6

Genetic Algorithms: An Overview with Applications in Evolvable Hardware

Popa Rustem
"Dunarea de Jos" University of Galati
Romania

1. Introduction

The genetic algorithm (GA) is an optimization and search technique based on the principles of genetics and natural selection. A GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the "fitness" (i.e., minimizes the cost function). The fundamental principle of natural selection as the main evolutionary principle has been formulated by Charles Darwin, without any knowledge about genetic mechanism. After many years of research, he assumed that parents qualities mix together in the offspring organism. Favorable variations are preserved, while the unfavorable are rejected. There are more individuals born than can survive, so there is a continuous struggle for life. Individuals with an advantage have a greater chance for survive i.e., the "survival of the fittest". This theory arose serious objections to its time, even after the discovering of the Mendel's laws, and only in 1920s "was it proved that Mendel's genetics and Darwin's theory of natural selection are in no way conflicting and that their happy marriage yields modern evolutionary theory" (Michalewicz, 1996).

The dynamical principles underlying Darwin's concept of evolution have been used to provide the basis for a new class of algorithms that are able to solve some difficult problems in computation. These "computational equivalents of natural selection, called evolutionary algorithms, act by successively improving a set or generation of candidate solutions to a given problem, using as a criterion how fit or adept they are at solving the problem." (Forbes, 2005). Evolutionary algorithms (EAs) are highly parallel, which makes solving these difficult problems more tractable, although usually the computation effort is huge.

In this chapter we focus on some applications of the GAs in Digital Electronic Design, using the concept of extrinsic Evolvable Hardware (EHW). But first of all, we present the genesis of the main research directions in Evolutionary Computation, the structure of a Simple Genetic Algorithm (SGA), and a classification of GAs, taking into account the state of the art in this field of research.

2. A brief history of evolutionary computation

In the 1950s and the 1960s several computer scientists independently studied evolutionary systems with the idea that evolution could be used as an optimization tool for engineering

problems. The idea in all these systems was to evolve a population of candidate solutions to a given problem, using operators inspired by natural genetic variation and natural selection.

In the 1960s, two german scientists, Ingo Rechenberg and Hans-Paul Schwefel introduced "evolution strategies", a method they used to optimize real−valued parameters for the shape of airplane wings. The field of evolution strategies “has remained an active area of research, mostly developing independently from the field of genetic algorithms (although recently the two communities have begun to interact)” (Mitchell, 1997). Around the same time, completely independently, american scientist Lawrence Fogel developed a method of computational problem solving he termed “evolutionary programming”, a technique in which candidate solutions to given tasks were represented as finite−state machines, which were evolved by randomly mutating their state−transition diagrams and selecting the fittest (Forbes, 2005).

Genetic algorithms (GAs) “were invented by John Holland in the 1960s and were developed by Holland and his students and colleagues at the University of Michigan in the 1960s and the 1970s. In contrast with evolution strategies and evolutionary programming, Holland's original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems. Holland's 1975 book *Adaptation in Natural and Artificial Systems* presented the genetic algorithm as an abstraction of biological evolution and gave a theoretical framework for adaptation under the GA. Holland's GA is a method for moving from one population of "chromosomes" (e.g., strings of ones and zeros, or "bits") to a new population by using a kind of "natural selection" together with the genetics−inspired operators of crossover, mutation, and inversion. (…) Holland's introduction of a population−based algorithm with crossover, inversion, and mutation was a major innovation” (Mitchell, 1997). Rechenberg's evolution strategies generate a single offspring, which is a mutated version of the parent.

“Holland was the first to attempt to put computational evolution on a firm theoretical footing. Until recently this theoretical foundation, based on the notion of "schemas," was the basis of almost all subsequent theoretical work on genetic algorithms. In the last several years there has been widespread interaction among researchers studying various evolutionary computation methods, and the boundaries between GAs, evolution strategies, evolutionary programming, and other evolutionary approaches have broken down to some extent. Today, researchers often use the term "genetic algorithm" to describe something very far from Holland's original conception” (Mitchell, 1997).

Current techniques are more sophisticated and combine the basic algorithms with other heuristics. Koza developed in 1992 “genetic programming”, which applies a GA to writing computer programs. “The variables are various programming constructs, and the output is a measure of how well the program achieves its objectives. The GA operations of mutation, reproduction (crossover) and cost calculation require only minor modifications. GP is a more complicated procedure because it must work with the variable length structure of the program or function. A GP is a computer program that writes other computer programs” (Haupt & Haupt, 2004). “Genetic Programming uses evolution-inspired techniques to produce not just the fittest *solution* to a problem, but an *entire optimized computer program*.” (Forbes, 2005).

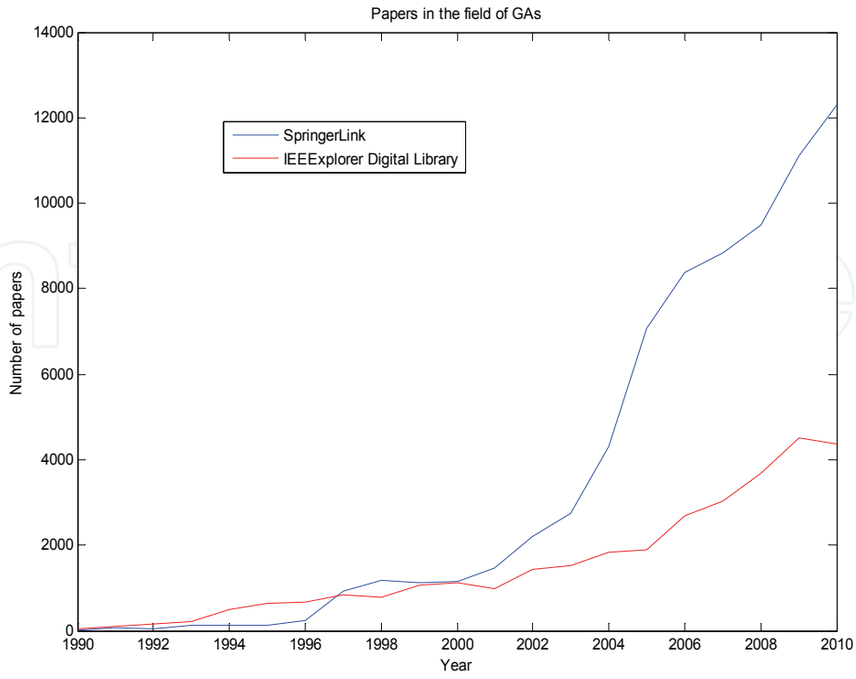


Fig. 1. Increasing the number of works in the field over the past 20 years

Figure 1 represents the number of papers in the field of GAs, in the last 20 years, in two of the most popular databases: SpringerLink from Springer, which contain 81187 papers on GAs, from a total amount of 5276591, and IEEExplore Digital Library from IEEE, which contain 32632 papers on GAs, from a total amount of 2926204.

3. A simple Genetic Algorithm

The set of all the solutions of an optimization problem constitutes the search space. The problem consists in finding out the solution that fits the best, from all the possible solutions. When the search space becomes huge, we need a specific technique to find the optimal solution. GAs provides one of these methods. Practically they all work in a similar way, adapting the simple genetics to algorithmic mechanisms. GA handles a population of possible solutions. Each solution is represented through a chromosome, which is just an abstract representation.

Coding all the possible solutions into a chromosome is the first part, but certainly not the most straightforward one of a GA. A set of reproduction operators has to be determined, too. Reproduction operators are applied directly on the chromosomes, and are used to perform selection of the parents, by using a fitness function (usually the most fitted, with some likelihood), recombinations (crossover) and mutations and over solutions of the problem. "Appropriate representation and reproduction operators are really something determinant, as the behavior of the GA is extremely dependant on it. Frequently, it can be extremely difficult to find a representation, which respects the structure of the search space and reproduction operators, which are coherent and relevant according to the properties of the problems" (Sivanandam & Deepa, 2008).

```
Procedure Genetic Algorithm
begin
    generate randomly the initial population of chromosomes;
    repeat
            calculate the fitness of chromosomes in population;
            repeat
                    select 2 chromosomes as parents;
                    apply crossover to the selected parents;
                    apply mutation to the new chromosomes;
                    calculate the fitness of new child chromosomes;
            until end of the number of new chromosomes
            update the population;
    until end of the number of generations
end
```

Fig. 2. Pseudocode description of the Procedure Genetic Algorithm

Once the reproduction and the fitness function have been properly defined, a GA is evolved according to the same basic structure (see source above in pseudocode). It starts by generating an initial population of chromosomes, which is generated randomly to ensure the genetic diversity. Then, the GA loops over an iteration process to make the next generation. Each iteration consists of fitness evaluation, selection, reproduction, new evaluation of the offsprings, and finally replacement in population. Stopping criterion may be the number of iterations (called here generations), or the convergence of the best chromosome toward the optimal solution.

4. Classification of Genetic Algorithms

Sometimes the cost function is extremely complicated and time-consuming to evaluate. As a result some care must be taken to minimize the number of cost function evaluations. An idea was to use parallel execution of various Simple GAs, and these algorithms are called Parallel Genetic Algorithms (PGAs). PGAs have been developed to reduce the large execution times that are associated with simple genetic algorithms for finding near-optimal solutions in large search spaces. They have also been used to solve larger problems and to find better solutions. PGAs have considerable gains in terms of performance and scalability. There are a lot of methods of PGAs (Independent PGA, Migration PGA, Partition PGA, Segmentation PGA) which are fully described in (Sivanandam & Deepa, 2008).

Hybrid Genetic Algorithms (HGAs) produce another important class of GAs. A hybrid GA combines the power of the GA with the speed of a local optimizer. The GA excels at gravitating toward the global minimum. It is not especially fast at finding the minimum when in a locally quadratic region. Thus the GA finds the region of the optimum, and then the local optimizer takes over to find the minimum. Some examples of HGAs used in Digital Electronics Design will be presented in the next section.

Adaptive genetic algorithms (AGAs) are GAs whose parameters, such as the population size, the crossing over probability, or the mutation probability are varied while the GA is running. “The mutation rate may be changed according to changes in the population; the longer the population does not improve, the higher the mutation rate is chosen. Vice versa,

it is decreased again as soon as an improvement of the population occurs" (Sivanandam & Deepa, 2008).

Fast Messy Genetic Algorithm (FmGA) is a binary, stochastic, variable string length, population based approach to solving optimization problems. The main difference between the FmGA and other genetic approaches is the ability of the FmGA to explicitly manipulate building blocks (BBs) of genetic material in order to obtain good solutions and potentially the global optimum. Some works, like (Haupt & Haupt, 2004), use only the term of Messy Genetic Algorihms (mGAs).

Finally, Independent Sampling Genetic Algorithm (ISGA) are more robust GAs, which manipulate building blocks to avoid the premature convergence in a GA. Implicit parallelism and the efficacy of crossover are enhanced and the ISGAs have been shown to outperform several different GAs (Sivanandam & Deepa, 2008). Other classes of efficient GAs may be implemented for different specific applications.

5. Applications of Genetic Algorithms

"GAs have been applied in science, engineering, business and social sciences. Number of scientists has already solved many engineering problems using genetic algorithms. GA concepts can be applied to the engineering problem such as optimization of gas pipeline systems. Another important current area is structure optimization. The main objective in this problem is to minimize the weight of the structure subjected to maximum and minimum stress constrains on each member. GA is also used in medical imaging system. The GA is used to perform image registration as a part of larger digital subtraction angiographies. It can be found that GAs can be used over a wide range of applications" (Sivanandam & Deepa, 2008). GAs can also be applied to production planning, air traffic problems, automobile, signal processing, communication networks, environmental engineering and so on. In (Bentley & Corne, 2002), Evolutionary Creativity is discussed, using a lot of examples from music, art in general, architecture and engineering design. Evolutionary Electronics, both Analog and Digital, have been investigated in many publications (Bentley & Corne, 2002; Popa, 2004; Popa et al., 2005). (Higuchi et al., 2006) is a very good book on Evolvable Hardware.

Evolvable Hardware (EHW) is a hardware built on software reconfigurable Programmable Logic Devices (PLDs). In these circuits the logic design is compiled into a binary bit string and, by changing the bits, arbitrary hardware structures can be implemented instantly. The key idea is to regard such a bit string as a chromosome of a Genetic Algorithm (GA). Through genetic learning, EHW finds the best bit string and reconfigures itself according to rewards received from the environment (Iba et al., 1996).

In the rest of this section we present three applications in evolutionary design of digital circuits developed by the author, using GAs. First of them describes a method of synthesis of a Finite State Machine (FSM) in a Complex Programmable Logic Device (CPLD), using a standard GA. The other two applications use different techniques of hybridisation of a standard GA: first of them with two other optimisation techniques (inductive search and simulated annealing), to solve the Automatic Test Pattern Generation for digital circuits, a problem described in (Bilchev & Parmee, 1996), and the second one to improve the convergence of the standard GA in evolutionary design of digital circuits, using the new paradigm of Quantum Computation (Han & Kim, 2002).

5.1 Implementation of a FSM using a standard GA

This first example uses extrinsic hardware evolution, that is uses a model of the hardware and evaluates it by simulation in software. The FSM represented in the figure 3 is a computer interface for serial communication between two computers. A transition from one state to another depends from only one of the 4 inputs $x_i, i = \overline{1,4}$. The circuit has 4 outputs, each of them beeing in 1 logic only in a single state. The FSM has 6 states and has been presented in (Popa, 2004).

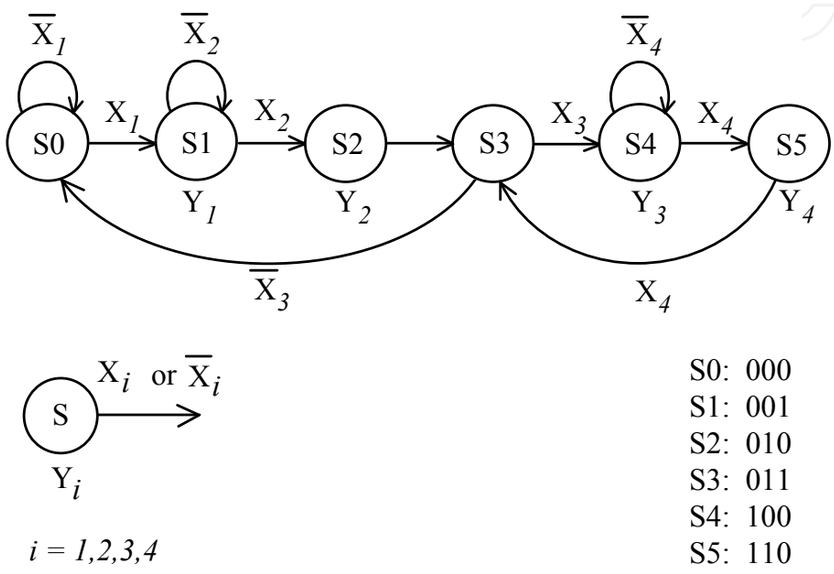


Fig. 3. A FSM described as state transition graph and manual state assignment

With the state assignment given in the figure 3, the traditional design with D flip-flops gives the following equations for the excitations functions:

$$D_2 = x_3 \cdot Q_1 \cdot Q_0 + Q_2 \cdot \overline{Q}_1 \tag{1}$$

$$D_1 = x_2 \cdot \overline{Q}_1 \cdot Q_0 + x_4 \cdot Q_2 + Q_1 \cdot \overline{Q}_0 \tag{2}$$

$$D_0 = x_1 \cdot \overline{Q}_2 \cdot \overline{Q}_0 + \overline{x}_2 \cdot \overline{Q}_1 \cdot Q_0 + Q_1 \cdot \overline{Q}_0 \tag{3}$$

The output functions, are given by the following equations:

$$y_1 = \overline{Q}_1 \cdot Q_0 \tag{4}$$

$$y_2 = \overline{Q}_2 \cdot Q_1 \cdot \overline{Q}_0 \tag{5}$$

$$y_3 = Q_2 \cdot \overline{Q}_1 \tag{6}$$

$$y_4 = Q_2 \cdot Q_1 \tag{7}$$

For the evolutionary design of this circuit we take into account that each boolean function has a maximum number of 5 inputs and a maximum number of 4 minterms. If we want to implement these functions in a PLD structure (an AND array and logic cells configurable as OR gate), then the number of fuse array links is $2 \cdot 5 \cdot 4 = 40$, and we may to consider this number as the total length of the chromosome (Iba et al., 1996).

Our GA is a standard one, with the population size of 30 chromosomes. One point crossover is executed with a probability of 80% and the mutation rate is 2%. Six worse chromosomes are replaced each generation. The stop criterion is the number of generations.

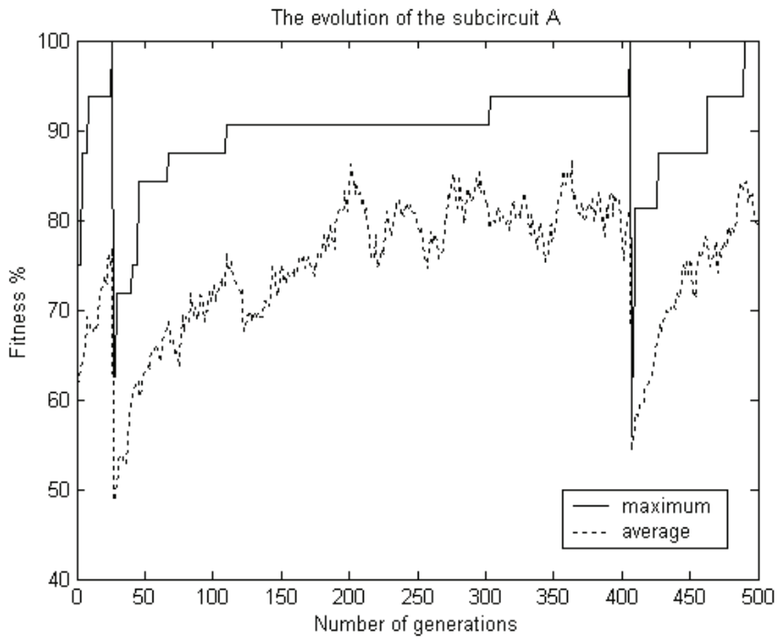


Fig. 4. The evolution of the excitation functions of the computer interface

Figure 4 reflects the evolution of the circuit for the first 3 functions, called excitation functions, which generate the subcircuit A. However, this circuit is built from 3 independent circuits, each generating one output bit. Therefore, the evolution of a circuit with one output bit is repeated 3 times. The Y axis is the correct answer rate. If it reaches 100%, then the hardware evolution succeeds.

In the same way, figure 5 reflects the evolution of the circuit for the output functions, which generate the subcircuit B. The evolution succeeds after a less number of generations because the total search space is in this case much lower than in previous case (all the output functions have only 3 variables).

Evolution may provide some non-minimal expressions for these boolean functions, but minimization is not necessary for PLD implementations. The length of the chromosomes is greater than the optimal one, and the evolved equations are much more complicated than the given equations (1-7). The complete cost of the whole combinational circuit is consisted of 15 gates and 37 inputs for traditional design, and 30 gates and 102 inputs for evolutionary design.

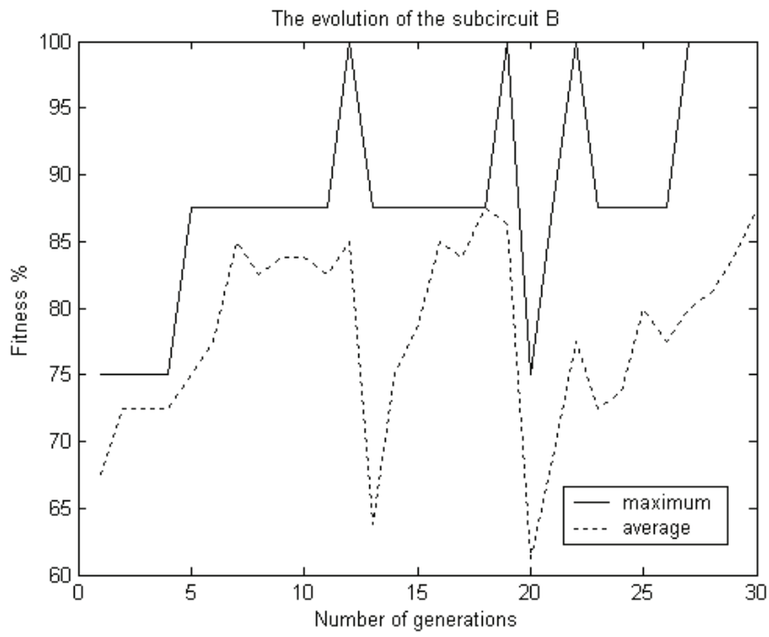


Fig. 5. The evolution of the output functions of the computer interface

We have implemented both the traditional design and the evolved circuit in a real Xilinx XCR3064 CoolRunner CPLD by using the Xilinx ISE 6.1i software. In traditional design, that is using equations (1-7), the FSM used only 7 macrocells from a total number of 64 macrocells, 11 product terms from a total number of 224 product terms, and 7 function block inputs, from a total number of 160. Surprising is the fact that, although evolutionary design, with the same state assignment, provides more complicated equations, the implementation of this circuit in XCR3064XL CPLD also used 7 macrocells from a total number of 64, 10 product terms from a total number of 224, and 7 function block inputs, from a total number of 160. This is even a better result than in preceding case, because the number of product terms is less with 1. Both implementations have used the same number of flip-flops (that is 3/64) and the same number of pins used like inputs/outputs (that is 9/32). We have preserved the state assignment of the FSM, and the subcircuits A and B are in fact as pure combinational circuits. The interesting fact is that our GA have supplied a better solution than the one given by the minimization tool used for this purpose by the CAD software.

5.2 Multiple hybridization of a GA

Hybrid Genetic Algorithms (HGAs) combine the power of the GA with the speed of a local optimizer. Usually the GA finds the region of the optimum, and then the local optimizer takes over to find the minimum. (Bilchev & Parmee, 1996) developed a search space reduction methodology, which was called the Inductive Search. The problem of global optimisation is partitioned into a sequence of subproblems, which are solved by searching of partial solutions in subspaces with smaller dimensions.

This method has been used to solve the Automatic Test Pattern Generation Problem in Programmable Logic Arrays (PLAs), that is to find an effective set of input test vectors,

which are able to cover as many as possible faults in the circuit (we have taken into account two PLA structures with a total number of 50 and respective 200 stuck-at 0 possible faults).

(Wong & Wong, 1994) designed a HGA using the algorithm of Simulated Annealing as local optimizer. The optimisation process in Simulated Annealing is essentially a simulation of the annealing process of a molten particle. Starting from a high temperature, a molten particle is cooled slowly. As the temperature reduces, the energy level of the particle also reduces. When the temperature is sufficiently low, the molten particle becomes solidified. Analogous to the temperature level in the physical annealing process is the iteration number in Simulated Annealing. In each iteration, a candidate solution is generated. If this solution is a better one, it will be accepted and used to generate yet another candidate solution. If it is a deteriorated solution, the solution will be accepted with some probability.

Each of this two methods of hybridisation discussed above have some advantages. The inductive search effort at each inductive step controls the trade-off between the

```
Procedure MHGA
begin
    Initialize a partial solution for N = 1 and establish the initial temperature T0 ;
    for k = 2 to N ,
        Generate randomly the initial population of chromosomes;
        repeat
            append each chromosome to the partial solution, and evaluate it;
            repeat
                select, proportional with fitness, 2 parents;
                apply crossover to obtain 2 offsprings;
                apply mutation to the new chromosomes;
                calculate the fitness of new chromosomes;
                the new chromosomes are accepted or not accepted;
            until   end of the number of chromosomes
            update the population, according with the fitness;
            the temperature is decreased;
        until end of the number of generations
        Update the partial solution;
    end
end
```

Fig. 6. The structure of the MHGA

computational complexity and the expected quality of results, while Simulated Annealing avoids the premature convergence and reduces the adverse effects of the mutation operation. In (Popa et al., 2002) we proposed a HGA that cumulates all these advantages in a single algorithm, through a double hybridisation of the Genetic Algorithm: with Inductive Search on the one hand, and with Simulated Annealing technique on the other hand. The structure of the Multiple Hybridated Genetic Algorithm is presented in figure 6.

We have conducted the experiments with all three HGAs described above, in the purpose to find the maximum fault coverage with a limited number of test vectors. We have tested first a PLA structure with 50 potential "stuck-at 0" faults, taking into account the maximum

coverage with the faults with only 6 test vectors, and results may be seen in the figure 7. Then, we repeated the same algorithm for a more complicated PLA structure, with 200 potential "stuck-at 0" faults, and we tried to cover the maximum number of faults with 24 test vectors. The evolutions of these three algorithms may be seen in figure 8.

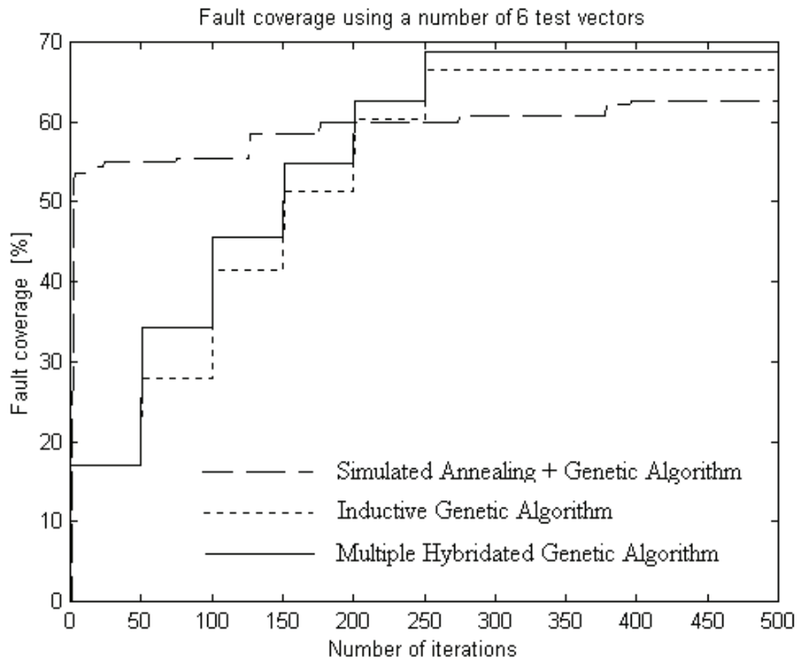


Fig. 7. Fault Coverage Problem of 50 possible faults solved with three HGAs in 500 iterations

If n is the number of covered faults and N is the number of all faults in the fault population, the associated fitness function is $f = \frac{n}{N} \cdot 100\%$. There may also be a number of constraints concerning the possible combinations of input signals. The designers of the circuit define the set of legal combinations in terms of the legal states of a number of channels. The set of all legal templates defines the feasible region. The main genetic parameters used in these algorithms are: a population size of 20 chromosomes, uniform crossover with 100% rate, uniform mutation with 1% rate. The maximum fault coverage achieved with the Multiple Hybridated Genetic Algorithm after 500 iterations was about 69%, while the maximum fault coverage achieved with the Inductive Genetic Algorithm, the best of the two single hybridated genetic algorithms, was about 66%. These results represent the average values of 5 succesive runnings. We have tried even with 10 or more number of runnings, but the results are basically the same.

Another set of experiments were made on a more complex digital structure of PLA type with 200 possible faults. Figure 8 shows the comparative performances of the three HGAs on this fault coverage problem. The number of input test vectors is 24. After 250 fitness function calls, that is 25 iterations, each with 10 generations per inductive step, the fault coverage of the Multiple Hybridated Genetic Algorithm is with about 1% better than the fault coverage of the Inductive Genetic Algorithm.

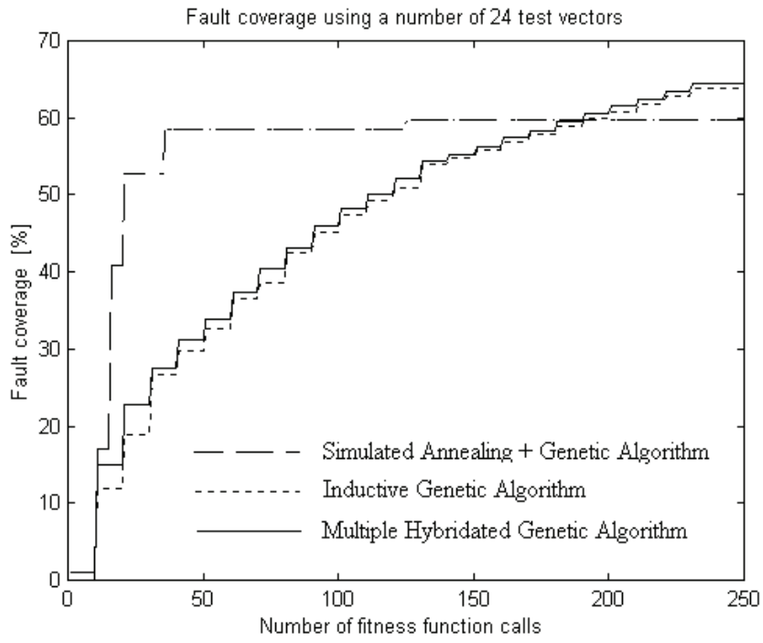


Fig. 8. Fault Coverage Problem of 200 possible faults solved with three HGAs

These experiments show that the proposed MHGA seems to offer a better performance than the two other HGAs: the Inductive Genetic Algorithm and the Genetic Algorithm hybridated by Simulated Annealing. We have proved on two different examples, with different complexities, that MHGA offers the greatest value of fault coverage in Automatic Test Pattern Generation Problem in digital circuits of PLA type.

5.3 A Quantum Inspired GA for EHW

Quantum Inspired Genetic Algorithm (QIGA) proposed in (Popa et al., 2010) uses a single chromosome, which is represented like a string of qubits, as is described in (Han & Kim, 2002; Zhou & Sun, 2005). A quantum chromosome which contains n qubits may be represented as:

$$q = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_n \\ \beta_1 & \beta_2 & \beta_n \end{bmatrix}, \quad (8)$$

where each couple α_i, β_i, for $i = 1,...,n$, are the probability amplitudes associated with the $|0\rangle$ state and the $|1\rangle$ state such that $\alpha_i^2 + \beta_i^2 = 1$ and the values α_i^2 and β_i^2 represent the probability of seeing a conventional gene, 0 or 1, when the qubit is measured.

A quantum chromosome can be in all the 2^n states at the same time, that is:

$$|q\rangle = a_0|00...0\rangle + a_1|00...1\rangle + ...a_{2^n-1}|11...1\rangle, \quad (9)$$

where a_i represents the quantum probability amplitude, and a_i^2 is the probability of seeing the i-th chromosome from the all 2^n possible classic chromosomes (Zhou & Sun, 2005).

Due to this superposition of states in a quantum chromosome, we use a single chromosome in population. In Conventional Genetic Algorithm (CGA) or Simple Genetic Algorithm with the structure given in figure 2, the population has always a number of chromosomes, and the efficiency of the algorithm depends usually on the size of population. But a quantum chromosome can represent all the possible conventional chromosomes at the same time, and so, it may generates an arbitrary population of conventional chromosomes each generation. Quantum population will be transformed to conventional population when the fitness is evaluated.

Single Chromosome Quantum Genetic Algorithm (SCQGA) is described in (Zhou & Sun, 2005). In the first step, a quantum chromosome is generated using (8). A random number is compared with probabilities of each qubit, and it collapses to 0, or to 1. The conventional population of N chromosomes is obtained by repeating this process N times. In the next step, the fitness value is calculated for each conventional chromosome. It requires a lot of time, that involves the speed performance of the algorithm. The same problem of fitness evaluation and the low speed of the algorithm subsists also in CGAs.

Our idea, which was implemented in QIGA, was to initiate the collapse of the quantum chromosome each generation but, from time to time to generate a whole population of conventional chromosomes, and in the remaining iterations to generate only a single conventional chromosome. A new parameter, which we called the *probability of collapse*, establishes the rate of generating a conventional population during the evolution. The last important step of the algorithm is to establish a method of updating the quantum chromosome from the current generation to the next one. QIGA uses the same method described in (Han, 2003). The idea is to modify the probabilities of each quantum gene (or qubit) from the quantum chromosome using quantum rotation gate. This operator changes the probability amplitude by altering the quantum phase θ to $\theta + \Delta\theta$. The idea for the construction of the rotation gate is to make the changing of the entire population (quantum chromosome) to the direction of the best individual. Each bit from the best conventional chromosome is compared with the adequate bit from the average version of the quantum chromosome (this version is build using a probability of 0.5 for each qubit). If the two bits are equal with 0 or 1, then $\Delta\theta = 0$. If the bit of the best chromosome is 1 and the other one is 0, then $\Delta\theta = a$, otherwise $\Delta\theta = -a$. The angle parameter of the rotation gate $\Delta\theta$ may be 0, -a, or a, depending on the position of each qubit in chromosome. The parameter a is a positive small parameter, which decides the evolving rate (Zhou & Sun, 2005).

Basic structure of QIGA is given in figure 9. $q(t)$ is the quantum chromosome in the iteration t, and $P(t)$ is the population in the same iteration t. This population may contain a lot of chromosomes, or only one, depending on the probability of collapse in $q(t)$. These three algorithms, CGA, SCQGA and QIGA have been compared on the same problem, which consists on synthesis of a boolean function with 4 variables, using different logic gates. The chromosomes define the connection in the network between the primary inputs and primary outputs of the gates, and decide the logic operators of the gates. The population of CGA has 64 chromosomes, 20 of them being changed each generation, and genetic operators use a single point 100% crossover and 5% rate mutation.

Figure 10 illustrates the average of evolutions of the three algorithms after 10 successful runnings on 300 generations. A successful running presumes a fitness evaluation of 100%,

that is the truth table of the evolved function must be identical with the truth table of the specified function. We can see some similarities in these evolutions, but significant differences may be seen in Table 1.

```
Procedure QIGA
begin
    t ← 0
    Initialize a quantum chromosome q(t);
    if the collaps of q(t) is likely
            generate multiple chromosomes in population P(t);
    else
            generate a single chromosome in population P(t);
    end
    evaluate all the chromosomes in population P(t);
    store the best solution b among P(t);
    while (not termination condition) do
            begin
                    t ← t + 1
                    if the collapse of q(t-1) is likely
                            generate multiple chromosomes in population P(t);
                    else
                            generate a single chromosome in population P(t);
                    end
                    evaluate all the chromosomes in population P(t);
                    update q(t) using quantum gates;
                    store the best result b among P(t);
            end
    end
end
```

Fig. 9. The structure of the QIGA

In CGA, global time of a successful run is about 74 seconds, and this value consists of both self time and the time spent for multiple evaluations of chromosomes in different populations. Self time is the time spent in an algorithm, excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling, but this additional time is not important in our case. Evaluation time is almost 60 seconds, because the number of appeals to the evaluation function is elevated (25200 calls, that is evaluation of 64 plus 20 chromosomes in 300 generations).

In SCQGA, global time is less than 40 seconds, because the number of calls to the evaluation function is less than above (only 19200 calls, that is evaluation of 64 chromosomes in 300 generations), and this quantum algorithm doesn't use anymore genetic operators like crossover and mutation. Finally, our QIGA has a global time less than 20 seconds, as a consequence of the insignificant number of calls to the evaluation function (only 4836 calls, a random number given by the probability of collapse). Self time is comparable with SCQGA, and evaluation time is less than 12 seconds. Taking into account all these times, QIGA has the best ratio between evaluation and global time.

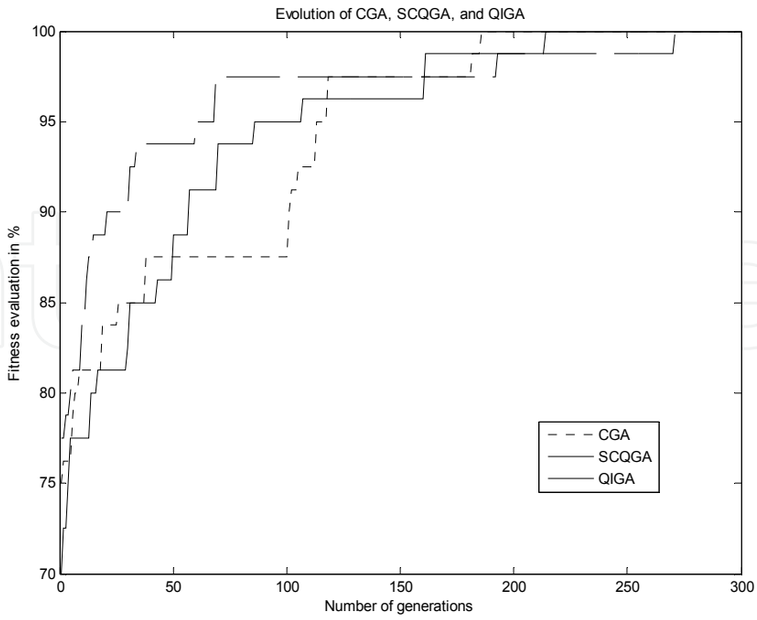


Fig. 10. The evolutions of CGA, SCQGA and QIGA

Parameter	CGA	SCQGA	QIGA
Global time	73.990 s	38.599 s	19.263 s
Self time	2.447 s	1.417 s	1.390 s
Evaluation time	59.561 s	31.536 s	11.750 s
Calls of evaluation function	25200	19200	4836
Ratio between evaluation and global time	80.5 %	81.7 %	60.9 %
Number of generations	300	300	300
Successful runnings in 10 attempts (with fitness of 100%)	7	6	6

Table 1. A comparison between CGA, SCQGA and IQGA

Unfortunately, the number of successful runs in 300 generations is only in the order of 70% for CGA, and 60% for the rest two algorithms. It occurs due to the constraint that only 100% in fitness evaluation is accepted. In other applications, this constraint may be not critical.

6. Conclusion

In this chapter we did a summary outline of GAs and discussed some possible applications. We presented three extrinsic evolutionary designs of digital circuits at gate level using GAs.

Future research must be done in this area. Firstly it is important to find a better representation of the circuit in chromosomes, because complex functions need a great number of architecture bits, which directly influences the GA search space. EHW successfully succeeds only when fitness reaches 100% and in huge search spaces this condition may be not always possible. This is the main reason that for the time being the

complexity of evolved circuits is so far small. In our opinion, conclusion drawn in the paper (Yao & Higuchi, 1999) is still available: "EHW research needs to address issues, such as scalability, online adaptation, generalization, circuit correctness, and potential risk of evolving hardware in a real physical environment. It is argued that a theoretical foundation of EHW should be established before rushing to large-scale EHW implementations".

Recently appeared the idea of hybridization of a GA with elements of quantum computation (Han & Kim, 2002; Han, 2003). We have proposed a new quantum inspired genetic algorithm (QIGA) considerably faster than other similar algorithms, based on the idea of introducing a new parameter, which we called the probability of collapse, and to initiate the collapse of the quantum chromosome in order to generate a conventional population of chromosomes from time to time, and not each generation, as usually is done. We believe that some improvements in this method may be found in a future research, by establishing of a new method of updating the quantum chromosome from the current generation to the next one. Finally, some hybridization techniques may be useful for new quantum inspired evolutionary algorithms. (Rubinstein, 2001) used Genetic Programming to evolve quantum circuits with various properties, and (Moore & Venayagamoorthy, 2005) has developed an algorithm inspired from quantum evolution and Particle Swarm to evolve conventional combinational logic circuits.

7. References

Bentley, P. J. & Corne, D. W. (Ed(s).). (2002). *Creative Evolutionary Systems*, Academic Press, ISBN: 1-55860-673-4, San Francisco, USA

Bilchev, G. & Parmee, I. (1996). Constraint Handling for the Fault Coverage Code Generation Problem: An Inductive Evolutionary Approach, *Proceedings of 4-th Conference on Parallel Problem Solving from Nature (PPSN IV)*, pp. 880-889, Berlin, September 1996

Burda, I. (2005). *Introduction to Quantum Computation,* Universal Publishers, ISBN: 1-58112-466-X, Boca Raton, Florida, USA

Forbes, N. (2005). *Imitation of Life. How Biology Is Inspiring Computing,* MIT Press, ISBN: 0-262-06241-0, London, England

Han, K. H. & Kim, J. H. (2002). Quantum-Inspired Evolutionary Algorithm for a Class of Combinatorial Optimization, *IEEE Transactions on Evolutionary Computation,*vol.6, no.6, (December 2002), pp. 580-593, ISSN 1089-778X

Han, K. H. (2003). *Quantum-Inspired Evolutionary Algorithm,* Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Korea, 2003

Haupt, R. L. & Haupt, S. E. (2004). *Practical Genetic Algorithms* (second edition), Wiley-Interscience, ISBN: 0-471-45565-2, New-Jersey, USA

Higuchi, T.; Liu, Y. & Yao, X. (Ed(s).). (2006). *Evolvable Hardware*, Spinger-Verlag, ISBN-13: 978-0387-24386-3, New-York, USA

Iba, H.; Iwata, M. And Higuchi, T. (1996). Machine Learning Approach to Gate-Level Evolvable Hardware. *Proceedings of the First International Conference on Evolvable Systems ICES'96,* pp.327-343, Tsukuba, Japan, October 1996

Mazumder, P. & Rudnick, E. M. (1999). *Genetic Algorithms for VLSI Design, Layout & Test Automation,* Prentice Hall PTR, ISBN: 0-13-011566-5, Upper Saddle River, New Jersey, USA

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs* (third edition), Springer-Verlag, ISBN: 3-540-58090-5, New-York, USA

Mitchell, M. (1997). *An Introduction to Genetic Algorithms* (third printing), MIT Press, ISBN: 0-262-13316-4, London, England

Moore, Ph. & Venayagamoorthy, G. K. (2005). Evolving Combinational Logic Circuits using a Hybrid Quantum Evolution and Particle Swarm Inspired Algorithm, *Proceedings of the 2005 NASA/DoD Conference of Evolution Hardware EH'05,* pp. 97–102, 2005

Popa, R.; Aiordachioaie, D. & Nicolau, V. (2002). Multiple Hybridization in Genetic Algorithms, *The 16-th European Meeting on Cybernetics and Systems Research EMCSR'2002*, pp.536-541, ISBN 3-85206-160-1, Vienna, Austria, April 2-5, 2002

Popa, R. (2004). Evolvable Hardware in Xilinx XCR3064 CPLD, *IFAC Workshop on Programmable Devices and Systems, PDS 2004*, pp. 232-237, ISBN: 83-908409-8-7, Cracow, Poland, November 18-19, 2004

Popa, R.; Aiordachioaie, D. & Sîrbu, G. (2005). Evolvable Hardware in Xilinx Spartan 3 - FPGA, *2005 WSEAS International Conference on Dynamical Systems and Control*, pp. 66-71, ISBN: 960-8457-37-8, Venice, Italy, November 2-4, 2005

Popa, R.; Nicolau, V. & Epure, S. (2010). A New Quantum Inspired Genetic Algorithm for Evolvable Hardware, *3rd International Symposium On Electrical and Electronics Engineering ISEEE 2010*, pp. 64-69, ISBN: 978-1-4244-8407-2, Galați, Romania, September 16-18, 2010 (in IEEExplore Digital Library, DOI 10.1109/ ISEEE.2010.5628539)

Rubinstein, B. I. P. (2001). Evolving Quantum Circuits using Genetic Programming, *Proceedings of the 2001 Congress on Evolutionary Computation, CEC2001,* pp. 114–121, IEEE Press, 2001

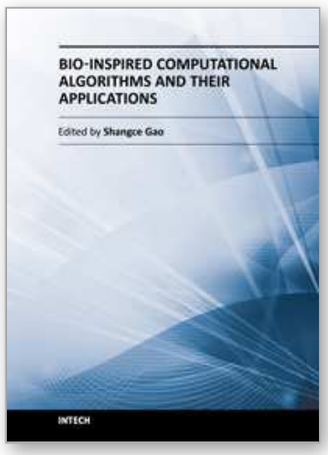
Schöneburg, E.; Heinzmann, F. & Feddersen, S. (1994). *Genetische Algorithmen und Evolutionsstrategien. Eine Einführung in Theorie und Praxis der simulierten Evolution,* Addison-Wesley, ISBN: 5-89319-493-2, München, Germany

Sivanandam, S. N. & Deepa, S. N. (2008). *Introduction to Genetic Algorithms*, Springer-Verlag, ISBN: 978-3-540-73189-4, India

Wong, K. P. & Wong, Y. W. (1994). Development of Hybrid Optimisation Techniques Based on Genetic Algorithms and Simulated Annealing, *Workshop on Evolutionary Computation (AI'94),* pp. 127-154, Armidale, Australia, November 1994

Yao, X. & Higuchi, T. (1999). Promises and Challenges of Evolvable Hardware, *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews, Evolutionary Computation,*vol.29, no.1, (February 1999), pp. 87-97, ISSN 1094-6977

Zhou, S. & Sun, Z. (2005). A New Approach Belonging to EDAs: Quantum-Inspired Genetic Algorithm with Only One Chromosome, *International Conference on Natural Computing, ICNC2005,* pp. 141–150, LNCS 3612, Springer-Verlag, 2005



Bio-Inspired Computational Algorithms and Their Applications

Edited by Dr. Shangce Gao

ISBN 978-953-51-0214-4
Hard cover, 420 pages
Publisher InTech
Published online 07, March, 2012
Published in print edition March, 2012

Bio-inspired computational algorithms are always hot research topics in artificial intelligence communities. Biology is a bewildering source of inspiration for the design of intelligent artifacts that are capable of efficient and autonomous operation in unknown and changing environments. It is difficult to resist the fascination of creating artifacts that display elements of lifelike intelligence, thus needing techniques for control, optimization, prediction, security, design, and so on. Bio-Inspired Computational Algorithms and Their Applications is a compendium that addresses this need. It integrates contrasting techniques of genetic algorithms, artificial immune systems, particle swarm optimization, and hybrid models to solve many real-world problems. The works presented in this book give insights into the creation of innovative improvements over algorithm performance, potential applications on various practical tasks, and combination of different techniques. The book provides a reference to researchers, practitioners, and students in both artificial intelligence and engineering communities, forming a foundation for the development of the field.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen