International Letters of Natural Sciences
ISSN: 2300-9675, Vol. 5, pp 16-26
doi:10.18052/www.scipress.com/ILNS.5.16
© 2013 SciPress Ltd., Switzerland
Online: 2013-11-16

New hardware engine for new operating systems

Nazjla Ahmadi[1,*], **Mehrad Kaveh**[2,**]

[1]Department of Computer Science, Islamic Azad University, Ghorveh Branch, Ghorveh, Iran

[2]Department of Computer Science, Islamic Azad University, Mianeh Branch, Mianeh, Iran

E-mail address: sanandajstudent@gmail.com , alimehrad@yahoo.com

RETRACTED

ABSTRACT

Genetic algorithm is a soft computing method that works on set of solutions. These solutions are called chromosome and the best one is the absolute solution of the problem. The main problem of this algorithm is that after passing through some generations, it may be produced some chromosomes that had been produced in some generations ago that causes reducing the convergence speed. From another respective, most of the genetic algorithms are implemented in software and less works have been done on hardware implementation. Our work implements genetic algorithm in hardware that doesn't produce chromosome that have been produced in previous generations. In this work, most of genetic operators are implemented without producing iterative chromosomes and genetic diversity is preserved. Genetic diversity causes that not only don't this algorithm converge to local optimum but also reaching to global optimum. Without any doubt, proposed approach is so faster than software implementations. Evaluation results also show the proposed approach is faster than hardware ones.

Keywords: Genetic algorithm; chromosome; Genetic diversity

1. INTRODUCTION

Genetic algorithm is a soft computing method that works on set of solutions. Each of these solutions are called chromosome and each population consists of a certain number of them. By applying some operators like selection, crossover, and mutation on the chromosomes of current population, the next generation is produced.

In the GA explained above, it is observed that by passing through a number of generations some chromosomes may be produced that are the same as the chromosomes in the previous generations. It is clear that these chromosomes are not suitable ones because they were eliminated in the previous generations because of low fitness value. The main problem of these chromosomes is increasing calculations of each generation because GA operators in current generation are applied on chromosomes that were produced and deleted in the previous generations [1].

This process also causes decreasing of convergence speed toward problem solutions. Assume two individuals in generation N; produce two offspring by applying crossover and mutation. Now in generation N+1, if these two offspring recombine together, it may be produced chromosomes that are similar to ones in generation N, consider the Figure 1 [1].

SciPress applies the CC-BY 4.0 license to works we publish: https://creativecommons.org/licenses/by/4.0/

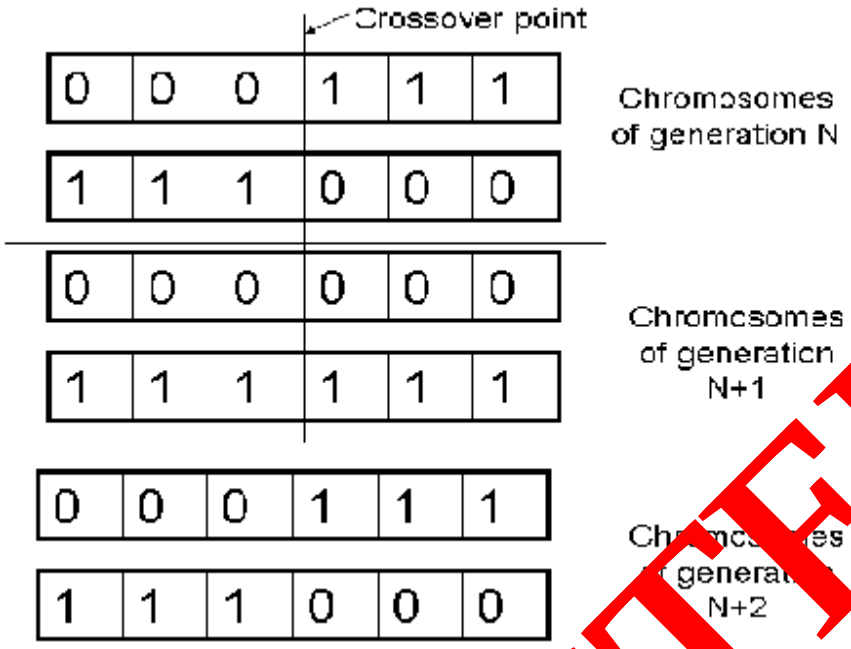


Figure 1. Producing of repeated chromosomes [1].

In [1] individuals are divided into two groups including chromosomes and ovums. In order to prevent from producing of iterative individuals, the recombination of the same sex individuals is forbidden. Also, the parent of each individual together with grandparent and third-ancestor are put in the individual's structure. Figure 2 shows the individuals structure presented in [1]. In this method, the following recombination is denied to avoid from producing of repeated individuals:

- parent (ovum) = = parent (chor)
- parent (ovum) = = grand parent (chor)
- parent (ovum) = = grand-grand parent (chor)
- ovum be in the previous generations of chor
- chor be in the previous generations of ovum
- parent (chor) = = parent (ovum)
- parent (chor) = = grand parent (ovum)
- parent (chor) = = grand-grand parent (ovum)
- chor = ovum

RETRACTED

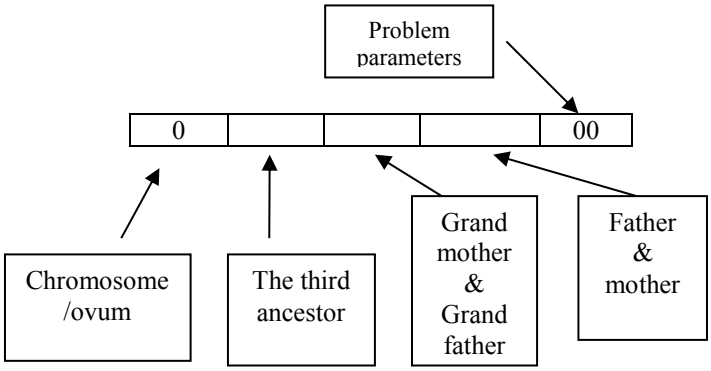


Figure 2. The structure of individuals.

Therefore, selection algorithms related to ovum and chromosome a[illegible] cha[illegible]d as fo[illegible]wing.

I. Select (Chromosome)

II. Move Chromosome to the interme[illegible]diate generation to crossover

Algorithm 1. Pseudo code [illegible]lating to c[illegible] [illegible]ne selection.

selection

Select([illegible]vum)

if [illegible]arent (ov[illegible] [illegible]arent (chromosome) OR

[illegible]arent (ovu[illegible] = grand parent (chromosome) OR

[illegible]aren[illegible]vum) = = [illegible]rand-grand parent (chromosome) OR

[illegible]hromosome = = ovum OR

vice versa

then

Repeat from first line 'select another ovum'

else

DELETE (ovum)

move ovum to the intermediate generation to be crossover with chor that selected in algorithm 1

Algorithm 2. Pseudo code relating to ovum.

Also in [1], the recombination method put the information of recombined individuals into next stage of produced individuals. Figure 3 shows the process of crossover method.

RETRACTED

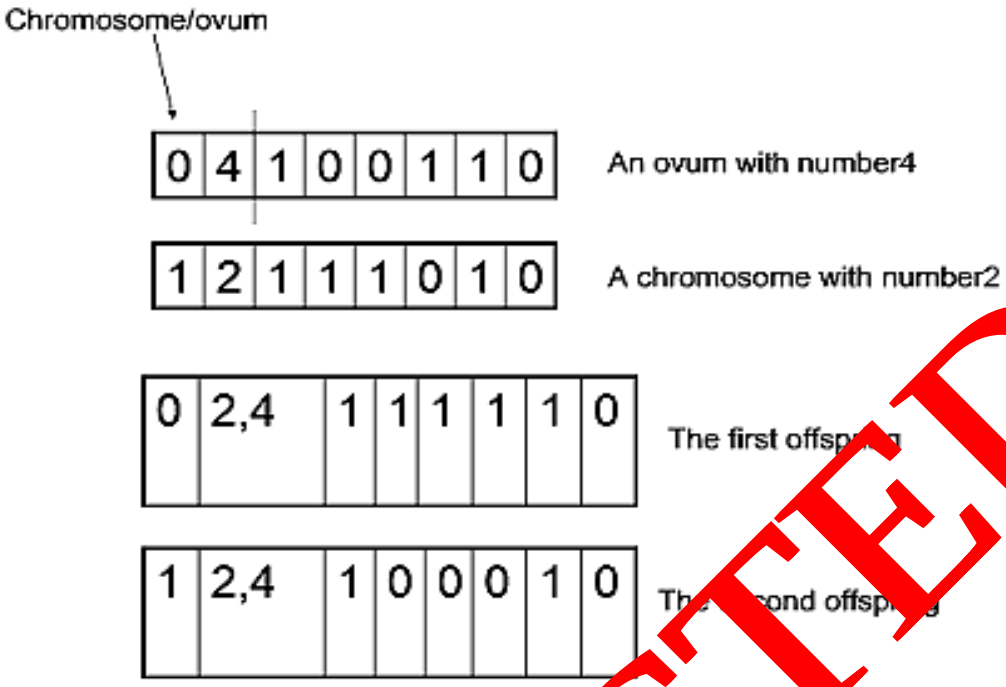


Figure 3. Recombination operation in proposed approach [1].

RETRACTED

For another respective, almost all of the genetic algorithms are implemented in software. But in this work, we propose parallel architecture of hardware for above mentioned genetic algorithm and compare it with previous works. The evaluation results show the usefulness of the researchers enterprise work. Previous works [1,2,4,5] can be divided into three categories. 1- Those that implement fitness function on hardware and other parts of genetic algorithm like crossover, mutation and selection in software. The disadvantage of this method is that hardware depends on a problem. 2- Those that implement fitness on software and other operators in hardware. In this method hardware is independent of a problem but the running time is increased [6,7]. 3- Those that implement both fitness and operators in hardware [2,4,16]. In this work we proposed an approach that improves this type of method for new approach to standard genetic algorithm [1]. This paper divided into 5 sections. In section 2, the proposed approach is described and in section 3, architectures of genetic operators are explained. Sections 4, 5 evaluate the results and draw some conclusion, respectively.

2. OVERALL VIEW OF PROPOSED ARCHITECTURE

The difference of genetic algorithm with other soft-computing methods is individuals' representation. Individuals in genetic algorithm are a stream of binary bits that are very attractive to hardware implementation. The selection, crossover, and mutation are generic operators and never depend on the problem types. These characteristic make them easy to implement. The main problem of implementing of genetic algorithm is fitness function. For this reason, to calculate it, hardware of neural networks is used [2,9,10]. Ideal estimation of each individual can be obtained using this network. Another reason to use neural network to compute fitness is its simplicity and stochastic representation of signals that reduces the

hardware area. In figure4 the overall view of proposed approach is showed. Genetic operators of this architecture are as follow.

- Selection operator: Rolette wheel & tournament
- Crossover operator: single point & two-point
- Mutation operator: bit complement
- Replacement operator: steady state & generalization

The main and novel contribution of this work consists of using all genetic operators in hardware implementation without producing repeated individuals in alternative generations. Hardware implementations of these operators are described in section 3. The main advantage of this architecture over previous works [16] is using fitness of next generation in replacement operator to avoid re-computing of fitness function in each generation. For this reason, fitness of each chromosome is kept in bank of registers and individuals' fitness of next generation is provided as input. It is necessary to remind that each register of register bank involve all part of individual structure mentioned above. When replacement operator is steady state and the fitness of input chromosomes are greater than ones that kept in the register bank, the input chromosomes are replaced. It should be noted that Fitness of each chromosome in the first population is considered as 0 [1].

RETRACTED

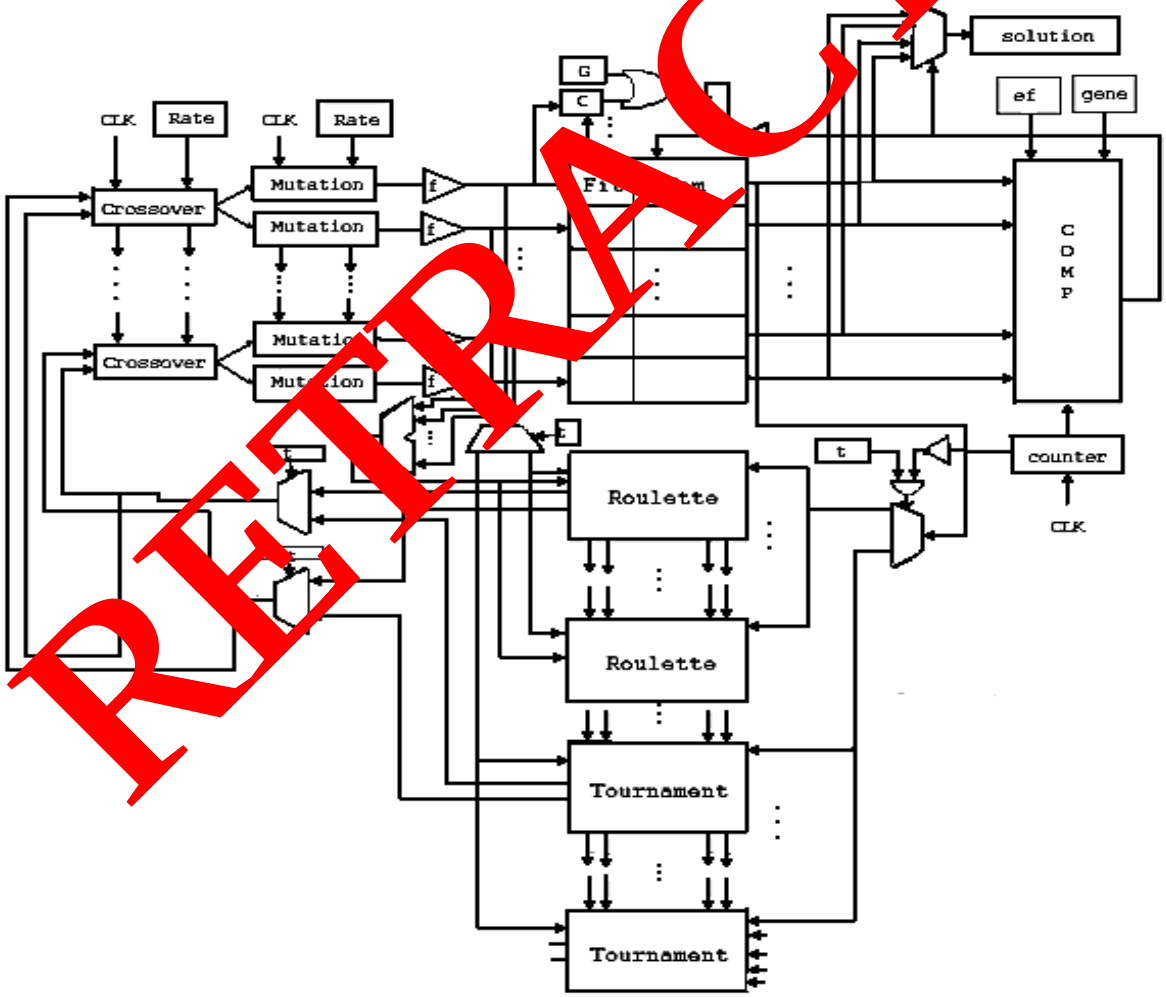


Figure 4. Overall view of architecture [2].

3. OPERATORS ARCHITECTURE

The architectures of selection, crossover, and mutation operators and shard memory are described in this section.

A. shared memory for generational population

All of the chromosomes are kept in synchronized bank of registers. Each register keeps individuals as described in Figure 2. These registers can be updated and written. The registers can be writte by the following steps.

1- Whenever the comparator output is 0 and replacement operator is generality

2- Whenever the comparator output is 0 and replacement operator is steady state and fitness measurements of chromosome in next generation are greater than one in current generation.

It is necessary bearing in mind that the output of comparator is 0 if the number of generation (*gene*) is greater than *counter* or fitness of all chromosomes less than *fitness*. In such condition genetic algorithm continues to work and current generation replaced by next generation.

B. Random number generator

This generator is used to produce crossover & mutation probability. In this work *linear feedback shift register* are used. More details about random number generator can be found in [11,12].

C. Roulette selection components

In this subsection the roulette operator is elaborated. Inputs of this component are generational chromosomes ($I_1, I_2, \ldots, I_n$), fitness function ($F_1, F_2, \ldots, F_n$), and cumulative sum of these fitness. This component selects two individuals and passes it to crossover unit. The hardware of this component is easy to implement. for more details see [1,16].

D. Tournament selection component

One of the novel contributions of this work is tournament operator. This component takes several chromosomes together with their fitness and their information and output the fittest one that not satisfy any of the following conditions.

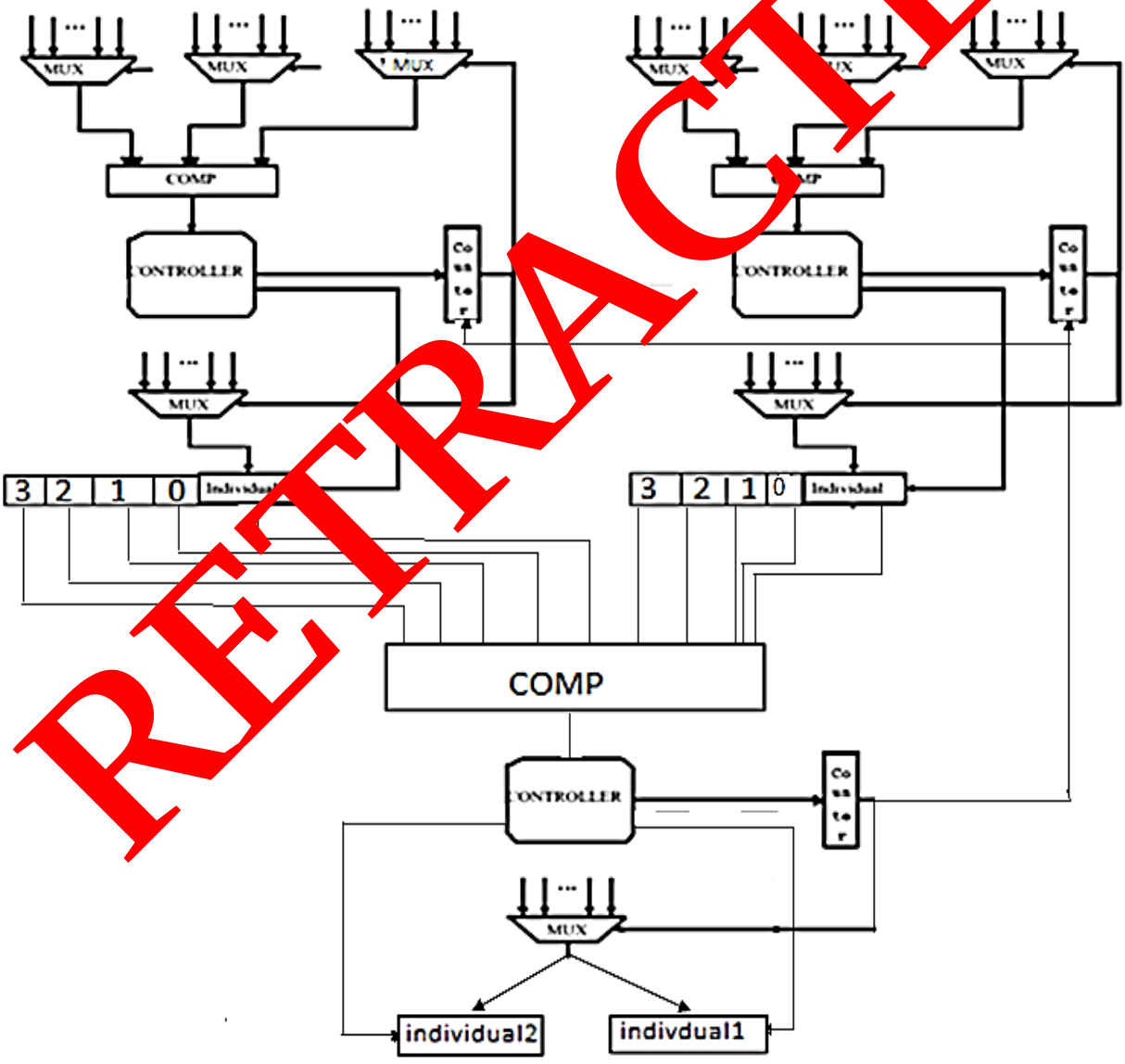
- parent (ovum) = = parent (chor)
- parent (ovum) = = grand parent (chor)
- parent (ovum) = = grand-grand parent (chor)
- ovum be in the previous generations of chor
- chor be in the previous generations of ovum
- parent (chor) = = parent (ovum)
- parent (chor) = = grand parent (ovum)
- parent (chor) = = grand-grand parent (ovum)
- chor = ovum

Algorithm 3 shows the hardware algorithm of this unit.

RETRACTED

1. Choose 3 individual in the population
2. Compare them according to their fitness's
3. Output the best one to individual1
4. repeat step 1 to 3 to select the second one
5. If two selected individuals doesn't satisfy any of the conditions, put them into final individual1 and final individual 2

Algorithm 3. Algorithm of tournament selection.



RETRACTED

Figure 5. Architecture of tournament selection.

In this algorithm, three chromosomes are selected and raced to be one of the individual intermediate generations. This routine is also repeated to select the second chromosome. The hardware implementation of this component is shown in Figure 5. Finally, if the selected individuals don't have any above mentioned properties, they are selected and put into final individual1 and final individual 2.

The synchronized bank of registers that keep the individuals are input of the multiplexers, *comp* unit compare the fitness of these chromosomes and also relationships between them. After passing from these steps, comp unit outputs the number of those individuals. This number is applied to selection signal of multiplexer in the middle of the Figure 5. In other word, the appropriate chromosome is selected and sent to individual 1. The same process is repeated for selecting ovum. After that, the properties of two individuals are compared to judge about their relationship. In this research iterative process is implemented using state machine (controller) [1,16].

E. Crossover component

Another contribution of this work is implementing a unit for new crossover using both one point and also two-point recombination. To crossover, firstly, the Random Number Generator produce a random number say *p*. if $p < \mu_{rate}$ then crossover operator is applied. In the case that crossover applied, the bits of the less significant half of the randomized number is used as the first crossover point and the most significant part as the second one. It should be reminded that in the single-point crossover, the bits of the most significant part are considered as 0. Finally, the information of selected individuals are shifted to next stage of produced individuals like as Figure 2. Figure 6 shows the hardware implementation of crossover.

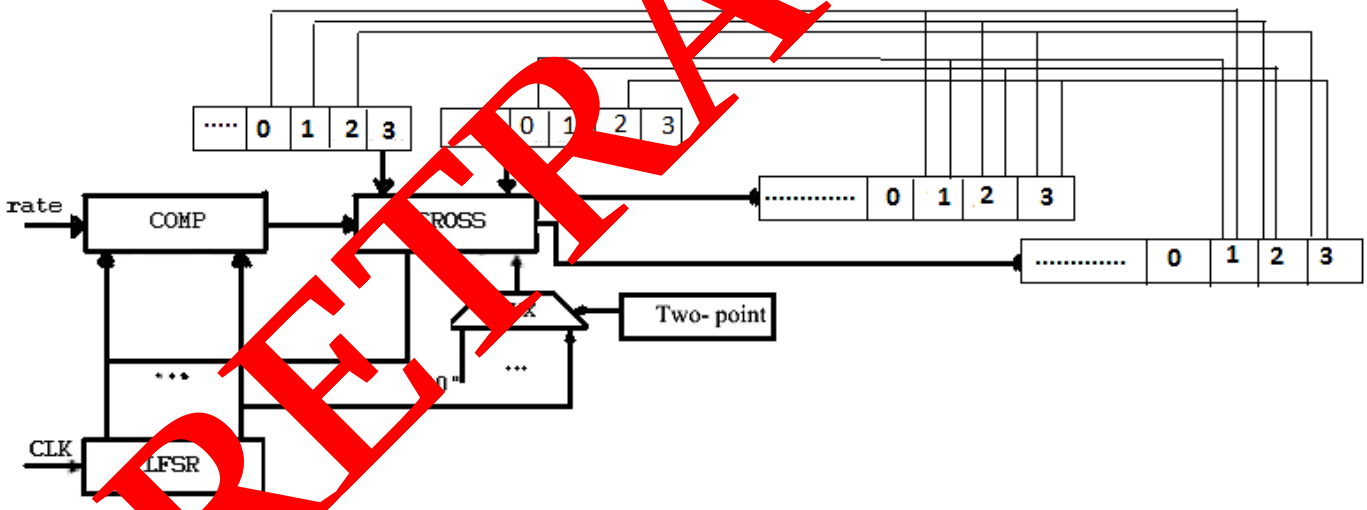


Figure 6. Hardware of crossover operator.

In this hardware if the value of *two-point* register be one, two-point crossover is applied; otherwise, one-point crossover is applied.

F. mutation components

This component has architecture like as crossover. To implement this component, a random number is produced using Random Number Generator. When this value is less than μ_{rate}, the mutation operator is applied. For more detail see [1,16].

RETRACTED

G. Fitness components

For the reason that the fitness function will not be depending on a problem, the neural network is used to estimate fitness measure [10] in this work. In [10] neural network has been implemented using stochastic signals and therefore reduces very significantly the hardware area required for the network. For the genetic hardware implementation, the number of input neurons is the same as the size of individuals in population. The output neuron is augmented with a shift register to store the final result. The training phase is supposed to be performed before the first use within the hardware genetic algorithm.

4. EVALUATION RESULTS

Hardware for new genetic algorithm was simulated and then programmed into a Sparatan3 Xilinx FPGA [13]. To assess and evaluate of proposed architecture the function in [14] was used to maximize. This function was also used in previous works to evaluate their hardware implementation for genetic algorithm. This function is as follows:

$$F(x,y) = 21.5 + x\sin(4\Pi x) + y\sin(2\Pi y) \quad (1)$$
$$-3.9 \leq x \leq 12.1$$
$$4.1 \leq y \leq 5.8$$

It is clear that this function is not easy to maximize and is a very attractive metric to evaluate and assess of proposed approach. The evaluation results are shown in Table 1.

Table1. Evaluation results.

Implementation	Time	Area	Area*Time	Absolute Answer
Software approach	42[illegible]0	0	0	32.21
[15]	1012	850	860200	38.8211
[5]	212	1943	411916	38.8214
[2]	194	2016	391104	38.848912
Proposed Approach	181	2114	382634	38.8476

RETRACTED

In this table the area is expressed in terms of CLBs and the time is in second, also previous works including the software and hardware implementation and our proposed approach have been compared.

The advantage of hardware approach to software approach is speed up. Therefore, in this work, besides software implementation, hardware implementations are studied and evaluated. It is clear that the area required by this work is more than [5,15,16]. From another respective, our proposed hardware is faster than [1,15] and also approximately is faster than [16]. The main advantage of proposed approach over [15,16] is applying almost all of the

genetic operators that result in genetic diversity. In the [16] only roulette selection, two-point crossover and mutation have been implemented, but in our work and [1] not only have been these operators implemented but also tournament, one-point crossover, and steady state replacement have been designed and implemented. In our work, unlike [1,5,15,16] producing of the iterative chromosome is avoided that causes this approach to increase in convergence speed in comparison to other approaches [1,5,15,16]. In our work like [1] the operators can be changed in the next generation by changing a value of corresponding registers i.e. genetic diversity is respected. For example, in one generation, tournament selection, two-point crossover, and general replacement may be used and in next generation roulette selection, one-point crossover, and steady-state replacement be used. This diversity is implemented neither in [15] nor in [16].

5. CONCLUSION

Genetic algorithm is an attractive method that used to solve NP-Hard problems. It is observed that by passing through some generation, it may be reproduced some chromosome that were produced in some generation ago. Therefore some methods are required to solve this problem. By putting ancestor information of each produced chromosome in its structure, this difficulty can be curable. From another respective, genetic algorithm is usually implemented in software and less works have been done to implement this algorithm in hardware. The main and novel contribution of this work is implementing almost all of the genetic operators that results in genetic diversity without producing of iterative individuals. In previous researches that work on hardware genetic algorithm, some of these operators are implemented and some of them are not implemented in hardware. To evaluate fitness measurements, neural network has been used and shown area required using this approach is significantly decreased in compare to previous works. In spite of previous works, proposed architecture preserve genetic diversity that causes to speed up in our architecture. From evaluation results can be drawn a conclusion that by changing genetic operators in specific conditions, convergence speed is significantly increased. For example when some chromosome are produced in alternate generations, changing one point crossover to two-point decreases production of these chromosome and improve genetic diversity and convergence speed. Evaluation results show advantage of our proposed architecture over previous works.

RETRACTED

References

[1] Faribo Ahmadi, Amir Shikh Ahmadi, *Intenational Journal of Computer Applications* 32 (2011) 46-50.

[2] Faribo Amadi, Reza Tati, *New hardware engine for genetic algorithm*, In Proc 5th International Conference on Genetic and Evolutionary Computing, 2012

[3] Liu J., *A general purpose hardware implementation of genetic algorithms*, MSc. Thesis, University of North Carolina, 1993.

[4] Scott S. D., Samal A., Seth S., *HGA: a hardware-based genetic algorithm*, In Proc. ACM/SIGDA 3rd. International Symposium in Field-Programmable Gate Array, pp. 53-59, 1995.

[5] Turton B. H., Arslan, T., *A parallel genetic VLSI architecture for combinatorial real-time applications – disc scheduling*, In Proc. IEE/IEEE International Conference on genetic Algorithms in Engineering Systems, pp. 88-93, 1994.

[6] Bland I. M., Megson G. M., Implementing a generic systolic array for genetic algorithms. In Proc. 1st. On-Line Workshop on Soft Computing, pp 268-273, 1996.

[7] Megson G. M., Bland I. M., *Synthesis of a systolic array genetic algorithm*. In Proc. 12th. International Parallel Processing Symposium, pp. 316–320, 1998.

[8] D. C. Goldberg. *Genic algorithm in search, optimization, and machine learning* Addison Welsey, 1989.

[9] Gaines B. R., *Advances in Information Systems Science* 2 (1969) 37-172.

[10] Nedjah, N., Mourelle, L.M., *Lecture Notes in Computer Science* 2657 (2003)1[illegible]

[11] Bade S. L. M., Hutchings B. L., *FPGA-Based Stochastic Neural Networks – Implementation*, IEEE Workshop on FPGAs for Custom Computing Machines, Napa Ca, April 10-13, pp. 189-198, 1994.

[12] Brown B. D., Card H. C., *IEEE Transactions on Computers* 5[illegible]) (2001) 891-905.

[13] Xilinx, http://www.xilinx.com/, 2004.

[14] Michalewics Z., *Genetic algorithms + data structures = evolution programs*, Springer-Verlag, Berlin, Second Edition, 1994.

[15] Scott S. D., Seth S., Samal A., *A hardware engine for genetic algorithms*, Technical Report, UNL-CSE-97-001, University of Nebraska-Lincoln, July 1997.

[16] N. Nedjah, *Pararllel evolutionary computations*, Springer 2006.

(Received 29 October 2013; accepted 04 November 2013)

RETRACTED