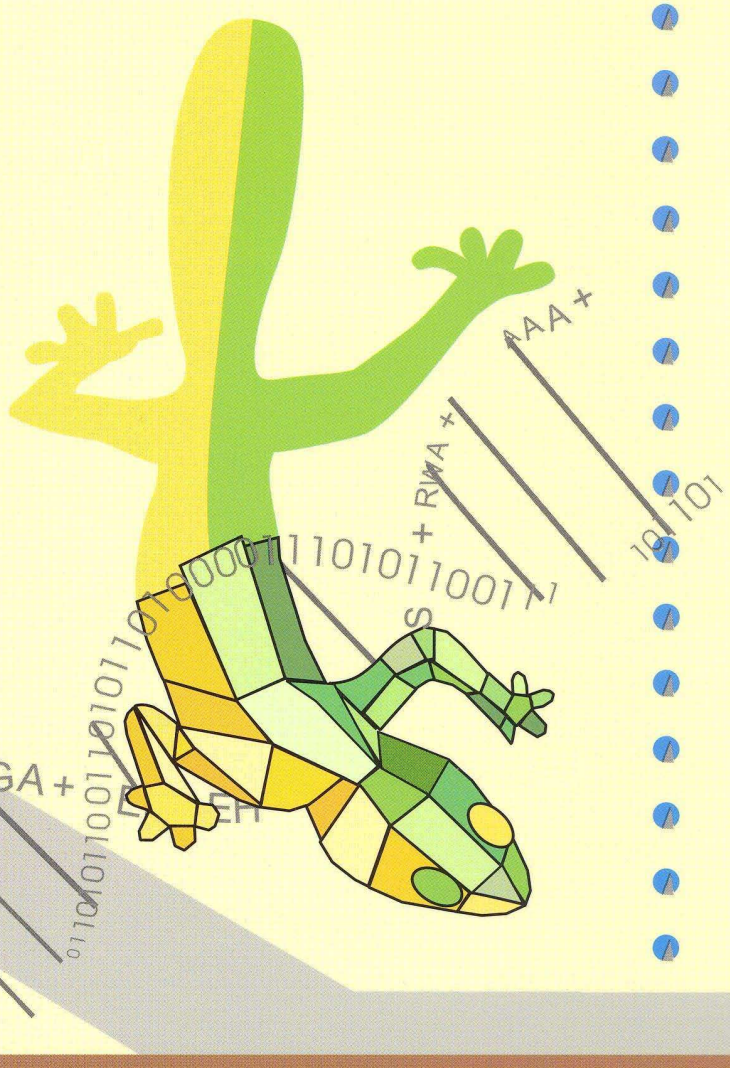
GECCO = GP + GA + ES + EP + EH + ER + DNA + LCS + RWA +

GECCO 2005

01001010101
011011101110
011011011010
100110011010 1
100110011110 1
100100101101 1

Genetic and Evolutionary Computation COnference

June 25-29, 2005
(Saturday - Wednesday)
Washington, D.C. USA



Sponsored by

ACM SIGEVO

Digital Circuit Design Using Dynamic Fitness Functions

Cecília Reis
Polytechnic Institute of Porto
Porto
351-228340500
cecilia@dee.isep.ipp.pt

J. A. Tenreiro Machado
Polytechnic Institute of Porto
Porto
351-228340500
jtm@dee.isep.ipp.pt

J. Boaventura Cunha
Univ. of Trás-os-Montes Alto Douro,
Vila Real
351-259350339
jboavent@utad.pt

ABSTRACT
This paper proposes and analyses the performance of a Genetic Algorithm using two new concepts, namely a static fitness function including a discontinuity measure and a fractional-order dynamic fitness function, for the synthesis of combinational logic circuits. In both cases, experiments reveal superior results in terms of speed and convergence to achieve a solution.

Categories and Subject Descriptors
B.6.3 [**Logic Design**]: Design Aids – *automatic synthesis, optimization.*

General Terms
Algorithms, Design.

Keywords
Fractional calculus, Genetic algorithms and Logic circuit design.

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) [25]. 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 [18].

One decade ago Sushil and Rawlins [8] applied GAs to the combinational circuit design problem. They combined knowledge-based systems with the GA and defined a genetic operator called masked crossover. This scheme leads to other kinds of offspring that can not be achieved by classical crossover operators.

John Koza [7] adopted genetic programming to design combinational circuits. His goal was the design of functional circuits through AND, OR and NOT logic gates.

Coello, Christiansen and Aguirre [2] presented a computer program that automatically generates high-quality circuit designs. They use five possible types of gates (AND, NOT, OR, XOR and WIRE) with the objective of finding a functional design that minimizes the use of gates other than WIRE (essentially a logical no-operation).

Miller, Thompson and Fogarty [10] applied evolutionary algorithms for the design of arithmetic circuits. The technique deals with 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.

Kalganova, Miller and Lipnitskaya [5] proposed a new technique for designing multiple-valued circuits. The EH is easily adapted to the distinct types of multiple-valued gates, associated with operations corresponding to different types of algebra, and can include other logical expressions. This approach is an extension of the EH method for binary logic circuits proposed in [10].

In order to solve complex systems, Torresen [19] proposed the method of increased complexity evolution. The idea is to evolve a system gradually as a kind of divide-and-conquer method. Evolution is first undertaken individually on a large number of simple cells. The evolved functions are the basic blocks adopted in further evolution or assembly of a larger and more complex system.

More recently, Hollingworth, Smith and Tyrrell [4] describe 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.

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 [21]. A possible method to solve this problem is to use building blocks rather than simple gates [13]. Nevertheless, this technique leads to another difficulty, which is how to define building blocks that are suitable for evolution.

Timothy Gordon [3] suggests an approach that allows evolution to search for good inductive bases for solving large-scale complex problems. This scheme generates, inherently, modular and iterative structures, that exist in many real-world circuit designs but, at the same time, allows evolution to search innovative areas of space.

The idea of using memory to achieve better fitness function performances was first introduced by Sano and Kita [23-24]. Their goal was the optimization of systems with randomly fluctuating fitness function and they developed a Genetic Algorithm with Memory-based Fitness Evaluation (MFEGA).

GECCO'05, June 25-29, 2005, Washington, DC, USA.

ACM 1-59593-010-8/05/0006.

The key ideas of the MFEGA are based on storing the sampled fitness values into memory as a search history, introducing a simple stochastic model of fitness values to be able to estimate fitness values of points of interest using the history for selection operation of the GA.

Bearing these ideas in mind, and looking for better performance GAs, this paper proposes a GA for the design of combinational logic circuits using fractional-order dynamic fitness functions.

The area of Fractional Calculus (FC) deals with the operators of integration and differentiation to an arbitrary (including noninteger) order and is as old as the theory of classical differential calculus [11, 14]. The theory of FC is a well-adapted tool to the modelling of many physical phenomena, allowing the description to take into account some peculiarities that classical integer-order models simply neglect. Nevertheless, the application of FC has been scarce until recently, but the advances on the theory of chaos motivated a renewed interest in this field. In the last two decades we can mention research on viscoelasticity/damping, chaos/fractals, biology, signal processing, system identification, diffusion and wave propagation, electromagnetism and automatic control [1, 6, 9, 15, 17, 20, 22].

The article is organized as follows. Section 2 describes the adopted GA as well as the fractional-order dynamic fitness functions. Section 3 presents the simulation results and finally, section 4 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 the static and dynamic fitness functions.

2.1 Problem Definition

We are using GAs to design combinational logic circuits. A truth table specifies the circuits 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, being Gset a the simplest one (i.e., a RISC-like set) and Gset b a more complex gate set (i.e., a CISC-like set). Logic gate named WIRE means a logical no-operation.

Table 1

Gate Set	Logic gates
Gset a	{AND,XOR,WIRE}
Gset b	{AND,OR,XOR,NOT,WIRE}

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 reproduce. 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 circuits are encoded as a rectangular matrix A (*row* × *column* = $r \times c$) of logic cells as represented in 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 form the chromosome. For example, the chromosome that represents a 3 × 3 matrix is depicted in figure 1.

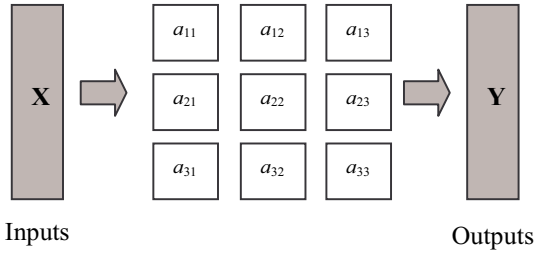


Figure 1. A 3 × 3 matrix A representing a circuit with input X and output Y.

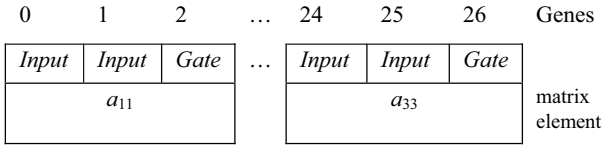


Figure 2. Chromosome for the 3 × 3 matrix of figure 1.

2.3 The Genetic Operators

The initial population of circuits (strings) has a random generation. 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 based on 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. In order to maintain the crossover integrity, the crossover point must be between cells.

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 reach the solution.

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

2.4 The Static and Dynamic Fitness Functions

The goal of this study is to find new ways of evaluating the individuals of the population in order to achieve better performance GAs.

In this paper we propose two concepts for the fitness functions, namely the static fitness function F_s and the dynamic fitness function F_d.

The calculation of F_s in (1) has two parts, f_1 and f_2, where f_1 measures the functionality and the error discontinuity 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 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. After this, f_{11} is decremented by $\delta \in [0, 1]$ for each $\mathbf{Y_R} - \mathbf{Y}$ error discontinuity, where discontinuity means passing from $\mathbf{Y_R} - \mathbf{Y} = 0$ to $\mathbf{Y_R} - \mathbf{Y} = 1$, or vice-versa when comparing two consecutive levels of the truth table. 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> ≡ <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:

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

$$f_{11} = f_{11} + 1 \text{ if } \{\text{bit } i \text{ of } \mathbf{Y}\} = \{\text{bit } i \text{ of } \mathbf{Y_R}\},\ i = 1, \ldots, f_{10} \quad \textbf{(1b)}$$

$$f_1 = f_{11} - \delta \text{ if } \text{error}_i \neq \text{error}_{i-1},\ i = 1, \ldots, f_{10} \quad \textbf{(1c)}$$

$$f_2 = f_2 + 1 \text{ if } \textit{gate type} = \textit{wire} \quad \textbf{(1d)}$$

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

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

The dynamic fitness function F_d concept arises from an analogy between Gas and control systems. In the GA case, the goal is to control the population through the fitness function. In this line of thought, the static fitness function corresponds to a kind of proportional algorithm while a dynamic one may be implemented by:

$$F_d = F_s + K\, D^{\alpha}[F_s] \quad \textbf{(2)}$$

where $-1 \leq \alpha \leq 1$ is the differential (integral) fractional-order for positive (negative) values of α and K is the 'gain' of the dynamical term.

The generalization of the concept of derivative $D^{\alpha}[f(x)]$ to noninteger values of α goes back to the beginning of the theory of differential calculus. In fact, Leibniz, in his correspondence with Bernoulli, L'Hôpital and Wallis, had several notes about its calculation for α = 1/2 [11, 14]. Nevertheless, the adoption of the *FC* in control algorithms has recent studies using the frequency and discrete-time domains [6, 9, 15, 17].

The mathematical definition of a derivative of fractional order α has been the subject of several different approaches. For example, Eq. (3) and Eq. (4), represent the Laplace (for zero initial conditions) and the Grünwald-Letnikov definitions of the fractional derivative of order α of the signal $x(t)$:

$$D^{\alpha}[x(t)] = L^{-1}\{s^{\alpha} X(s)\} \quad (3)$$

$$D^{\alpha}[x(t)] = \lim_{h \to 0} \frac{1}{h^{\alpha}} \sum_{k=0}^{\infty} \frac{(-1)^k \Gamma(\alpha + 1)}{k!\,\Gamma(\alpha - k + 1)} x(t - kh) \quad (4)$$

where Γ is the gamma function and h is the time increment. This formulation [17] inspired a discrete-time calculation algorithm, based on the approximation of the time increment h through the sampling period T and an r-term truncated series yielding the equation:

$$D^{\alpha}[x(t)] \approx \frac{1}{T^{\alpha}} \sum_{k=0}^{r} \frac{(-1)^k \Gamma(\alpha + 1)}{k!\,\Gamma(\alpha - k + 1)} x(t - kT) \quad \textbf{(5)}$$

3. EXPERIMENTS AND RESULTS

Reliable execution and analysis of a GA usually requires a large number of simulations to provide a reasonable assurance that the stochastic effects are properly considered [12]. Therefore, in this study are developed n = 1000 simulations for each case under analysis.

The experiments consist on running the GA 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 schemes described previously. The circuits are generated with the gate sets presented in Table 1 for *CR* = 95%, *MR* = 20% and *P* = 100.

Having a superior GA performance means achieving solutions with a smaller number *N* of generations and a smaller standard deviation in order to reduce the stochastic nature of the algorithm.

3.1 Using the Static Fitness Function

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

Figures 3 and 4 show the average number of generations to achieve the solution *AV*(*N*) and the corresponding standard deviation *SD*(*N*) versus the discontinuity factor δ = {0, 0.25, 0.5, 0.75, 1}, using Gset a and Gset b, for the *M2-1* and the *PC4* circuits, respectively.

The results reveal that, as it was expected from previous studies [16], the RISC-like set Gset a presents better performance than the CISC-like gate set Gset b for all values of δ. On the other hand, analysing the influence of δ we conclude that the GA response is best mostly in the region around δ = 0.5 for the two circuits and for the two gate sets.

3.2 Using the Dynamic Fitness Function

In this sub-section, we analyze the GA performance when we adopt a dynamic scheme for the fitness function.

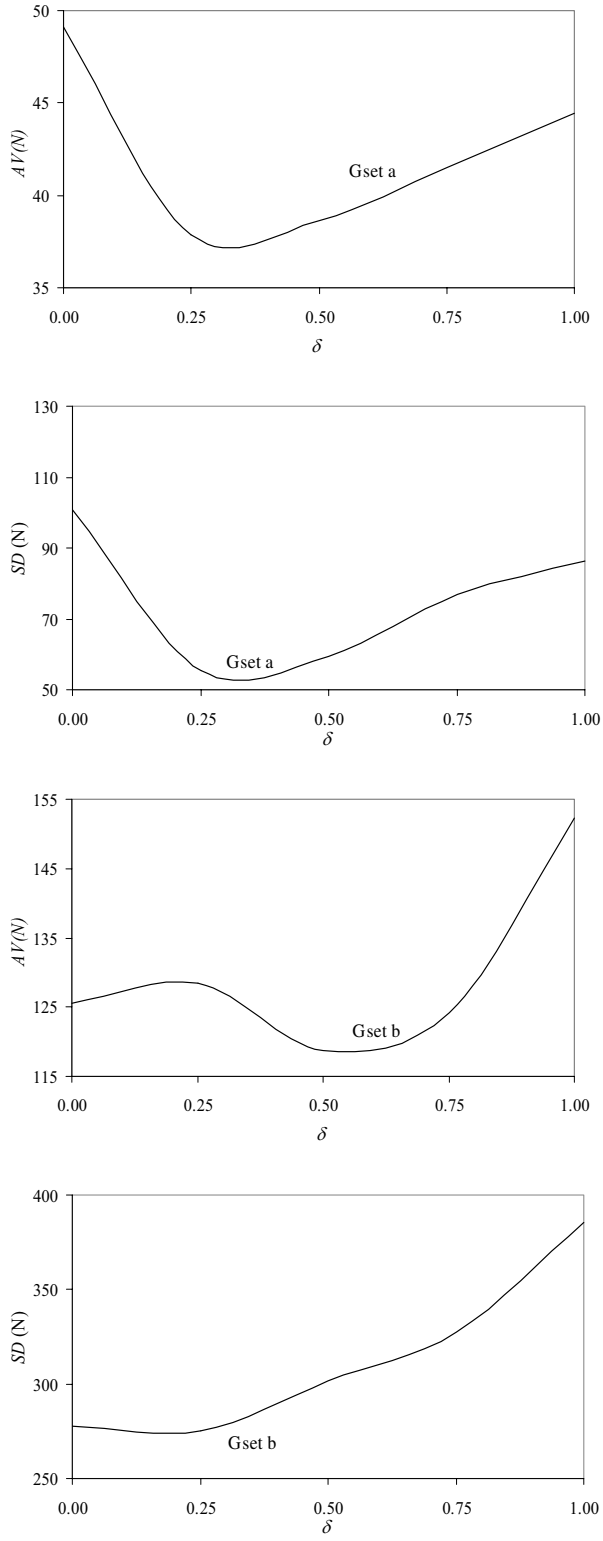


Figure 3. M2-1 average number of generations to achieve a solution $AV(N)$ and standard deviation $SD(N)$ for δ= {0, 0.25, 0.5, 0.75, 1} with Gsets a and b

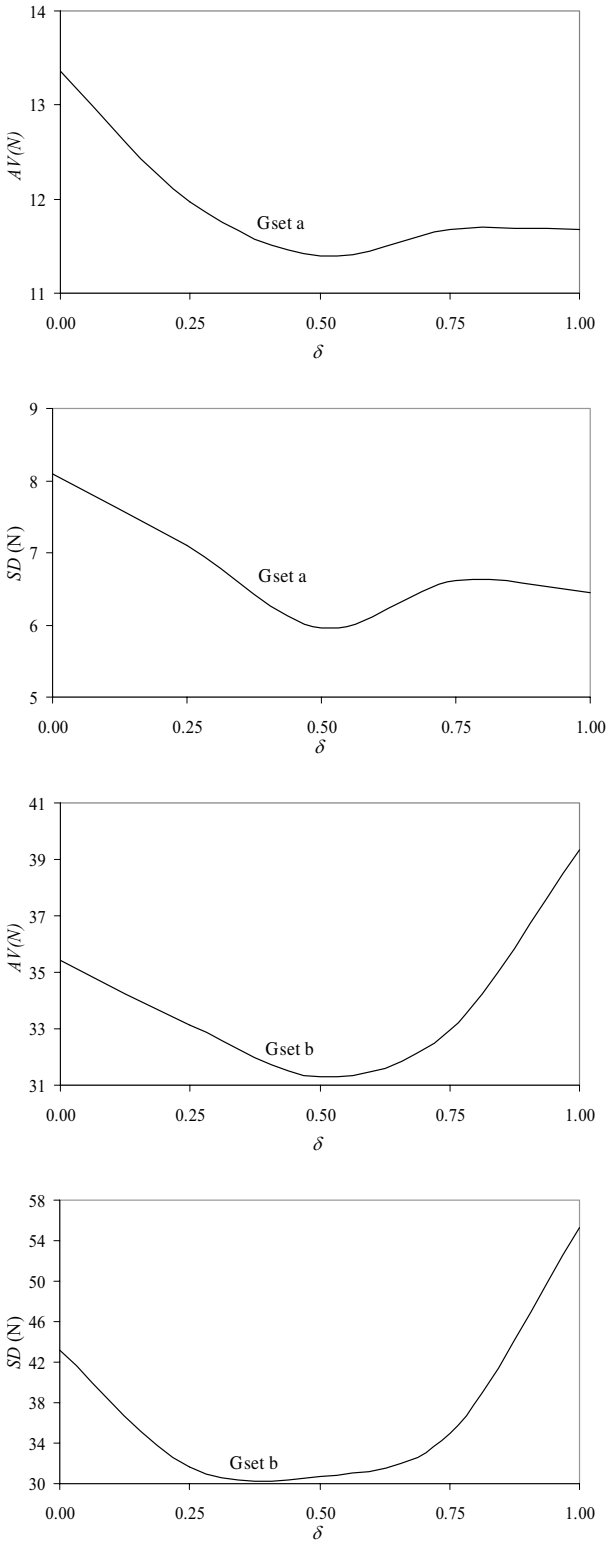
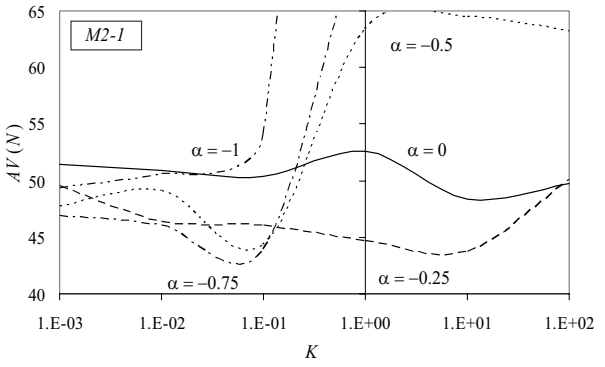
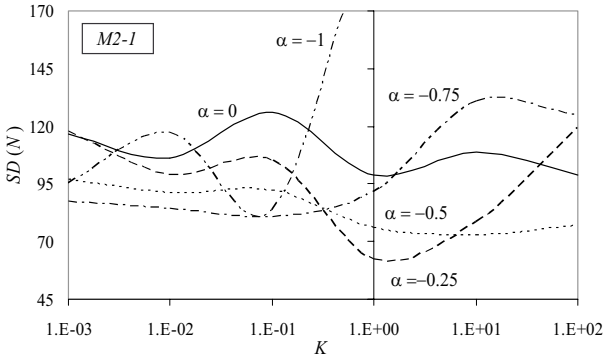
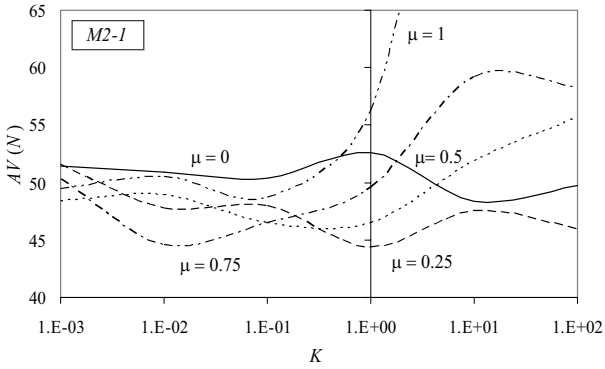


Figure 4. *PC4* average number of generations to achieve a solution $AV(N)$ and standard deviation $SD(N)$ for δ = {0, 0.25, 0.5, 0.75, 1} with Gsets a and b







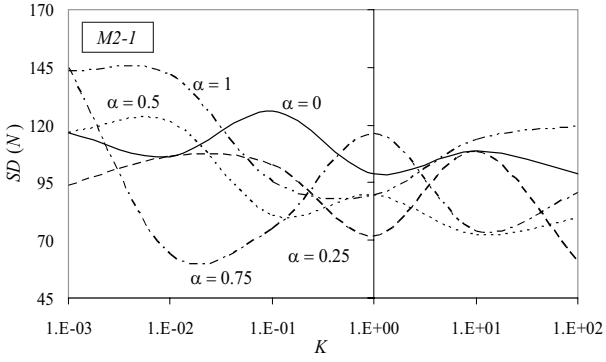
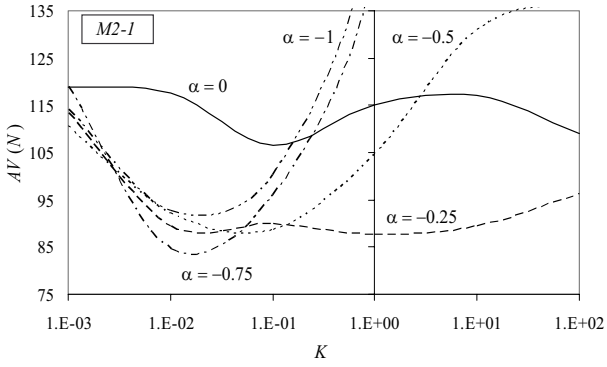
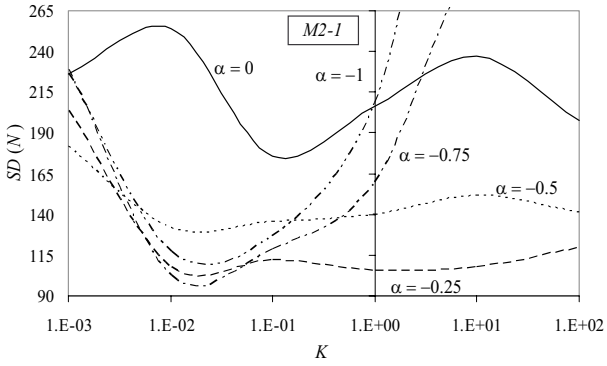
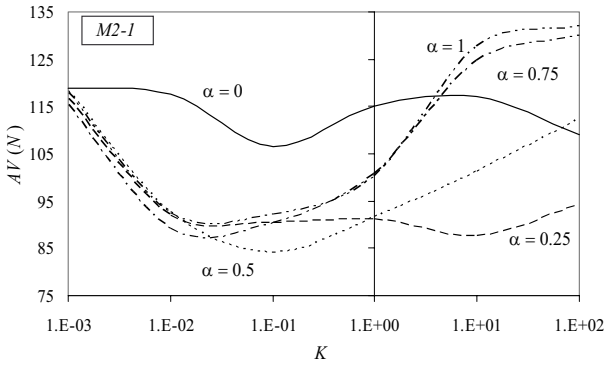


Figure 5. ***M2-1*** **average number of generations to achieve a solution** ***AV*****(*****N*****) and standard deviation** ***SD*****(*****N*****) for the integral and the differential schemes with Gset a**







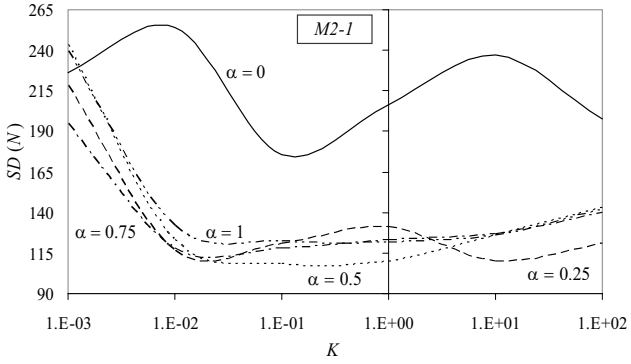


Figure 6. ***M2-1*** **average number of generations to achieve a solution** ***AV*****(*****N*****) and standard deviation** ***SD*****(*****N*****) for the integral and the differential schemes with Gset b**

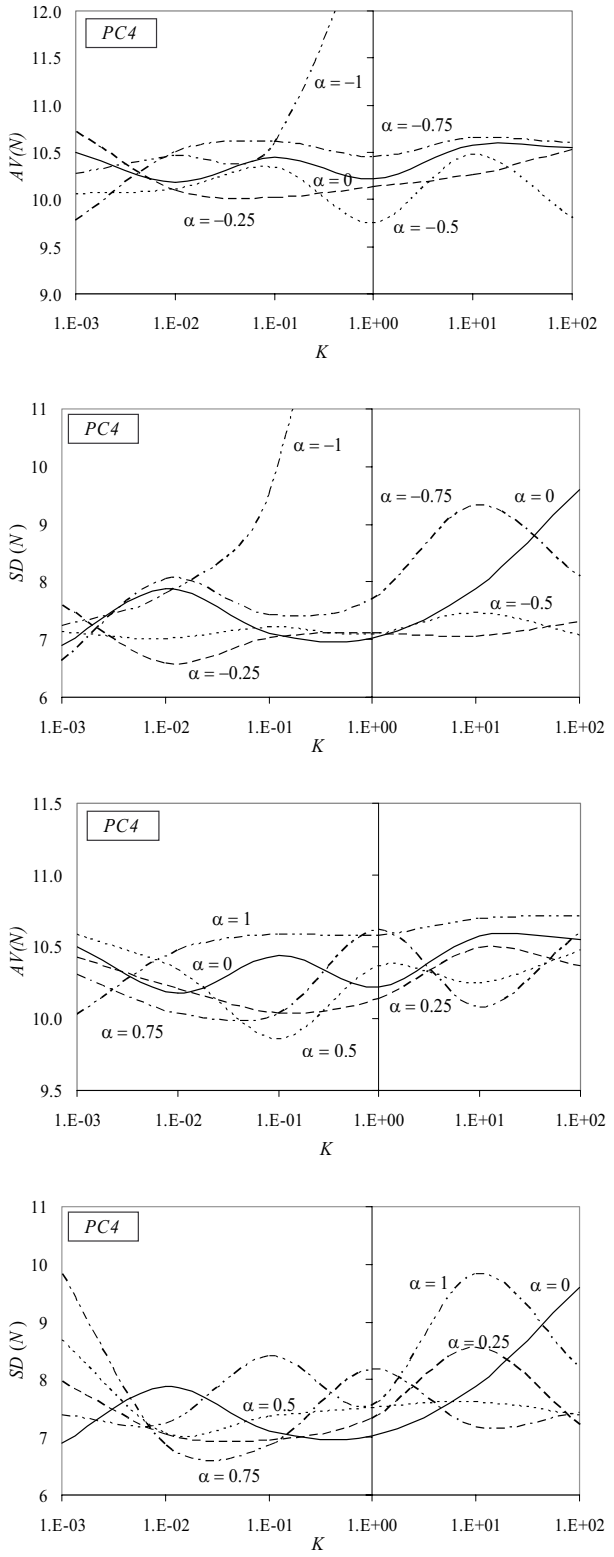


Figure 7. *PC4* average number of generations to achieve a solution $AV(N)$ and standard deviation $SD(N)$ for the integral and the differential schemes with Gset a

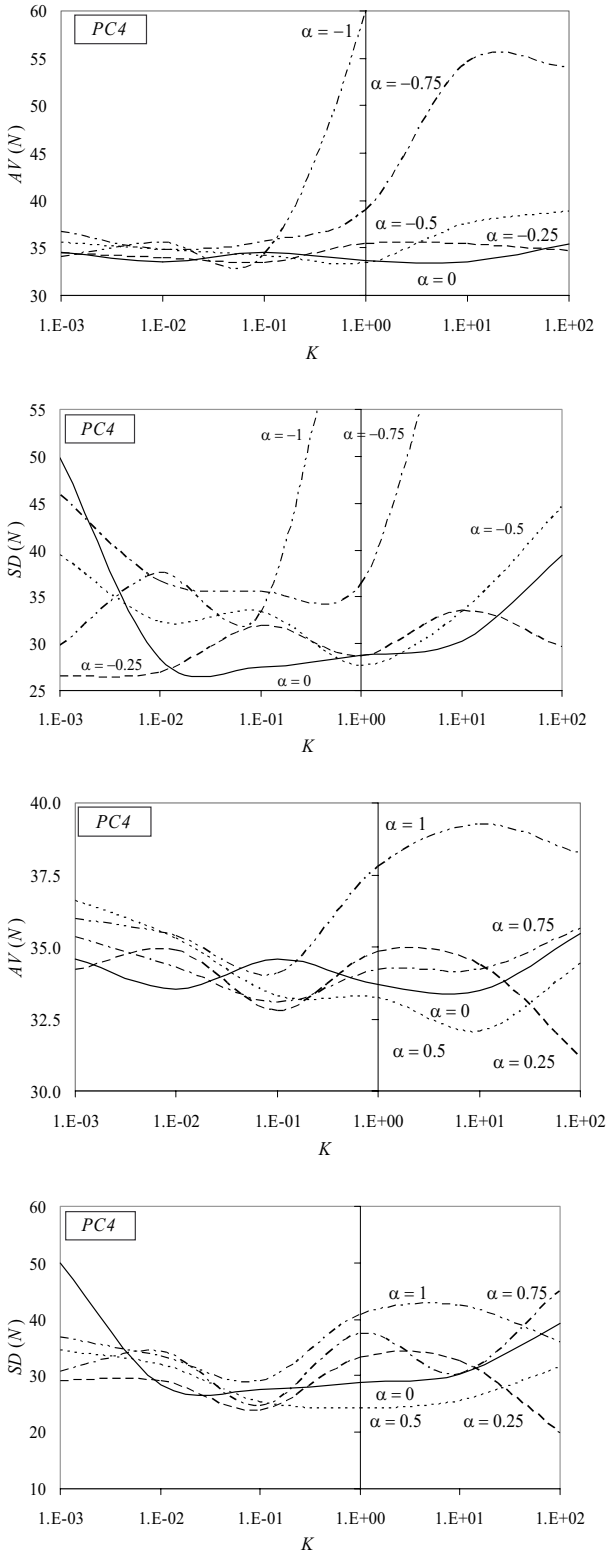


Figure 8. *PC4* average number of generations to achieve a solution $AV(N)$ and standard deviation $SD(N)$ for the integral and the differential schemes with Gset b

The simulations investigate an integral scheme ($\alpha = \{-1, -0.75, -0.5, -0.25, 0\}$) and a differential scheme ($\alpha = \{0, 0.25, 0.5, 0.75, 1\}$) in F_d for gains $10^{-3} \leq K \leq 10^{2}$. The implementation of the integral/differential fractional order operator adopts Eq. (5) with a series truncation $r = 50$ terms.

Figures 5 to 8 show the average number of generations to achieve a solution $AV(N)$ and the standard deviation $SD(N)$ for the integral and the differential schemes, for the *M2-1* and the *PC4* circuits, using Gset a and Gset b.

Tables 2 and 3 present the pair of parameters (α, K) for each best solution obtained in terms of average number of generations $AV(N)$ and in terms of standard deviation $SD(N)$, respectively, for the integral and the differential schemes of F_d.

Table 2. The (α, K) parameters for each best solution obtained in terms of $AV(N)$

Circuit	Gset a	Gset b
M2-1	(α, K) = (0.25, 1)	(α, K) = (0.5, 0.1)
	(α, K) = (0.25, 10)	(α, K) = (–0.75, 0.01)
PC4	(α, K) = (0.5, 0.1)	(α, K) = (0.25, 100)
	(α, K) = (0.5, 1)	(α, K) = (–0.25, 0.1)

Table 3. The (α, K) parameters for each best solution obtained in terms of $SD(N)$

Circuit	Gset a	Gset b
M2-1	(α, K) = (0.25, 100)	(α, K) = (0.5, 0.1)
	(α, K) = (–0.25, 1)	(α, K) = (–0.75, 0.01)
PC4	(α, K) = (0.75, 0.1)	(α, K) = (0.25, 100)
	(α, K) = (–0.25, 0.01)	(α, K) = (–0.25, 0.001)

In general, we conclude that the F_d concept produces better results particularly for the differential scheme. Moreover, once again, the RISC-like gate set is superior to the CISC-like gate set and the best results occur for fractional order α.

4. CONCLUSIONS

This paper presented a technique for improving the GA performance. In what concerns with the classical static fitness function we conclude that it is possible to get superior results by measuring the error discontinuity. On the other hand, the new concept of fractional-order dynamic fitness function, demonstrates to be an important method that outperforms the traditional static fitness function approach. In both cases, the tuning of the 'optimal' parameters δ or (α, K) was established by trial and error. Therefore, future research will address the problem of having a more systematic design method.

These conclusions encourage further studies that explore deeper the two proposed concepts.

5. REFERENCES

[1] Chen, Y. Q. and Moore, K. L., Discretization schemes for fractional-order differentiators and integrators. *IEEE Trans. On Circuits and Systems*, vol. 49, pp 363-367, March 2002.

[2] 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.

[3] 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.

[4] 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.

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

[6] Koh, C. G. and Kelly, J. M. Application of fractional derivatives to seismic analysis of base-isolated models. *Earthquake Engineering and Structural Dynamics*, vol.19, pp.229-241, 1990.

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

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

[9] Méhauté, A. L.. *Fractal Geometries: Theory and Applications*. Penton Press, London, 1991.

[10] 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.

[11] Miller, K. S. and Ross, B. *An Introduction to the Fractional Calculus and Fractional Differential Equations*. John Wiley & Sons, New York, 1993.

[12] Morrison, R. W. Dispersion-Based Population Initialization. In *Proceedings of the Genetic and Evolutionary Computation Conference* – GECCO 2003, pp 1210-1221.

[13] Murakawa, M., Yoshizawa, S., Kajitani, I., Furuya, T., Iwata, M., Higuchi, T. Hardware Evolution at Function Level. In *Proceedings of The 4th International Conference on Parallel Problem Solving from Nature*, Berlin, Germany, September 22-26, 1996.

[14] Oldham, K. B. and Spanier, J. *The Fractional Calculus: Theory and Application of Differentiation and Integration to Arbitrary Order*. Academic Press, New York, 1974.

[15] Oustaloup, A. *La Dérivation Non Entier: Théorie, Synthèse et Applications*. Editions Hermes, 1995.

[16] Reis, Cecília, Tenreiro Machado, J. A. and Boaventura Cunha, J. 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.

[17] Tenreiro Machado, J. A. Analysis and Design of Fractional-Order Digital Control Systems. *SAMS Journal Systems Analysis, Modelling, Simulation*, vol. 27: 107-122, 1997.

[18] Thompson, A. and Layzell, P. Analysis of unconventional evolved electronics. *Communications of the ACM*, vol. 42, 1999, pp. 71-79.

[19] 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.

[20] Torvik, P. J. and Bagley, R. L. On the Appearance of the Fractional Derivative in the Behaviour of real materials. *ASME Journal of Applied Mechanics*, vol. 51, pp. 294-298, June 1984.

[21] 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.

[22] Westerlund, S. *Dead Matter Has Memory!* Causal Consulting. Sweden: Kalmar, 2002.

[23] Y. Sano and H. Kita. Optimization of Noisy Fitness Functions by means of Genetic Algorithms using History of Search. In *Proceedings of PPSN VI*, pp. 571-581, 2000.

[24] Y. Sano and H. Kita. Optimization of Noisy Fitness Functions by means of Genetic Algorithms using History of Search with test of Estimation. In *Proceedings of CEC*, 2002.

[25] 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.