International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:1, No:7, 2007

Mutation Rate for Evolvable Hardware

Emanuele Stomeo, Tatiana Kalganova, and Cyrille Lambert

***Abstract*—**Evolvable hardware (EHW) refers to a self-reconfiguration hardware design, where the configuration is under the control of an evolutionary algorithm (EA). A lot of research has been done in this area several different EA have been introduced. Every time a specific EA is chosen for solving a particular problem, all its components, such as population size, initialization, selection mechanism, mutation rate, and genetic operators, should be selected in order to achieve the best results. In the last three decade a lot of research has been carried out in order to identify the best parameters for the EA's components for different "test-problems". However different researchers propose different solutions. In this paper the behaviour of mutation rate on (1+λ) evolution strategy (ES) for designing logic circuits, which has not been done before, has been deeply analyzed. The mutation rate for an EHW system modifies values of the logic cell inputs, the cell type (for example from AND to NOR) and the circuit output. The behaviour of the mutation has been analyzed based on the number of generations, genotype redundancy and number of logic gates used for the evolved circuits. The experimental results found provide the behaviour of the mutation rate to be used during evolution for the design and optimization of logic circuits. The researches on the best mutation rate during the last 40 years are also summarized.

***Keywords*—**Evolvable hardware, mutation rate, evolutionary computation, design of logic circuit.

I. Introduction

EVOLVABLE hardware (EHW) [1-2] also called evolutionary electronics or hardware evolution, gained inspiration from the consideration of three disciplines: computer science, electronic engineering and biology. The basic schema of the approach is given in Fig. 1. EHW is a technique, inspired by natural evolution, to automatically design circuits, where the configuration is under the control of evolutionary algorithms (EA) [3]. These techniques began to be treated with increasing interest since the 60s when Holland introduced the concept of genetic algorithms (GA) [4], [5], which are the most general methods of solving search and optimization problems.

A lot of research has been done in order to improve the classic GA for a given problem and many new versions of the original EA have been introduced as genetic programming (GP) [6], evolution strategy (ES) [7-10] grammatical evolution (GE) [11], evolutionary programming (EP) [12], [13], Cartesian Genetic Programming (CGP) [14], parallel genetic algorithm (PGA) [15], adaptive genetic algorithm (AGA) [16], particle swarm optimization (PSO) [17]. However, every time a specific evolutionary algorithm is chosen, all its parameters such as population size, type of initialization, selection mechanism, and genetic operators should be tuned in order to achieve the best results. This is because the efficiency of EA is highly dependent on all its parameters [3], [16], [18-21]. In order to find the best values for evolutionary algorithm's parameters several researchers tuned them [20-22] in an attempt to find a general optimum for test functions. However, the results obtained are different for different types of algorithms and problems as shown in Table I. The test functions used not included the circuit design.

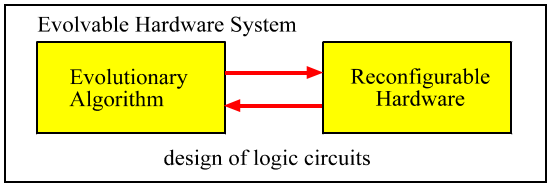

Fig. 1 Basic evolvable hardware system. The evolutionary algorithm sends the circuit configurations to the chip. The electronic chip configures itself with the input configuration received and sends the fitness value to the evolutionary algorithm. The fitness value describes how good the circuit configuration found is. Based on the value received, the EA modifies the chromosomes and gives back a new configuration to the chip

For the same reason, different problems are solved using different parameters value. So, based on these results, we start to investigate the behaviour of the mutation rate in the design of logic circuits. The mutation operator involves flipping the value of some genes (Fig. 2). The aim of this operation is to bring more change (diversity) into the population. By increasing the mutation rate, the genetic search will be transformed into a random search but it also helps to reintroduce lost genetic material [29].

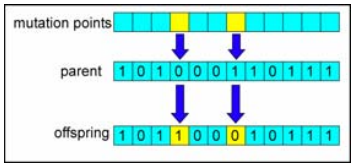

Fig. 2 Mutation operator: the offspring is generated from only one parent and one ma

Manuscript received July 15, 2005. This work was supported in part by the EPSRC under Grant GR/S17178/.

E. Stomeo, C. Lambert are PhD students at School of Engineering and Design, Brunel University, West London. T. Kalganova is lecture at the same university. UB8 2TR, Uxbridge, Middlesex, UK. (phone: 0044 01895 266777; e-mail: emanuele.stomeo@brunel.ac.uk).

TABLE I
RESEARCH RESULTS ON MUTATION RATE

Author	Year	Approach	Proposed mutation rate	Problems
De Jong [21]	1975	GA (for online and offline performance)	0.001	General optimization problems (EHW not included)
Grefenstette [20]	1986	Meta GA	0.01	General optimization problems (EHW not included)
Shaffer et al [23]	1989	GA (using online average performance)	0.005-0.01	Multimodal functions, FIR filter, 30 city travel sales person, graph partitioning
Mühlenbein et al. [24]	1992	Iterated Hillclimbing or (1+1,*m*,*hc*)-algorithm	$1/l$ l=chromosome length	Binary functions
Srinivas et al. [25]	1994	AGA	$0.5(f_{max}-f)/(f_{max}-f_{avg})$ where f_{max} is the maximum fitness value and f_{avg} is the average of the fitness	Several multimodal function including TSP, neural network weight optimization problems and generation of test vectors for VLSI circuits
Niwa et al [26]	1995	GA	$1/2n$ n=population size	Markov chain
Haupt [27]	2000	GA	0.05-0.2	Electromagnetic (array factors)
Nijssen [28]	2003	(1:λ) EA	$1/l$ l=bit-string length	Trap functions

The evaluation of the evolved logic circuits can be done via software [23], [30-34] or via hardware [35-36]. The extrinsic evolvable hardware has been chosen as an object of investigation due to its ability to collect any relative information fairly easy. The (1+λ) rudimentary evolution strategy [37-39] has been chosen as evolutionary algorithm.

The performance of the evolutionary algorithm (number of generations required to completely design the logic circuits) has been explored, together with the quality (based on redundancy and number of logic gates used during design and optimization of the logic circuits) of the obtained results.

The experimental results (average out of 100 runs all completely evolved for each logic circuits, which are randomly generated) achieved indicates that a fixed mutation rate should not be used for designing logic circuits. Furthermore the behaviour of the mutation rate to be used during evolution, for those who want to use a dynamic mutation rate for design and optimization of logic circuits, has been extrapolated. In this paper we focus only on online average performance [23].

The paper is organised as follows, section 2 describes an extrinsic evolvable hardware system, from the definition of the evolutionary algorithm to the description of the fitness functions implemented. Section 3 gives the system set-up for the EA used. Section 4 presents the experimental results. Last section gives conclusions and indicates possible areas for future investigation.

II. EXTRINSIC EVOLVABLE HARDWARE

In this section the evolutionary algorithm used to evolve logic circuits, together with fitness function and chromosome representations are presented. The approach in question was introduced in [30].

A. Chromosome Encoding

The chromosome defines the structure of the logic circuit. In our approach the logic circuit is presented as a rectangular array of logic gates. The type of each logic cell is randomly chosen from the set of AND, OR, XOR, NOT and MUX, where MUX is a multiplexer with 2 inputs and one control signal.

The chromosome is represented by a 3 level structure (see Fig. 3):

- Geometry level contains information about the *number of rows*, the *number of columns* of the rectangular array and the degree of internal connectivity, also referred to as *level-back parameter* [14]. The level-back parameter, or so called connectivity parameter, defines how many columns of cells to the left of the current column might have their outputs connected to the inputs of the current cell.
- Circuit level describes the array of cells and determines the circuit's outputs
- Gate level represents the structures of each cell in the circuit.

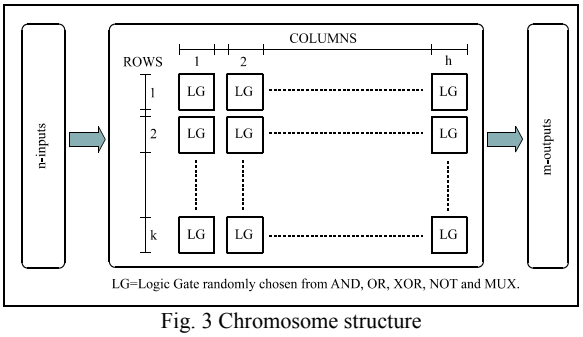

Fig. 3 Chromosome structure

B. Case of Study

At this point some definitions should be introduced. In this paper N_{lg} represents the total number of logic gates in our chromosome. N_{alg} refers to the number of active logic gates, which are the logic gates that are currently used in a circuit configuration; therefore N_{alg} is equal to or less than N_{lg}. N_{plg} represents the number of primitive logic gates (AND, OR, XOR, NOT and MUX) used in the chromosome. The redundancy has been defined as:

$$r = 1 - \frac{N_{a\lg}}{N_{\lg}} \tag{1}$$

Let us consider a numerical example. Assume that the chromosome in Fig.4 is made of the a rectangular array with 6 columns and 4 rows of logic gates.

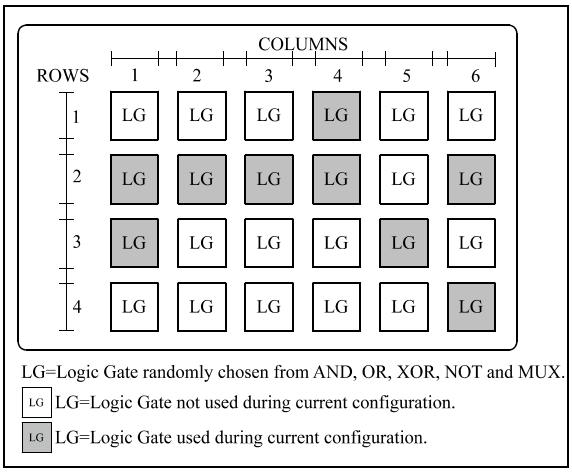

Fig. 4 Example of chromosome during evolution

The highlighted logic gates are used for a particular configuration; therefore they are connected to each other in order to create the logic circuit. The logic gates not highlighted are not connected, so they are redundant. In this example, N_{lg} (the total number of logic gates) is 6*4=24. The number of active logic gates (the logic gates that participate in creating the digital circuit) N_{alg} is 9. The redundancy, calculated using equation (1) for the circuit's configuration of Fig. 4 is 0.625.

C. Fitness Function

The fitness function evaluates the evolved circuits in terms of their functionality. In our experiment a dynamic fitness function has been considered. It has two main criteria: first *design*, the fully functional circuit and second, once the circuit is fully functional evolved, *optimization* which leads to reduced numbers of active logic gates used in the circuit configuration.

The dynamic fitness function *f* is calculated as:

$$f = \begin{cases} f_1 & f < 100 \quad \text{circuit design} \\ f_1 + f_2 & f \geq 100 \quad \text{circuit optimization} \end{cases} \tag{2}$$

where f_1 is a design criterion that defines the percentage of correct bits in the evolved circuit, f_2 is the optimization criterion for the optimization stage.

The fitness function for the functionality of the evolved circuit f_1, or so called design criterion is calculated as follows:

$$f_1 = \frac{100}{m \cdot p} \sum_{f_c}^{2^{n-1}} \sum_{i=0}^{m-1} 2^{i-1} \cdot \left| y_i - d_i \right| \tag{3}$$

where *m* and *n* are the number of outputs and the number of inputs of the given logic function, respectively; *p* is the number of input-output combinations; y_i is the i^{th} digit of the output combination produced by the evaluation of the circuit, d_i is the desired output for the fitness case f_c. $|yi\text{-}di|$ is the absolute difference between the actual and the required outputs.

The fitness function for the optimization stage is calculated as:

$$f_2 = \left(N_{\lg} - N_{a\lg}\right) \cdot N_{p\lg} \tag{4}$$

Fig. 6 shows the behaviour of a dynamic fitness function during the evolution of a 2 bit multiplier (see Fig. 5). In Fig. 6 two different stages are noticeable. The first shows the design of the multiplier, with each generation the fitness function value increases until it reaches 100%. At this point the functionality of the circuit is completely evolved. During the first stage the fitness function is calculated based on equation 3. The second stage starts just after the circuit is evolved. This stage performs the optimization of evolved circuits by reducing with each generation the number of active logic gates. Furthermore during this stage the fitness function, calculated from equation 4, also increases its value because the circuit is better optimised.

x_0	x_1	x_2	x_3	f_0	f_1	f_2	f_3
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Fig. 5 Truth table of a 2 bit multiplier

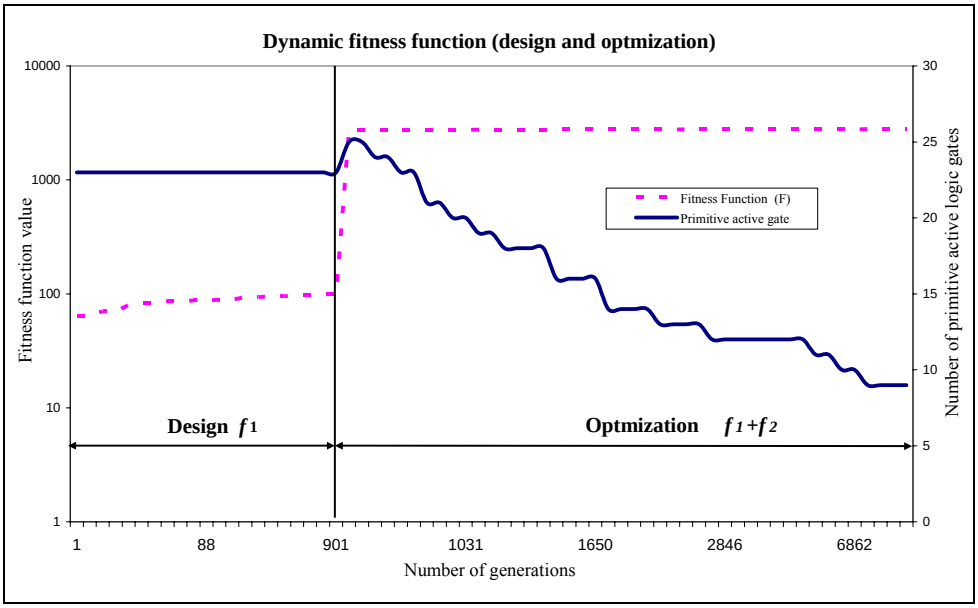

Fig. 6 This graphic shows the behaviour of the dynamic fitness function. Two different stages may be seen, design f1 and optimization (f1 + f2). It is also notable that the number of primitive active gates is reduced

D. Evolutionary Algorithm

All the experiments have been carried out using the (1+λ) rudimentary evolution strategy, where λ represents the population size [30].

Firstly, all the chromosomes are randomly initialized. Secondly, the fitness function of each individual is calculated, the fittest individual is selected and duplicated for the population to the next generations. The new population is brought up to date by mutating the best chromosome of the previous generation (see Fig. 7).

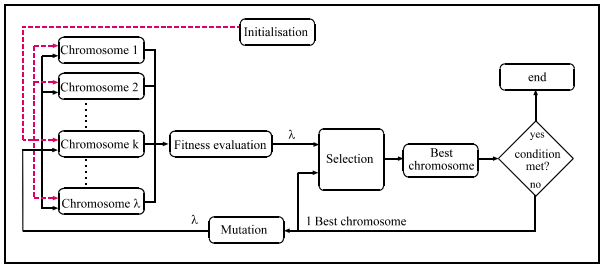

Fig. 7 Schematic of (1+λ) rudimentary evolution strategy

III. Setting Parameters

In this section the system set-up used to carry out all the experiments is described. Firstly the EA's parameters are given, and then the circuit layout and the logic of the evolved circuits are provided. Finally a description of the circuits evolved is given together with one example.

A. Evolutionary Algorithm

In Table II the EA's parameters are given; where the number of generations refers to the number of cycles which each experiment has been evolved; population size refers to the number of different chromosomes; termination criteria is the maximum number of generations the EA can perform before the process will be stopped; the mutation rate modifies the cell input, cell type (for example form AND to XOR) and circuit output. Each logic circuit is evolved 100 times with fixed ES parameters (as reported in Table I). For the given logic functions the results are considered only if the logic circuit has been successfully evolved 100 times out of 100 runs (i.e. success rate 100%). Each logic circuit has been evolved 100 times for each different mutation rate, starting from 0.09 to 0.5 with an increasing step of 0.01. In Table III the features of the circuit layout are given. Definitions of the number of rows, columns and level back have been provided in the previous section. The logic gates that participate in evolutionary processes are AND, OR, XOR, NOT, and multiplexer with 2 inputs and one control.

TABLE II
Initial Data for the Experiments Carried Out Using (1+λ) Rudimentary ES

Number of generations	5000
Population size	5
Number of runs per each evolved circuit	100
Elitism is applied	
Termination criteria for evolutionary process	5000 generations
Mutation rate	0.09 – 0.5

TABLE III
INITIAL DATA: DIMENSION SIZE AND CONNECTIVITY OF THE CIRCUIT LAYOUT USED DURING SIMULATIONS

Circuit layout	Number of rows	10
	Number of columns	10
	Level back or connectivity parameter [14]	10

The connection between building blocks (combination of primitive logic gates) is in interactive and cascade mode. Each logic gate has up to 4 inputs. The structures of cascade and interactive building blocks are given in Fig. 8 and Fig. 9. Each of these blocks represents the combination of primitive logic gates.

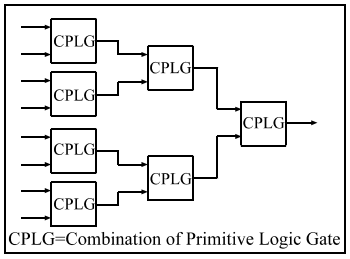

Fig. 8 Cascade building blocks

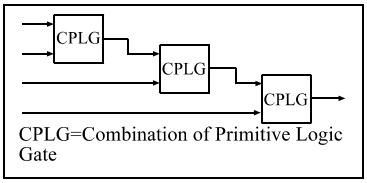

Fig. 9 Interactive building blocks

The logic circuits evolved are randomly generated and fully defined by truth tables. The truth tables used to describe the logic circuits are compatible with the Berkeley format, see Fig. 10. Where .i specifies the number of inputs, .o the number of outputs, .p the number of product or input-output combinations and .e the end of file.

```
.i 3
.o 3
.p 8
000 010
001 011
010 101
011 101
100 011
101 110
110 101
111 011
.e
```

Fig. 10 Example of truth table (in Berkley format) used for the evolution of logic circuit with (1+λ) evolution strategy

IV. Experimental Results

In this section the results of the evolved logic circuits are presented. The intention of these experiments is to analyse the variation of mutation rate depending on the evolution strategies, by taking into account

- the number of generations required to design the circuit
- the number of active logic gates obtained during design and optimization stages
- the redundancy of logic gates

In Fig. 11 and Fig. 12 the average of the *number of generations* of the evolved logic circuits with 2 and 3 inputs respectively is given. This average is calculated by taking into account the results out of 100 runs, which are all successfully evolved. To better clarify how the average has been calculated, one solution has been considered and analyzed. Let us consider the circuit with 2 inputs and 3 outputs see Fig. 11. This circuit has been evolved 100 times (with a success rate of 100%) with a mutation rate equal to 0.009; 100 times with mutation rate 0.01 and so on until the mutation rate is equal to 0.5. So, this circuit has been evolved 1500 times. The average is calculated out of 100 runs per each value of mutation rate. Therefore each point on that graph represents the average out of 100 runs of the number of generations required to evolve the circuits. From Fig. 11 and Fig. 12 it may be observed that, in terms of the number of generations for evolving small circuits, the best value of mutation rate is 0.1. By increasing the complexity of the logic circuit based on the number of input-output combinations [40], the mutation rate which gives the best performance in terms of number of generations is decreased to 0.03. So, the best mutation rate should be chosen according to the complexity of the task to be solved. The complexity of those circuits in evolvable hardware is mainly dependant on the number of inputs rather than outputs [40].

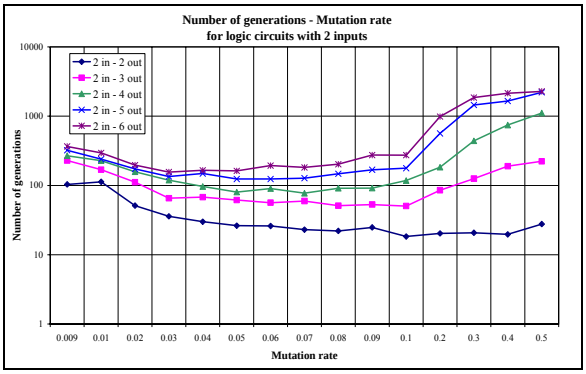

Fig. 11 Average out of 100 experiments of the number of generations required to completely evolve a logic circuit by changing the mutation rate. The circuits are with 2 inputs and varying numbers of outputs

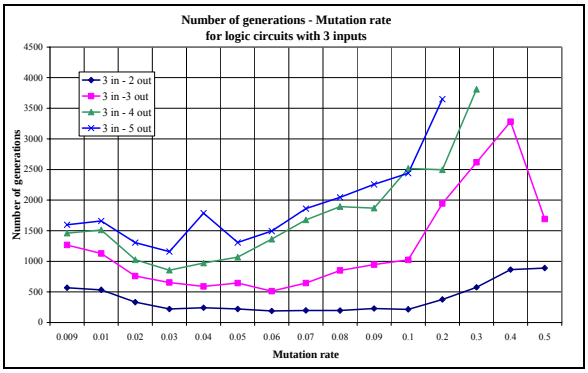

Fig. 12 Average out of 100 experiments of the number of generations required to completely evolve a logic circuit by changing the mutation rate for evolving logic circuits with 3 inputs. The solutions are given only for the circuits which are 100% evolved

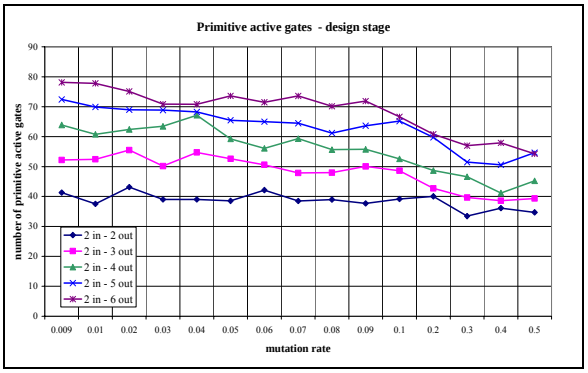

Fig. 13 Average out of 100 experiments of the primitive active gates required for the design stage, before the logic circuits are optimized. The solutions are given only for the circuits which are 100% evolved. The circuits are with 2 inputs and varying number of outputs

Moreover it may be noticed in Fig. 12, that if the mutation rate is very high (more than 0.3) the evolution of more complex tasks is not performed. This is because of the high randomness introduced in the chromosome. Therefore, the random processes become dominant under the evolutionary process if a mutation rate higher than 0.3 is used.

Table IV illustrates the quality of the evolved logic circuits based on redundancy (before the optimization stage). The circuits with higher redundancy are those obtained with very high mutation rates. These results place us in contrast to the previously obtained results which advise the use of smaller values of mutation rate in order to find a solution with fewest numbers of generations. Therefore at this point the best trade-off between a small mutation rate for finding the final circuit's configuration with the fewest number of generations and a high mutation rate for finding the better-optimized circuits should be determined. However, before the final solution is outlined, the number of active logic gates required during design and optimization should be analyzed Fig. 13 and Fig. 14 show the number of active logic gates by changing the mutation rate when the circuits are designed.

The experimental results demonstrate that the circuits with fewer logic gates are those created with high mutation rates, between 0.1 and 0.3. In Table V the number of active logic gates used after the optimization stage is presented. In that table the solutions with the smallest number of active logic gates are highlighted.

One may notice that the best solutions are achieved by decreasing the mutation rate as the complexity of the logic circuits increases. Based on those results one may conclude that the mutation rate for the better optimized circuit in terms of logic gates used is inversely proportional to the complexity of the evolved circuit. Therefore, supposing that a very simple circuit should be solved, the best mutation rate to be chosen in order to have the best result in terms of number of logic gates should be between 0.3 and 0.5.

If the circuits are more complex, that value (according with the experimental results shown in Table V) should be reduced to between 0.02 and 0.04.

TABLE IV
SIMULATION RESULTS: REDUNDANCY OF EVOLVED RANDOMLY GENERATED LOGIC CIRCUITS. N.E. REFERS TO CIRCUITS WHICH ARE NOT EVOLVED WITH SUCCESS RATE EQUAL TO 100%. IN EACH CELL THE AVERAGE VALUES OF THE REDUNDANCY CALCULATED USING 100 EXPERIMENTS ALL COMPLETELY EVOLVED ARE REPORTED. THE BEST CIRCUIT'S CONFIGURATION FOR EACH LOGIC CIRCUIT IS HIGHLIGHTED

Redundancy of evolved logic circuits																	
Logic circuits			**Mutation rate**														
in	**out**	**p**	**0.009**	**0.01**	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
2	**2**	**4**	0.816	0.830	0.809	0.824	0.824	0.829	0.811	0.826	0.824	0.829	0.829	0.819	0.851	0.840	0.842
2	**3**	**8**	0.762	0.759	0.750	0.771	0.752	0.758	0.769	0.777	0.776	0.769	0.772	0.801	0.812	0.816	0.817
2	**4**	**16**	0.711	0.729	0.717	0.710	0.700	0.731	0.742	0.730	0.744	0.748	0.756	0.777	0.779	0.809	0.789
2	**5**	**32**	0.671	0.686	0.683	0.687	0.692	0.700	0.703	0.703	0.720	0.713	0.700	0.723	0.765	0.763	0.749
2	**6**	**64**	0.646	0.644	0.650	0.675	0.674	0.665	0.672	0.668	0.680	0.668	0.691	0.715	0.730	0.734	0.748
3	**2**	**4**	0.790	0.788	0.790	0.809	0.808	0.798	0.813	0.814	0.811	0.818	0.816	0.845	0.850	0.868	0.869
3	**3**	**8**	0.738	0.735	0.726	0.740	0.740	0.740	0.740	0.752	0.748	0.775	0.781	0.805	0.837	0.831	0.737
3	**4**	**16**	0.685	0.688	0.691	0.704	0.700	0.707	0.709	0.738	0.732	0.723	0.733	0.802	0.755	N.E.	N.E.
3	**5**	**32**	0.635	0.657	0.655	0.676	0.681	0.677	0.693	0.687	0.706	0.703	0.694	0.690	N.E.	N.E.	N.E.
3	**6**	**64**	0.638	0.623	0.654	0.652	0.650	0.665	0.674	0.703	0.686	0.715	0.711	0.773	N.E.	N.E.	N.E.

TABLE V
SIMULATION RESULTS OF THE MUTATION RATE FOR EVOLVING RANDOMLY GENERATED LOGIC. N.E. REFERS TO CIRCUITS WHICH ARE NOT EVOLVED WITH SUCCESS RATE EQUAL TO 100%. IN THIS TABLE THE BEST SOLUTIONS (THOSE WITH THE SMALLEST AMOUNT OF LOGIC GATES REQUIRED) ARE HIGHLIGHTED. IN EACH CELL THE AVERAGE VALUES OF THE REDUNDANCY CALCULATED USING 100 EXPERIMENTS IS REPORTED

Average of number of active logic gates after optimization stage																	
Logic circuits			**mutation rate**														
in	**out**	**p**	**0.009**	**0.01**	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
2	**2**	**4**	2.75	2.45	2.11	2.17	2.04	2.08	2.01	2.00	2.00	2.01	2.01	2.00	2.00	2.00	2.00
2	**3**	**8**	4.36	4.25	3.51	3.40	3.23	3.16	3.14	3.27	3.06	3.10	3.06	3.05	3.03	3.38	4.25
2	**4**	**16**	6.23	5.56	4.69	4.32	4.33	4.37	4.18	4.16	4.15	4.12	4.10	4.11	7.41	10.57	17.59
2	**5**	**32**	7.24	6.98	5.95	5.46	5.45	5.35	5.27	5.38	5.36	5.33	5.24	12.61	29.19	27.96	41.65
2	**6**	**64**	8.21	7.52	6.59	6.35	6.21	6.13	6.10	5.87	5.78	6.07	5.88	16.97	36.00	43.48	49.00
3	**2**	**4**	10.43	10.05	8.49	7.42	7.62	7.69	7.12	7.11	6.86	7.18	7.39	7.69	8.55	10.77	12.55
3	**3**	**8**	16.84	16.50	13.37	11.25	11.05	10.57	10.85	10.88	11.81	11.45	11.66	24.07	26.44	36.40	50.00
3	**4**	**16**	24.18	21.78	17.66	16.70	16.61	18.52	19.71	24.96	27.70	32.23	36.84	33.00	55.00	N.E.	N.E.
3	**5**	**32**	26.99	24.95	20.64	20.77	21.12	20.98	21.75	24.63	28.65	30.71	39.62	36.33	N.E.	N.E.	N.E.
3	**6**	**64**	26.82	29.54	22.11	22.13	26.15	29.09	27.75	33.75	36.86	36.61	37.38	47.33	N.E.	N.E.	N.E.

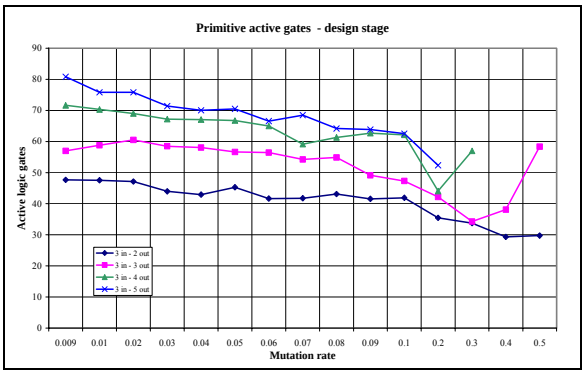

Fig. 14 Average out of 100 experiments of the primitive active gates required for the design stage, before the logic circuits are optimized. The solutions are given only for the circuits which are 100% evolved. The circuits are with 3 inputs and varying number of outputs

By taking into account the results found in terms of number of generations, redundancy and number of active logic gates used for getting the optimized solutions, one may conclude that the mutation rate should be chosen according to the complexity of the task to be evolved and with the wishes of the user: less generations, fewer logic gates or good redundancy. Simpler tasks should be solved with a higher mutation rate. By increasing the complexity of the task the mutation rate should decreased to 0.02-0.04.

V. CONCLUSION

This paper describes how the mutation rate for an evolvable hardware system should be chosen in order to solve and better optimize the evolution of logic circuits. It should be noted that the mutation rate for EHW systems modifies the logic cell inputs, the cell type (for example from AND to NOR) and the system output. The experimental results found prove that the mutation rate should be inversely proportional to the complexity of logic circuits: more complex circuits require a smaller mutation rate. Therefore, these results are especially important for all researchers who are using decomposition strategies for evolving logic circuits; because they could implement a dynamic mutation rate which changes the behavior in real-time based on the complexity of the decomposed task. Further work will be focused on exploring the evolution of bigger logic circuits together with the use of different population sizes. This will be done in order to find the best set up for all the parameters in the evolutionary algorithms for designing logic circuits.

REFERENCES

[1] X. Yao, T. Higuchi; "Promises and challenges of evolvable hardware" *IEEE Trans. Systems, Man and Cybernetics, Part C*, volume 29, pp. 87 - 97, February 1999.
[2] H. de Garis. "Evolvable Hardware: Principles and Practice". *Communications of the Association for Computer Machinery (CACM Journal)*. August 1997.
[3] D. E. Goldberg. Genetic algorithm in search, optimization and machine learning. Addison-Wesley Publishing Company, Incorporated, Reading, Massachusetts, 1989.
[4] J. Holland. Adaptation in Natural and Artificial Systems. Ann Arbor, MI: University of Michigan Press, 1975.
[5] M. D. Vose. "The Simple Genetic Algorithm". MA: MIT Press 1999.
[6] J. R Koza. Genetic Programming: On the Programming of Computers by Means of Natural selection. ISBN 0-262-11170-5. MIT Press, 1992.
[7] I. Rechenberg, "Evolution Strategy", in J. Zurada, R. Marks II, and C. Robinson (Eds.), *Computational Intelligence: Imitating Life*, 1994, pp. 147-159.
[8] H. G. Beyer and H. P. Schwefel, "Evolution strategies: A comprehensive introduction," *Natural Computing: an international journal*. Volume 1, Issue. 1, pp. 3–52, 2002.
[9] T. Bäck, Evolutionary Algorithms in Theory and Practice. New York: Oxford Univ. Press, 1996.
[10] H.-P. Schwefel, Numerical Optimization of Computer Models. New York: Wiley, 1981.
[11] C. Ryan, J. J. Collins, M. O' Neill; "Grammatical Evolution: Evolving Programs for an Arbitrary Language". Lecture Notes in Computer Science 1391. *First European Workshop on Genetic Programming* 1998.

[12] L. J. Fogel, A. J. Owens, M. J. Walsh. Artificial Intelligence through Simulated Evolution. New York: Wiley, 1966.
[13] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs. Berlin, Germany: Springer-Verlag, 1994.
[14] J. F. Miller and P. Thomson. "Cartesian genetic programming". In Riccardo Poli, Wolfgan Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin and Terence C. Forgaty, editors. *Genetic Programming, Proceedings of EuroGP 2000*. Vol. 1802 of LNCS, pages 121-132, Edinburg, 16 April 2000. Springer-Verlag.
[15] H. Mühlenbein. "Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization". In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms,* pp 416-421, San Mateo, 1989. Morgan Kaufman.
[16] Myung-Sook Ko, Tae-Won Kang and Chong-Su Hwang. "Function optimisation using an adaptive crossover operator based on locality". *Eng. Applic. Artif. Intell.* Vol. 10, No 6 pp. 519-524, 1997.
[17] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. 6th Symp. MicroMachine and Human Science*, Nagoya, Japan, 1995, pp. 39–43.
[18] K. Y. Chan, M. E. Aydin, T. C. Fogarty; "Parameterisation of mutation in evolutionary algorithms using the estimated main effect of genes" *Congress on Evolutionary Computatio. CEC2004*. ,Volume: 2 , 19-23 June 2004 Pages:1972 – 1979.
[19] K. Y. Chan, M. E. Aydin, T. C. Fogarty; "An epistasis measure based on the analysis of variance for the real-coded representation in genetic algorithms" *Congress on Evolutionary Computation. CEC '03*. Vol.: 1 , 8-12 Dec. 2003 Pages:297 – 304.
[20] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Systems, Man, Cybernetics*. Vol. 16, no. 1, pp. 122–128, 1986.
[21] K. De Jong, "The analysis of the behavior of a class of genetic adaptive systems." Ph.D. dissertation, Dept. Computer Science, University of Michigan, Ann Arbor, 1975.
[22] A. E. Eiben, R. Hinterding, Z. Michalewicz; "Parameter control in evolutionary algorithms" *IEEE Transactions on Evolutionary Computation*, Volume: 3, Issue: 2, July 1999 Pages:124 – 141.
[23] J. D. Schaffer, R. Caruana, L. Eshelman and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization." *Proceedings of the Third International Conference on Genetic Algorithms*, ed. J. D. Schaffer, Los Altos, CA: Morgan Kaufmann, June 4-7, 1989, pp. 51-60.
[24] H. Mühlenbein. "How genetic algorithms really work: I.Mutation and Hillclimbing," in *Parallel Problem Solving from Nature- PPSN II*, R. Männer and B. Manderick, Eds., Amsterdam, The Netherlands, 1992, pp. 15-25.
[25] M. Srinivas, L. M. Patnaik. "Adaptive probabilities of crossover and mutation in genetic algorithms". *IEEE Transactions on Systems, Man and Cybernetics*, Volume 24, Issue 4, April 1994 Page(s):656 - 667
[26] T. Niwa, M. Tanaka. "On the mean convergence time for simple genetic algorithms". *IEEE International Conference on Evolutionary Computation*. Volume 1, 29 Nov.-1 Dec. 1995 Page(s):373.
[27] R. L. Haupt. "Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors". *IEEE International Symposium Antennas and Propagation Society*. Volume: 2, 16-21 July 2000. Pages:1034 – 1037
[28] S. Nijssen, T. Back; "An analysis of the behaviour of simplified evolutionary algorithms on trap functions". *IEEE Transactions on Evolutionary Computation.* Volume: 7, Issue: 1, Feb. 2003. Pages:11 – 22.
[29] M. Srinivas, L. M. Patnaik; "Genetic algorithms: a survey". *IEEE JNL Computer*, Volume: 27, Issue: 6, June 1994. Pages:17 - 26
[30] T. Kalganova, J. Miller, "Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness". *Proc. of the First NASA/DoD Workshop on Evolvable Hardware. IEEE Computer Society*, Pages 54–63. July 1999
[31] T. Kalganova; "Bidirectional incremental evolution in extrinsic evolvable hardware". *Proc. of the Second NASA/DoD Workshop on Evolvable Hardware. IEEE Computer Society*, 13-15 July 2000. Pages:65 – 74
[32] E. H. Luna, C.A. Coello Coello, A.H. Aguirre. "On the use of a population-based particle swarm optimizer to design combinational logic circuits". *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, 24-26 June 2004. Pages:183 – 190.
[33] S. Balkir, G. Diindar, G. Alpaydin,; "Evolution based synthesis of analog integrated circuits and systems" *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, 24-26 June 2004 Pages:26 – 29.
[34] M. Oltean, C. Grosan; "Evolving digital circuits using multi expression programming" *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, 24-26 June 2004 Pages:87 – 94.
[35] Yang Zhang, S.L. Smith, A.M. Tyrrell;"Digital circuit design using intrinsic evolvable hardware" *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, 24-26 June 2004 Pages:55 – 62
[36] J.C. Gallagher, S. Vigraham, G. Kramer; "A family of compact genetic algorithms for intrinsic evolvable hardware". *IEEE Transactions on Evolutionary Computation*, Volume: 8 , Issue: 2 , April 2004 Pages:111 – 126.
[37] J. Miller. "An empirical study of the efficiency of learning Boolean functions using a Cartesian genetic programming approach" *In Proc. of the Genetic and Evolutionary Computation Conference.* Volume 1, pp. 1135–1142, Orlando, USA, July 1999.
[38] T. Bäck, F. Hoffmeister, H. P. Schwefel. "A survey of evolutionary strategies". In R. Belew and L. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, 1991. Morgan Kaufmann.
[39] H. P. Schwefel. Numerical Optimization of Computer Models. John Wiley & Sons, Chichester, UK, 1981.
[40] E. Stomeo and T. Kalganova. "Improving EHW performance introducing a new decomposition strategy." 2004 *IEEE Conference on Cybernetics and Intelligent Systems*. Singapore 1-3 December 2004, pp. 439-444.