^6$	-1.17

Table 9 Parameters of $S_{av} \approx \chi P^{\delta}$ for high P

	2-to-1 Multiplexer		*4 bit Parity Checker*		*1 bit Full Adder*		*2 bit Multiplier*	
Gset	χ	δ	χ	δ	χ	δ	χ	δ
6	0.58	0.22	0.08	0.26	$3.2\ 10^3$	-1.02	$4.7\ 10^2$	-0.41
4	2.6	-0.07	0.66	-0.04	$6.2\ 10^2$	-0.84	$1.5\ 10^3$	-0.56
3	2.42	0.08	11.99	-0.53	$1.7\ 10^2$	-0.76	$2.2\ 10^3$	-0.56
2	0.37	0.21	24.42	-0.44	$1.5\ 10^3$	-0.76	$8\ 10^{10}$	-2.36

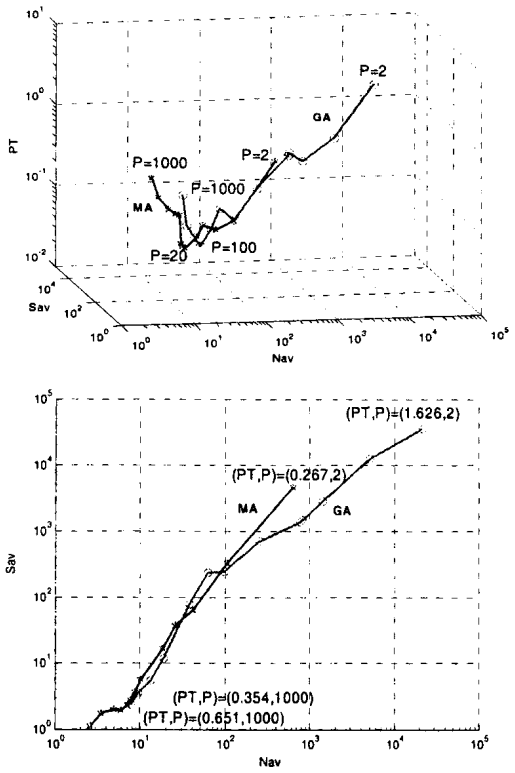

Fig. 9: Processing time PT versus the standard deviation of the number of generations to achieve the solution S_{av} and the average number of generations to achieve the solution N_{av}, for the MA and the GA algorithms, Gset 2 and the four-bit parity checker circuit.

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 [12].

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 that after elapsing an initial 'transient' we have an exponential law given by:

$$F_{av} = \lambda e^{\mu N_{av}} \quad \lambda, \mu \in \Re \tag{4}$$

Table 10 presents the coefficients (λ, μ) that result for each gate set and for each of the algorithms. For the GA algorithm and with respect to coefficient λ 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 μ, 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.

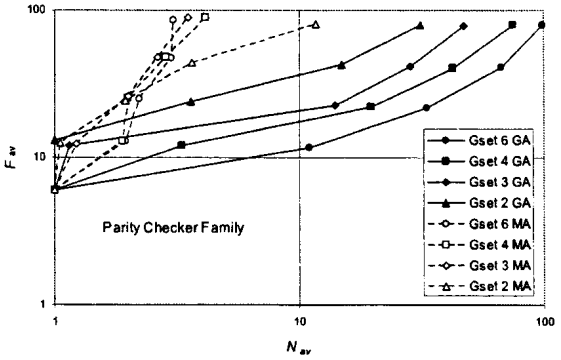

Fig. 10: F_{av} versus N_{av} for the parity checker family, for the GA and the MA algorithms and for the Gsets under evaluation for $P = 3000$.

Table 10 Coefficients of equation 4

	GA		*MA*	
Gate Set	λ	μ	λ	μ
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] Thompson, A. and Layzell, P. "Analysis of unconventional evolved electronics," Com. of the ACM, Vol. 42, 1999, pp. 71-79.

[3] 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.

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

[5] 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.

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

[7] Kalganova, T., Miller, J. F. and Lipnitskaya, N., "Multiple_Valued Combinational Circuits Synthesised using Evolvable Hardware," in Proceedings of the 7th Workshop on Post-Binary Ultra Large Scale Integration Systems, 1998.

[8] Torresen, J., "A Divide-and-Conquer Approach to Evolvable Hardware," in Proceedings of the Second International Conference on Evolvable Hardware. Vol. 1478, 1998, pp. 57-65.

[9] Hollingworth, G. S., Smith, S. L. and Tyrrell, A. M., "The Intrinsic Evolution of Virtex Devices Through Internet Reconfigurable Logic," in Proceedings of the Third International Conference on Evolvable Systems. Vol. 1801, 2000, pp. 72-79.

[10] Vassilev, V. K. and Miller, J. F., "Scalability Problems of Digital Circuit Evolution," in Proceedings of the Second NASA/DOD Workshop on Evolvable Hardware, 2000, pp. 55-64.

[11] Gordon, T. G. and Bentley, P., "Towards Development in Evolvable Hardware," in Proceedings of the 2002 NASA/DOD Conference on Evolvable Hardware, 2002. pp. 241-250.

[12] 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.

[13] Coello, C. A., Christiansen, A. D. and Aguirre, A. H., "Use of Evolutionary Techniques to Automate the Design of Combinational Circuits", International Journal of Smart Engineering System Design 2 (4), 299-314.

[14] 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.

[15] 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.

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

[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, 2000, pp. 525-532.

[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 1999 Congress on Evolutionary Computation, pp. 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", IEEE Transactions On Evolutionary Computation, Vol. 8, No. 2, pp. 99-110, April 2004.

[22] Cecília 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, 30/Aug-1/Sep/2004, Vienna University of Technology, Austria.