Evolutionary Systems for Electronic Engineers

RUSTEM POPA, MIRCEA ILIEV, VIOREL NICOLAU
"Dunărea de Jos" University of Galati,
Automatic Control and Electronics Department

Key words: evolutionary computation, genetic algorithms, evolvable hardware, electronic circuits

Abstract. Evolutionary systems are designed by the means of evolutionary computation. These "designs" are evolved by a process of natural selection, like in the living matter. The mechanism of evolution is entirely blind and has no particular object other than survivability. The survivability of the organism can be seen as a process of assembling a larger system from a number of component parts and then testing the organism in the environment in which it finds itself. The concept of *assemble-and-test* together with an evolutionary algorithm can explore the entire design space because of the absence of imposed rules of design. In this way, in electronics, evolutionary design becomes in fact an Invention Machine that generates new unexpected and usually useful electronic circuits. We have prepared some programs with Graphic User Interfaces (GUIs) in Matlab5.3 for evolutionary experiments with digital and analog electronic circuits. All these programs have been used with our students in the laboratory of Evolutionary Systems.

Introduction

There is a great difference between the way in which physical systems have been designed by blind evolution and the methods employed by human designers. In the former, entire systems are constructed and tested in situ without a conscious application of principles. In the latter, systems are developed by a process of human creation, which employs a huge collection of rules, concepts and principles. It is indeed curious that organisms such as ourselves who are capable of imagining a world that operates according to definite laws and abstract design process were themselves produced by a mechanism that is entirely blind and has no particular object other than survivability [1].

Today we can find evolutionary systems in the field of visual arts, music, architecture, designs, and, of course, in the field of electronic circuit design. The building of new electronic circuits by evolutionary computation has been created the concept of Evolvable Hardware (EHW). Research in EHW can be divided into *intrinsic evolution*, when each new circuit is built in electronic hardware and tested, and *extrinsic evolution*, that uses a model of the hardware and evaluates the circuit by simulation in software.

We are convinced that very soon, as reprogrammable integrated circuits will become larger and larger, EHW will be dominant in electronics, and the electronic engineer must be ready for this future evolution. We have prepared some laboratories of extrinsic evolution of digital and analog circuits, and for solving of some wellknown optimization problems, like the generation of test vectors in digital circuits, the finding of the global minima in a

multimodal function, or the solving of the Traveling Salesman Problem (TSP). We have also prepared some laboratories of intrinsec evolvable hardware of digital circuits by using common digital CMOS circuits, but these implementations are not presented in this paper.

A brief introduction in genetic algorithms

Almost all our implementations in this paper are based on Genetic Algorithms (GAs), an adaptive searching technique for solving optimisation problems based on the mechanics of natural genetics and natural selection. The success of the application of GAs to an optimisation problem depends on the representation of chromosomes, fitness function, method of crossover, mutation operation, and on the diverse information from the chromosomes. When the diversity is lost before the global optimum solution is found, the performance of GAs deteriorates and their solution processes converge prematurely. Moreover, the mutation operation is important. While the mutation operation adds new information to a chromosome, it can also destroy useful information held in the chromosome.

In GAs the search is conducted using information of a population of candidate solutions, called chromosomes, so that the chance of the search being settled in a local optimum can be significantly reduced. Four essential components need to be designed in applying a GA for an optimisation problem: chromosomes representation, crossover operator, mutation operator and fitness function.

Each chromosome is represented by a binary string of finite length. The function of the crossover operator is to generate new child chromosomes from two selected parents chromosomes. We have used uniform crossover with a high rate, even if the computation effort in exploring unpromising regions of the solution space may be higher. Mutation operator changes the value of one bit in the chromosome, with some probability. If the frequency of the mutation operation is too high, valuable old information may be destroyed in the chromosomes. On the other hand, if the frequency is low, the chromosomes remain virtually unchanged and little information will be added to assist the search in GA. After all, the fitness function can be defined in terms of the objective function of the optimisation problem. The search for the global optimum solution is then equivalent to finding the chromosomes having the maximum fitness.

The structure of a Standard Genetic Algorithm (SGA) is as follows:

begin
- generate randomly the initial population of chromosomes;
repeat
- calculate the fitness of chromosomes in current generation;
repeat
- select proportional with fitness, in a stochastic manner, 2 chromosomes as parents;
- apply crossover to the selected parents to obtain 2 child chromosomes;
- calculate the fitness of chromosomes;

until end of the number of new chromosomes
- apply mutation to the new chromosomes;
- update the population, in accordance with the fitness of each chromosome;

until end of the number of generations
end

Like mutation, crossover may be applied with some probability to reduce the computation effort, and the number of generations can be conveniently expressed as number of iterations.

Cellular automata, multimodal functions and TSP

Cellular Automata (CA) are a scheme for computing using local rules and local communication. Typically a CA is defined on a grid, with each point on the grid representing a cell with a finite number of states. A transition rule is applied to each cell simultaneously. Typical transition rules depend on the state of the cell and its (4 or 8) nearest neighbors, although other neighborhoods are used. We have used four different types of CA: Conway's life, excitable media, forest fire and diffusion limited aggregation. The last one system simulates sticky particles aggregating to form a fractal structure, like in figure 1. These particles are assumed to be bouncing around in some dense (but invisible) liquid. The effect is to randomize the direction of motion of every particle at every time step. Put differently, every time-step is a collision step. The simulation is also seeded with one fixed particle in the center of the array. Any diffusing particle which touches it sticks to it, and itself becomes a non-moving, sticky particle.

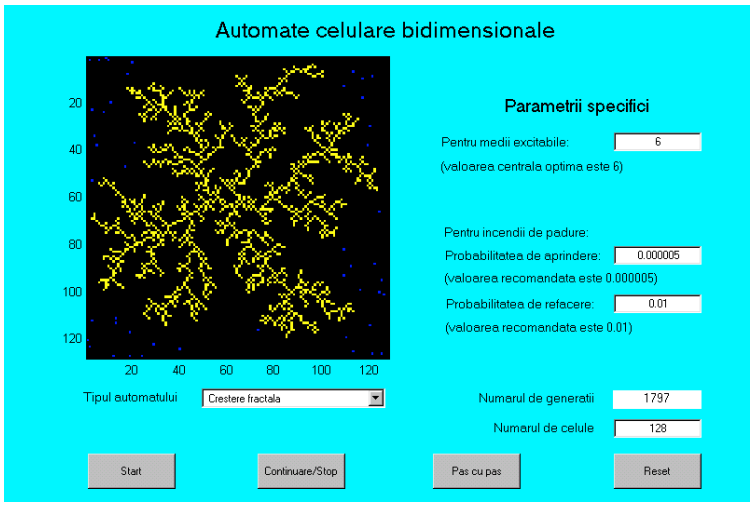

Fig. 1 A GUI for experiments with some types of bidimensional cellular automata

Multimodal Functions (MF) are frequently used to compare the efficiency of different search algorithms. We have implemented a GA for minima search in a set of eight most known unidimensional and bidimensional MF (Rastrigin function in the figure 2).

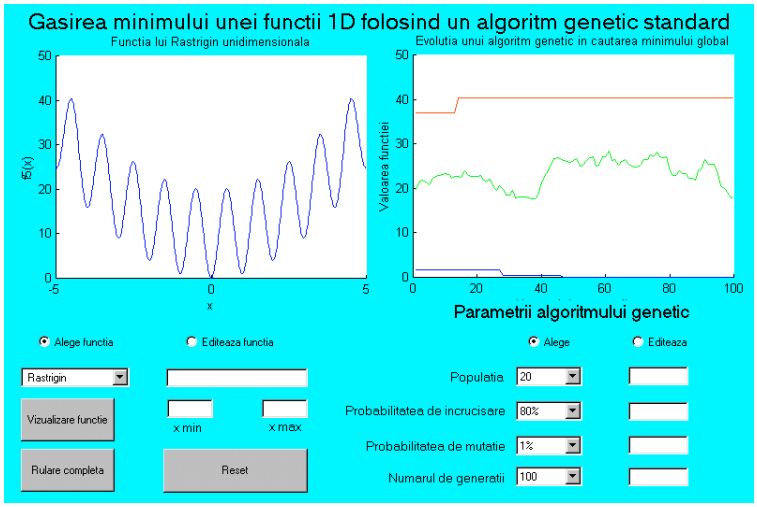

Fig. 2 A GUI for experiments with some multimodal test functions

We can see that the global minima of the function presented in the figure 2 is $f(0) = 0$, and this result is achieved after almost 50 generations (the lower line in the figure 2). The GA has a population of 20 chromosomes, the crossover rate is 80% and the mutation rate is 1%. The fitness of the worse chromosomes decreases in the process of evolution (the upper line in the figure 2), because these chromosomes are not eliminated with each new generation. If we decide to exclude them, then the fitness of the whole population increases in time.

TSP is one of the most interesting problems in optimization. The traveling salesman must visit every city in his territory exactly once and then return to the starting point. The goal is the minimization of the cost of travel between all the cities, that is the total distance between the cities. We have presented two distinct methods for solving this problem. In the former, we have used a heuristic approach which solves the problem very quickly with an acceptable result, even for a great number of cities (200 for example). In the latter, we have used a GA with an adequate representation of chromosomes, the final result is probably better, but the time of the searching increases substantially. The GUI from the figure 3 presents a solution with GA for 30 cities placed randomly in the left. In the right we can see the evolution of the fitness (the distance between cities) in 200 generations, but the minimum distance is achieved after less than 150 generations. The GA has a population of 100 chromosomes, the crossover rate is 80% and the mutation rate is 1%. The number of replaced chromosomes with each new generation is sufficiently high: 40 from the entire population of 100 chromosomes.

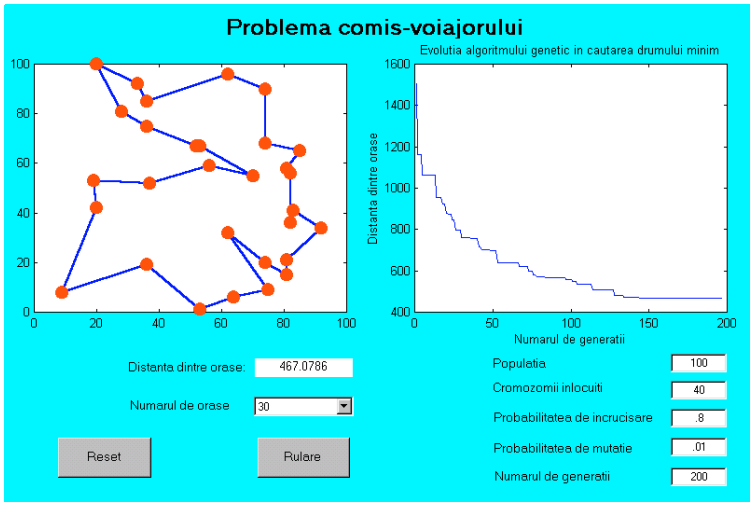

Fig. 3 A GUI for the solving of the TSP problem

Evolutionary synthesis of digital circuits

Two laboratories have been prepared for the purpose of automated generation of test vectors in digital circuits. If we want to generate a test to detect a stuck-at 0 fault in the marked node of the circuit represented in the figure 4, the required vector is 1111111111000000000, a combination of bits nearly impossible to find using a random approach [4]. The GA used to solve this problem finds the correct solution in 40 generations. The algorithm uses a population of 32 chromosomes and a mutation rate of 3%. Fitness was calculated like the sum of (1 if fault is excited or 0 otherwise) + (fraction of inputs in AND gate set to 1) + (fraction of inputs in OR gate set to 0).

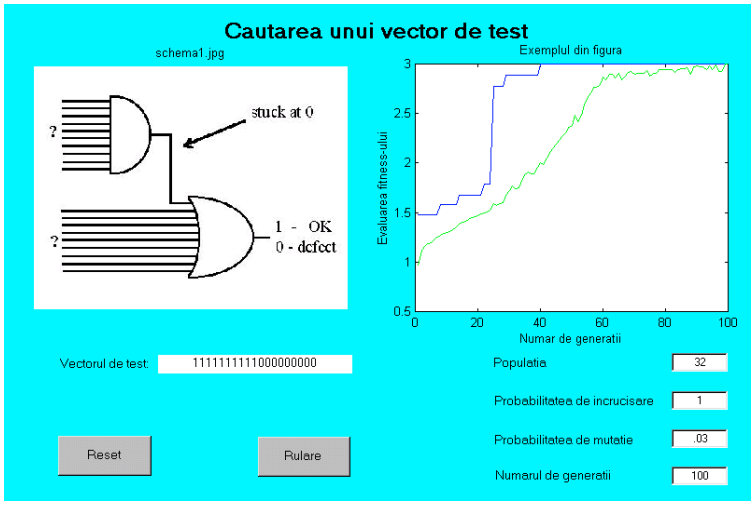

Fig. 4 A GUI for the generation of a test vector for a stuck-at 0 fault in the marked node

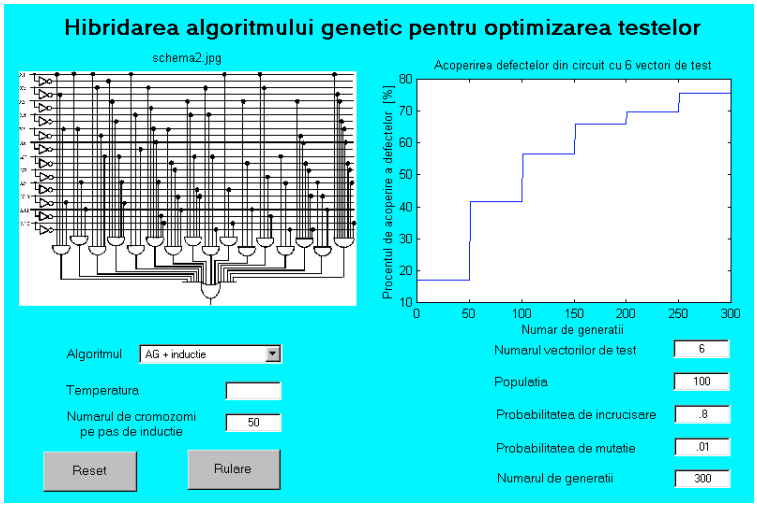

Fig. 5 A GUI for the generation of an optimum set of test vectors by hibridation of a GA

By using the GUI from the figure 5, we can solve the Fault Coverage Code Generation Problem for a more complex combinational logic. The problem consists in finding of a given number of test vectors that maximizes the fault coverage of the circuit. We have chosen two ways of hibridation of the SGA: by using the inductive search, like in the figure 5, or by using the simulated annealing algorithm. The example from the figure 5 shows that only 6 test vectors could cover more than 75% from the total number of stuck-at 0 faults in the circuit.

Figure 6 shows a method of coding of a digital circuit with 4 types of gates. Using this type of convention, we have achieved by evolution the circuit from the figure 7 for the function $f = x_3 \oplus \overline{x_1 \cdot x_2}$.

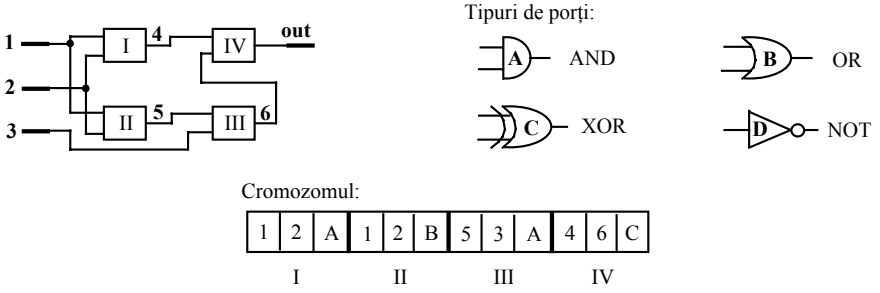

Fig. 6 An example of a digital circuit at gate level and the chromosome coding

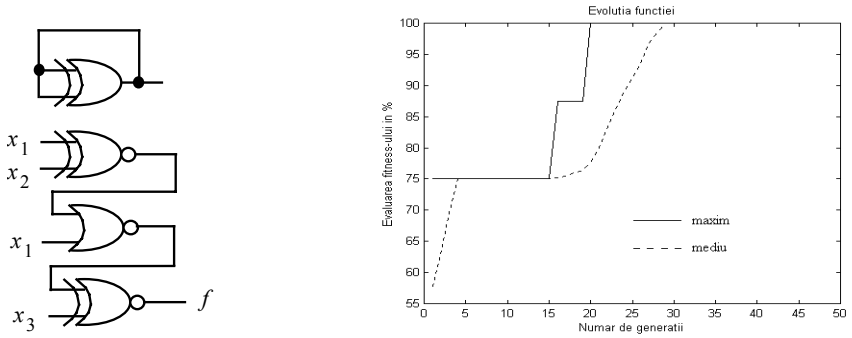

Fig. 7 The circuit achieved by evolution for the function $f = x_3 \oplus \overline{\overline{x}_1 \cdot x_2}$ *and the evolution of the fitness across 50 generations*

Evolutionary synthesis of analog circuits

Analog circuit synthesis entails the creation of both the topology and the sizing (numerical values) of all of the circuit's components. The difficulty of this problem is well known and the first auspicious approach, based on genetic programming, was presented in [2]. Another method of automatically generating analog circuit designs based on a parallel GA and a set of circuit constructing primitives is presented in [3].

Both methods need a huge computation power (few days on a parallel computer with 64 processors in the first case, or a network of workstations in the second case). Our last laboratory verifies through PSPICE simulation some of the circuits presented in [2] and [3]. Interesting is the fact that not all given circuits were successful in our simulations.

Conclusions

Evolutionary design is in fact a creative machine for new designs and may be useful for electronic engineers. The laboratories presented here display the generation of complexity with very simple rules, and the solving of complex NP-problems with simple GAs. GAs may be useful for automated generation of test vectors and for synthesis of digital circuits. Analog circuit synthesis needs more powerful computers, but in the near future this impediment will be probably avoided.

References

1. Bentley, P. J., Corne, D. W.: *Creative Evolutionary Systems*. Morgan Kaufmann Publishers, San Francisco, 2002.
2. Koza, J. R., Bennett III, F. H., Andre, D., Keane, M. A., Dunlap, F.: *Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming*. In: IEEE Transactions on Evolutionary Computation, vol. 1, nr. 2, July 1997, pp. 109-127.
3. Lohn, J. D., Colombano, S. P.: *A Circuit Representation Technique for Automated Circuit Design*. In: IEEE Transactions on Evolutionary Computation, vol. 3, nr. 3, September 1999, pp. 205-219.
4. Mazumder, P., Rudnick, E. M.: *Genetic Algorithms for VLSI Design, Layout & Test Automation*. Prentice Hall PTR, New Jersey, 1999.