http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 13 (6): 1110-1117, 2014
ISSN 1812-5638 / DOI: 10.3923/itj.2014.1110.1117

Circuit Automatic Design Algorithm Base on Cultural Evolution Theory

[1,2]Hanmin Liu, [1]Xuesong Yan and [3,4]Qinghua Wu
[1]School of Computer Science, China University of Geosciences, Wuhan 430074, China
[2]Wuhan Institute of Ship Building Technology, Wuhan 430050, China
[3]Hubei Provincial Key Laboratory of Intelligent Robot, Wuhan Institute of Technology, Wuhan 430073, China
[4]School of Computer Science and Engineering, Wuhan Institute of Technology, Wuhan 430073, China

Abstract: In the circuit optimization design field, the circuit scale and the time of evaluate the circuit are the challenge problems. The traditional optimization algorithms cannot solve the two problems very well. In this paper, we propose an algorithm based on cultural evolution theory, this algorithm is a class of computational models derived from observing the cultural evolution process in nature. The new algorithm not only has the fast convergence speed but has the capability of global searching. For the case studies, this means has proved to be efficient and the experiment results show that the new means have got the better results in the circuit optimization design.

Key words: Circuit design, optimization, cultural evolution, cultural algorithm

INTRODUCTION

Candidate solutions to some problems are not simply deemed correct or incorrect but are instead rated in terms of quality and finding the candidate solution with the highest quality is known as optimization. Optimization problems arise in many real-world scenarios. Take for example the spreading of manure on a cornfield, where depending on the species of grain, the soil quality, expected amount of rain, sunshine and so on, we wish to find the amount and composition of fertilizer that maximizes the crop, while still being within the bounds imposed by environmental law.

Several challenges arise in optimization. First is the nature of the problem to be optimized which may have several local optima the optimizer can get stuck in, the problem may be discontinuous, candidate solutions may yield different fitness values when evaluated at different times and there may be constraints as to what candidate solutions are feasible as actual solutions to the real-world problem. Furthermore, the large number of candidate solutions to an optimization problem makes it intractable to consider all candidate solutions in turn, which is the only way to be completely sure that the global optimum has been found. This difficulty grows much worse with increasing dimensionality, which is frequently called the curse of dimensionality, a name that is attributed to Bellman see for example (Bellman, 1957). This phenomenon can be understood by first considering an n-dimensional binary search-space. Here, adding another dimension to the problem means a doubling of the number of candidate solutions. So the number of candidate solutions grows exponentially with increasing dimensionality. The same principle holds for continuous or real-valued search-spaces, only it is now the volume of the search-space that grows exponentially with increasing dimensionality. In either case it is therefore of great interest to find optimization methods which not only perform well in few dimensions, but do not require an exponential number of fitness evaluations as the dimensionality grows. Preferably such optimization methods have a linear relationship between the dimensionality of the problem and the number of candidate solutions they must evaluate in order to achieve satisfactory results, that is, optimization methods should ideally have linear time-complexity O(n) in the dimensionality n of the problem to be optimized.

Another challenge in optimization arises from how much or how little is known about the problem at hand. For example, if the optimization problem is given by a simple formula then it may be possible to derive the inverse of that formula and thus find its optimum. Other families of problems have had specialized methods developed to optimize them efficiently. But when nothing is known about the optimization problem at hand, then the No Free Lunch (NFL) set of theorems by Wolpert and Macready states that any one optimization method will be as likely as any other to find a satisfactory solution (Wolpert and Macready, 1997). This is especially important in deciding what performance goals one should have when designing new optimization methods and whether one should attempt to devise the ultimate optimization method which will adapt to all problems and perform well. According to the NFL theorems such an optimization method does not exist and the focus of this thesis will therefore be on the opposite: Simple optimization methods that perform well for a range of problems of interest.

Corresponding Author: Hanmin Liu, School of Computer Science, China University of Geosciences, Wuhan 430074, China

Evolutionary Electronics applies the concepts of genetic algorithms to the evolution of electronic circuits. The main idea behind this research field is that each possible electronic circuit can be represented as an individual or a chromosome of an evolutionary process, which performs standard genetic operations over the circuits. Due to the broad scope of the area, researchers have been focusing on different problems, such as placement, Field Programmable Gate Array (FPGA) mapping, optimization of combinational and sequential digital circuits, synthesis of digital circuits, synthesis of passive and active analog circuits, synthesis of operational amplifiers and transistor size optimization. Of great relevance are the works focusing on "intrinsic" hardware evolution in which fitness evaluation is performed in silicon, allowing a higher degree of exploration of the physical properties of the medium. This particular area is frequently called Evolvable Hardware (EH) (Salem-Zebulum *et al*., 2002; Thompson and Layzell, 1999; Louis and Rawlins, 1991; Ferreira, 2001).

In the sequence of this work, Coello, Christiansen and Aguirre presented a computer program that automatically generates high-quality circuit designs (Koza, 1992). They use five possible types of gates (AND, NOT, OR, XOR and WIRE) with the objective of finding a functional design that minimizes the use of gates other than WIRE (essentially a logical no-operation).

Miller, Thompson and Fogarty applied evolutionary algorithms for the design of arithmetic circuits. The technique was based on evolving the functionality and connectivity of a rectangular array of logic cells, with a model of the resources reflecting the Xilinx 6216 FPGA device (Coello *et al*., 1996). Kalganova, Miller and Lipnitskaya proposed another technique for designing multiple-valued circuits. The EH is easily adapted to the distinct types of multiple-valued gates, associated with operations corresponding to different types of algebra and can include other logical expressions (Miller *et al*., 1997). In order to solve complex systems, Torresen (1998) proposed the method of increased complexity evolution. The idea is to evolve a system gradually as a kind of divide-and-conquer method. Evolution is first undertaken individually on a large number of simple cells. The evolved functions are the basic blocks adopted in further evolution or assembly of a larger and more complex system (Kalganova *et al*., 1998).

More recently Hollingworth, Smith and Tyrrell describe the first attempts to evolve circuits using the Virtex Family of devices. They implemented a simple 2-bit adder, where the inputs to the circuit are the two 2-bit numbers and the expected output is the sum of the two input values (Torresen, 1998). Based on the Miller's method, Yan applied Gene Expression Programming (GEP), Particle Swarm Optimization Algorithms (PSO) and Cultural Algorithms (CA) for the design of electronic circuits (Yan *et al*., 2006, 2007, 2010, 2011a, b).

A major bottleneck in the evolutionary design of electronic circuits is the problem of scale. This refers to the very fast growth of the number of gates, used in the target circuit, as the number of inputs of the evolved logic function increases. This results in a huge search space that is difficult to explore even with evolutionary techniques. Another related obstacle is the time required to calculate the fitness value of a circuit. Then the traditional optimization methods cannot solve these problems well and the researchers are wanted to find more efficiency method for solving these problems.

Cultural Algorithms (CA) proposed by Reynoids (1989). Cultural algorithm is in-depth analysis of the superiority of the original evolution theory on the basis of drawing on the social (cultural) evolution theory in the social sciences and has achieved broad consensus on the research results and proposed a new algorithm. Cultural algorithm is used to solve complex calculations of the new global optimization search algorithms, cultural algorithms in the optimization of the complex problems of its superior performance.

ALGORITHM BASED ON CULTURAL EVOLUTION THEORY

Evolutionary computation (EC) (Goldberg, 1989; Michalewicz, 1996) methods have been successful in solving many diverse problem in search and optimization due to the unbiased nature of their operations which can still perform well in situation with little or no domain knowledge. However, there can be considerable improvement in their performance when problem specific knowledge is used to bias the problem solving process in order to identify patterns in their performance environment. These patterns are used to promote more instances of desirable candidates or to reduce the number of less desirable candidates in the population. In either case, this can afford the system an opportunity to reach the desired solution more quickly.

Adaptive evolutionary computation takes place when an EC system is able to incorporate such information into its representation and operators in order to facilitate the pruning and promoting activities mentioned above. Some research works have shown that self-adaptation can take place on several levels within a system such as the population level, the individual level and the component level. At the population level, aspects of the system

parameters that control all elements of the population can be modified. At the individual level, aspects of the system that control the action of specific individual can be modified. If the individual is specified as s collection of components then component level adaptation is possible. This involves the adaptation of parameters that control the operation of one or more components that make up an individual.

In human societies, culture can be a vehicle for the storage of information in a form that is independent of the individual or individuals that generated and are potentially accessible to all members of the society. As such culture is useful in guiding the problem solving activities and social interaction of individuals in the population. This allows self-adaptive information as well as other knowledge to be stored and manipulated separately from the individuals in the social population. This provides a systematic way of utilizing self-adaptive knowledge to direct the evolution of a social population. Thus, cultural systems are viewed as a dual inheritance system where, at each time step, knowledge at both the population level and the level of acquired beliefs is transmitted to the next generation. This acquired knowledge is viewed to act as beacons by which to guide individuals towards perceived good solutions to problems and away from less desirable ones at a given time step. Cultural algorithms in order to model the evolution of cultural systems based upon principles of human social evolution taken from the social science literature.

Cultural algorithms are a class of computational models derived from observing the cultural evolution process in nature (Reynoids, 1989; Chung, 1997; Miller, 2003). In this algorithm, individuals are first evaluated using a performance function. The performance information represents the problem-solving experience of an individual. An acceptance function determines which individuals in the current population are able to impact, or to be voted to contribute, to the current beliefs. The experience of these selected individual is used to adjust the current group beliefs. These group beliefs are then used to guide and influence the evolution of the population at the next step, where parameters for self-adaptation can be determined from the belief space. Information that is stored in the belief space can pertain to any of the lower levels, e.g. population, individual, or component. As a result, the belief space can be used to control self-adaptation at any or all of these levels. The cultural algorithm is a dual inheritance system with evolution taking place at the population level and at the belief level. The two components interact through a communications protocol. The protocol determines the set of acceptable individuals that are able to update the belief space. Likewise the protocol determines how the updated beliefs are able to impact and influence the adaptation of the population component.

The cultural algorithm is a dual inheritance system that characterizes evolution in human culture at both the macro-evolutionary level, which takes place within the belief space and at the micro-evolutionary level, which occurs at the population space. CA consists of a social population and a belief space. Experience of individuals selected from the population space by the acceptance function is used to generate problem solving knowledge that resides in the belief space. The belief space stores and manipulates the knowledge acquired from the experience of individuals in the population space. This knowledge can control the evolution of the population component by means of the influence function. As a result, CA can provide an explicit mechanism for global knowledge and a useful framework within which to model self-adaptation in an EC system. The population level component of the cultural algorithm will be Evolutionary Programming (EP). The global knowledge that has been learned by the population will be expressed in terms of both normative and situational knowledge as discussed earlier.

A pseudo-code description of the Cultural Algorithms is described as follows:

Framework of cultural algorithm

```
1   Begin
2   t = 0;
3   Initialize population P(t);
4   Initialize belief space B(t);
5   Repeat
6   Evaluate P(t);
7   Update(B(t), accept(P(t)));
8   Generate (P(t), influence(B(t));
9   Select P(t) from P(t-1);
10  t+ = 1;
11  Until (termination condition)
12  End
```

In this algorithm, first the belief space and the population space are initialized. Then, the algorithm will repeat processing for each generation until a termination condition is achieved. Individuals are evaluated using the performance function. The two levels of Cultural Algorithm communicate through the acceptance function and the influence function. The acceptance function determines which individuals from the current population are selected to impact the belief space. The selected individuals' experiences are generalized and applied to adjust the current beliefs in the belief space via the update function. The new beliefs can then be used to guide and influence the evolutionary process for the next generation.

Cultural algorithms as described above consist of three components. First, there is a population component that contains the social population to be evolved and the mechanisms for its evaluation, reproduction and modification. Second there is a belief space that represents the bias that has been acquired by the population during its problem-solving process. The third component is the communications protocol that is used to determine the interaction between the population and their beliefs.

For the population component, any existing evolutionary computation model for a population can be viewed as an analogue model for the social component. For example, early mathematical models of cultural evolution viewed each type of traits as analogous to a gene and different values for a trait correspond to alleles for a given gene. Thus, an individual consisted of a vector of traits. These traits could be affected by the interaction with other individuals and the spread of traits could be modeled mathematically in terms of population genetics. The earliest model for the population component used was the genetic algorithm and it was used as the basis for modeling the evolution of agriculture.

The belief space in these early models corresponded to knowledge about the utility of a given social-learning operator. The probability of applying an operator was increased for an individual when use of the operator produced a new trait sequence with improved performance. Likewise, it was decreased for an individual if its use produced an individual with decreased performance. In these models, the acceptance function was 100%, since all individuals were used. The influence function affected the probability of applying an operator based upon the current belief about the operator utility. This allowed the social system to adjust its social learning practices to reflect the current evolutionary stage.

CIRCUIT AUTOMATIC DESIGN ALGORITHM

Circuit representation: In this study, the chromosome representation we use (Miller, 2003). This representation is based on the FPGA of Xilinx Virtex-II. The starting point in this technique is to consider, for each potential design, a geometry (of a fixed size array) of uncommitted logic cells that exist between a set of desired inputs and outputs. The uncommitted logic cells are typical of the resource provided on the Xilinx FPGA part under consideration. An uncommitted logic cell refers to a two-input, single-output logic module with no fixed functionality. The functionality may then be chosen, at the implementation time, to be any two input variable logic function. In this technique, a chromosome is defined as a

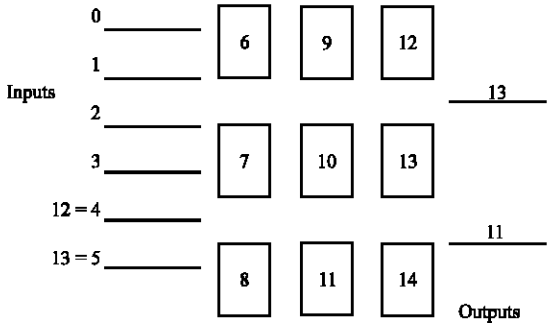

Fig. 1: A 3×3 geometry of uncommitted logic cells with inputs, outputs and netlist numbering

02-1 13-5 243 08-10 78-4 6119 64-9 2117 13 11

Fig. 2: A typical netlist chromosome for the 3×3 geometry of Fig. 1

set of interconnections and gate level functionality for these cells from outputs back toward the inputs based upon a numbered rectangular grid of the cells themselves, as in Fig. 1. The inputs that are made available are logic '0', logic '1', all primary inputs and primary inputs inverted. To illustrate this consider a 3×3 array of logic cells between two required primary inputs and two required outputs.

The inputs 0 and 1 are standard within the chromosome and represent the fixed values, logic '0' and logic '1', respectively. The inputs (two in this case) are numbered 2 and 3, with 2 being the most significant. The lines 4 and 5 represent the inverted inputs 2 and 3, respectively. The logic cells which form the array are numbered column-wise from 6 to 14. The outputs are numbered 13 and 11, meaning that the most significant output is connected to the output of cell 13 and the least significant output is connected to the output of cell 11. These integer values, whilst denoting the physical location of each input, cell or output within the structure, now also represent connections or routes between the various points. In other words, this numbering system may be used to define a netlist for any combinational circuit. Thus, a chromosome is merely a list of these integer values, where the position on the list tells us the cell or output which is being referred to, while the value tells us the connection (of cell or input) to which that point is connected and the cells functionality.

Each of the logic cells is capable of assuming the functionality of any two-input logic gate, or, alternatively a 2-1 Multiplexer (MUX) with single control input. In the geometry, a sample chromosome is shown in Fig. 2.

Gene value	Gate function
-1	A and B
-2	A and !B
-3	!A and B
-4	A^B
-5	A|B
-6	!A and !B
-7	!A^B
-8	!A
-9	A | !B
-10	!B
-11	!A | B
-12	!A | !B

Fig. 3: Cell gate functionality according to negative gene value in chromosome

Notice, in this arrangement that the chromosome is split up into groups of three integers. The first two values relate to the connections of the two inputs to the gate or MUX. The third value may either be positive-in which case it is taken to represent the control input of a MUX-or negative-in which case it is taken to represent a two-input gate, where the modulus of the number indicates the function according to Fig. 3. The first input to the cell is called A and the second input called B for convenience. For the logic operations, the C language symbols are used: (1) and for AND, (2) | for OR, (3) ^ for exclusive-OR and (4) ! for NOT. There are only 12 entries on this table out of a possible 16 as 4 of the combinations: (i) all zeroes, (ii) all ones, (iii) input A passed straight through and (iv) input B passed straight through are considered trivial-because these are already included among the possible input combinations and they do not affect subsequent network connections in cascade.

This means that the first cell with output number 6 and characterized by the triple {0, 2, -1} has it's a input connected to '0', its B input connected to input 2 and since the third value is -1, the cell is an AND gate (thus in this case always produces a logical output of 0). Picking out the cell who's output is labeled 9, which is characterized by the triple {2, 6, 7}, it can be seen that its A input is connected to input 2 and its B input is connected to the output of cell 6, while since the third number is positive the cell is a MUX with control input connected to the output of cell 7.

Belief space design: In this study, the algorithm's belief space uses the {S, N} structure. The formal syntax for the belief space B, used in this study is: B = S|N|⟨S, N⟩, where S denotes structure for situational knowledge and N denotes structures for normative knowledge. The definition above means the belief space can consist of situational knowledge only, normative knowledge only, or both. The situational knowledge S is represented formally as a pair wise structure: $S = \langle <E_1, E_2, ..., E_e>, adjust_E(e)\rangle$, where E_i represent an ith best exemplar individual in the evolution history. There can be e best exemplars in S as s set that constitutes the situational knowledge. Each exemplar individual has n parameters and a performance value. $Adjust_E$ (e) is the belief space operator applied to update e number of exemplar individuals in S. The normative knowledge, N, a set of interval information for each of the n parameters is defined formally as 4-tuple: $N = \langle I_j, L_j, U_j, adjust_N\rangle$, j = 1, 2,..., n, where I_j denotes the closed interval of variable j, that is a continuous set of real numbers x represented as a ordered number pair: $I_j = [l_j, u_j] = \{x|l_j \le x \le u_j, x \in R\}$. L_j (lower bound) and u_j (upper bound) are initialized by the give domain values. L_j represents the performance score of the lower bound l_j for parameter j. U_j represents the performance score of the upper bound u_j for parameter j.

Knowledge update function design: We defined the update function like this: S = {st}, select the best individual st update the situation knowledge S in belief space. The update process follows the Eq. 1:

$$s^{t+1} = \begin{cases} x^t_{best} & f(x^t_{best}) < f(s^t) \\ s^t & \text{others} \end{cases} \quad (1)$$

where, x^t_{best} bestdenotes tth best individual.

Update the normative knowledge N in belief space uses the Eq. 2:

$$\begin{aligned} l_i^{t+1} &= \begin{cases} x_{j,i} & x_{j,i} \le l_i^t \text{ or } f(x_j) < L_i^t \\ l_i^t & \text{others} \end{cases} \\ L_i^{t+1} &= \begin{cases} f(x_j) & x_{j,i} \le l_i^t \text{ or } f(x_j) < L_i^t \\ L_i^t & \text{others} \end{cases} \\ u_i^{t+1} &= \begin{cases} x_{j,i} & x_{j,i} \ge u_i^t \text{ or } f(x_j) < U_i^t \\ u_i^t & \text{others} \end{cases} \\ U_i^{t+1} &= \begin{cases} f(x_j) & x_{j,i} \ge u_i^t \text{ or } f(x_j) < U_i^t \\ U_i^t & \text{others} \end{cases} \end{aligned} \quad (2)$$

Influence function design: In our algorithm, the knowledge represented in the belief space can be

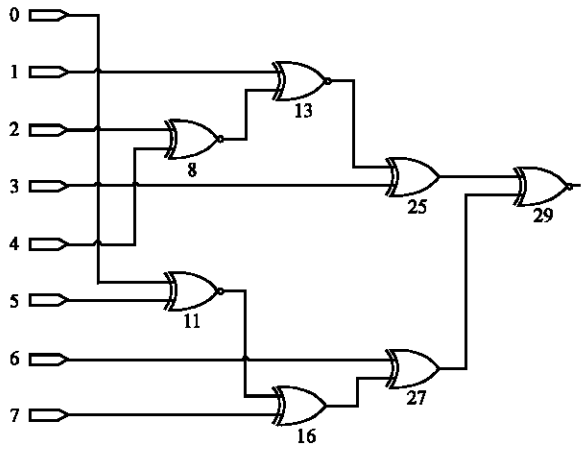

Fig. 4: Eight-bit even checker's circuit

explicitly used to influence the creation of the offspring via an influence function. In our sliding window model, the strategy can be simply described as follows. The first is if a parent is in a promising region, the offspring are created by randomly changing the problem parameters of the parent just a little. In this case, the normative knowledge applies. The offspring $x_{j,i}^{t+1}$, will be created by using this normative knowledge as a beacon to attract the parent $x_{j,i}^{t}$ to move a copy toward the current sliding window, the influence function defined by the Eq. 3:

$$x_{j,i}^{t+1} = \begin{cases} x_{j,i}^{t} + |size(I_i) * N(01)| & x_{j,i}^{t} < l_i^t \\ x_{j,i}^{t} - |size(I_i) * N(01)| & x_{j,i}^{t} > u_i^t \\ x_{j,i}^{t} + \lambda_1 * size(I_i) * N(0,1) & others \end{cases} \quad (3)$$

The second is if a parent is in an unpromising region, moving a copy of the parent to a more promising region can be used to create a new offspring. In this case, the constraint knowledge applies. The creation of offspring will be affected by the characteristic of the cells within the sliding window, the influence function defined by the Eq. 4.

$$x_{j,i}^{t+1} = \begin{cases} x_{j,i}^{t} + |size(I_i) * N(01)| & x_{j,i}^{t} < s_i^t \\ x_{j,i}^{t} - |size(I_i) * N(01)| & x_{j,i}^{t} > s_i^t \\ x_{j,i}^{t} + \lambda_1 * size(I_i) * N(0,1) & others \end{cases} \quad (4)$$

EXPERIMENT RESULTS

Parity checker circuit: We use this algorithm design Parity Checker's circuit. The following Fig. 4 and 5 are the results (for the eight-bit parity Checker).

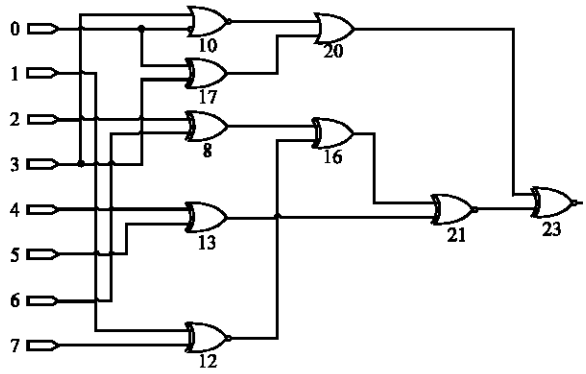

Fig. 5: Eight-bit odd checker's circuit

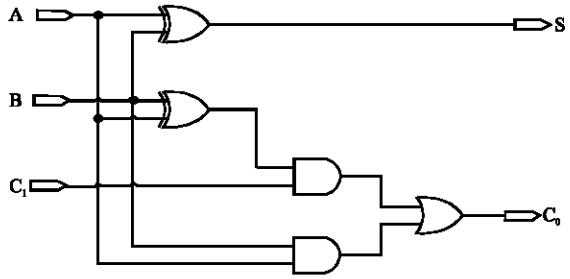

Fig. 6: One-bit full adder circuit designed by manual

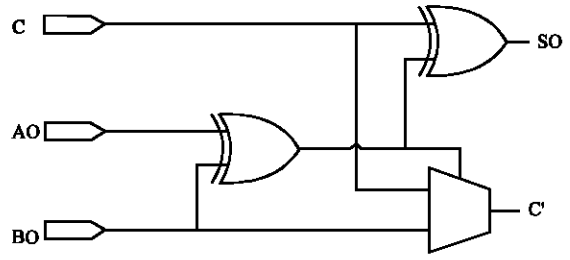

Fig. 7: One-bit full adder circuit designed by our algorithm

Full adder circuit: Evolving the one-bit adder was easier to do on a larger geometry but resulted in a less efficient circuit. That is many genetic algorithm was able to discover 100% functional solutions was intimately related to the size of the geometry but our algorithm use small geometry to find the fully functional solutions. The circuit design by GA is shown in Fig. 6 (with five gates), Fig. 7 is our algorithm's results (with three gates). From the figures we know it is a gratifying result to obtain as it is clear that this design is an optimum solution.

A two-bit full adder circuit, which with a truth table with 5 inputs and 3 outputs. In this case, Our algorithm use small geometry to find the fully functional solutions, the matrix has a size of 3×3. The original circuit

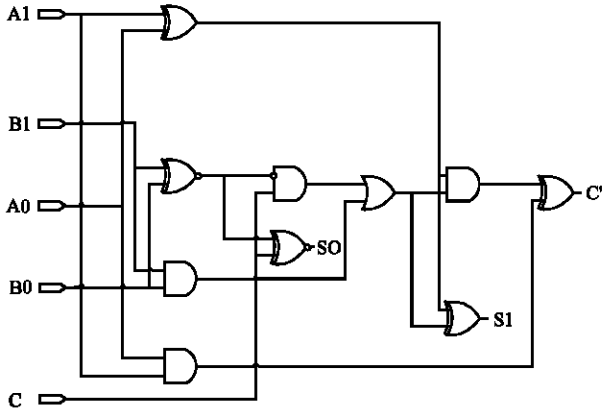

Fig. 8: Two-bit full adder circuit designed by manual

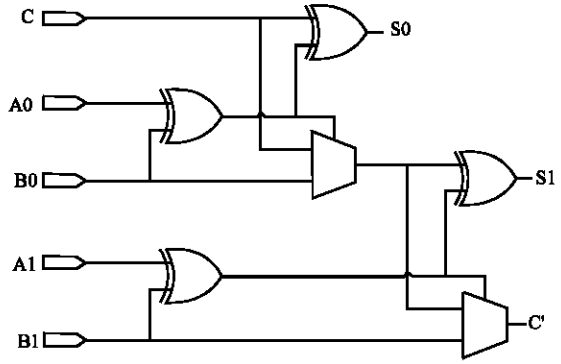

Fig. 9: Two-bit full adder circuit designed by our algorithm

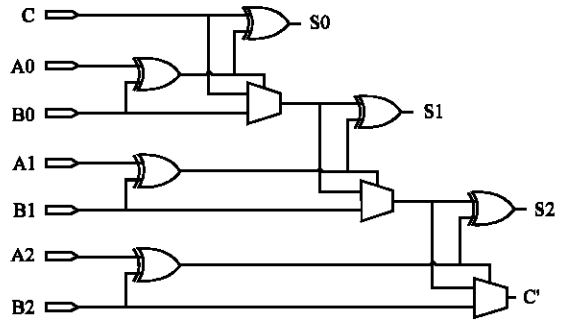

Fig. 10: Three-bit full adder circuit designed by our algorithm

is shown in Fig. 8 (with ten gates) and our resulting circuits as shown in Fig. 9 (with six gates). From the figures we know it is a gratifying result to obtain as it is clear that this design is an optimum solution.

We also use our algorithm to evolve the three-bit full adder and four-bit full adder. A three-bit full adder, with a truth table with 7 inputs and 4 outputs. Our algorithm use small geometry to find the fully functional solutions, the

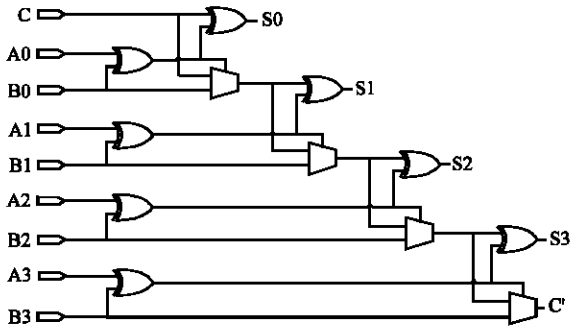

Fig. 11: Four-bit full adder circuit designed by our algorithm

Table 1: Experiment results statistics

Case	Evolution times	Success rate (%)	Optimal gates
One-bit full adder	3 (sec)	100	3
Two-bit full adder	50 (sec)	100	6
Three-bit full adder	20 (min)	90	9
Four-bit full adder	7 (h)	50	12

matrix has a size of 4×4. The resulting circuits as shown in Fig. 10 (with 9 gates). A four-bit full adder, with a truth table with 9 inputs and 5 outputs. Our algorithm use small geometry to find the fully functional solutions, the matrix has a size of 8×8. The resulting circuits as shown in Fig. 11 (with 12 gates). Table 1 is the experiment results statistics for one-bit to four-bit full adder circuit, each case is run 50 times.

CONCLUSION

This study proposed a new means for designing electronic circuits given a set of logic gates. The final circuit is optimized in terms of complexity (with the minimum number of gates). For the case studies this means has proved to be efficient, experiments show that we have better results.

There are still many avenues for further work. Other ways of representing rectangular arrays of logic cells may be devised and also, the relationship between cell connectivity and the evolvable of designs has still to be explored. There are many wider issues to be considered also which relate to the problem of evolving much larger circuits. It is a feature of the current technique that one has to specify the functionality of the target circuit using a complete truth table, however this is impractical for circuits with large numbers of inputs.

ACKNOWLEDGMENTS

This study is supported by the Provincial Natural Science Foundation of Hubei (No. 2011CDB334 and

2011CDB346), the Fundamental Research Founds for National University, China University of Geosciences (Wuhan) and National Civil Aerospace Pre-research Project of China.

REFERENCES

Bellman, R.E., 1957. Dynamic Programming. Princeton University Press, Princeton, New Jersey.

Chung, C., 1997. Knowledge-based approaches to self-adaptation in cultural algorithms. Ph.D. Thesis, Wayne State University, Detroit, Michigan, USA.

Coello, C.A., A.D. Christiansen and A.H. Aguirre, 1996. Using genetic algorithms to design combinational logic circuits. Intell. Eng. Artif. Neural Networks, 6: 391-396.

Ferreira, C., 2001. Gene expression programming: A new adaptive algorithm for solving problems. Complex Syst., 13: 87-129.

Goldberg, D.E., 1989. Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley, New York, USA.

Kalganova, T., J.F. Miller and N. Lipnitskaya, 1998. Multiple valued combinational circuits synthesized using evolvable hardware. Proceedings of the 7th Workshop on Post-Binary Ultra Large Scale Integration Systems, May 27-29, 1998, Fukuoka, Japan.

Koza, J.R., 1992. Genetic Programming: On Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA.

Louis, S.J. and G.J. Rawlins, 1991. Designer genetic algorithms: Genetic algorithms in structure design. Proceedings of the 4th International Conference on Genetic Algorithms, July 13-16, 1991, University of California, San Diego, pp: 53-60.

Michalewicz, Z., 1996. Genetic Algorithms+Data Structure = Evolutionary Programs. 3rd Edn., Springer, New York.

Miller, J.F., 2003. Designing electronic circuits using evolutionary algorithms. Arithmetic circuits: A case study. Department of Computer Studies, Napier University, New Zealand.

Miller, J.F., P. Thomson and T.C. Fogarty, 1997. Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. In: Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications, Quagliarella, D., J. Periaux, C. Poloni and G. Winter (Eds.). Wiley, New York, USA.

Reynoids, R., 1989. An introduction to cultural algorithms. Proceedings of the 3rd Annual Conference on Evolutionary Programming, February 24-26, 1944, San Diego, CA, USA., pp: 131-139.

Salem-Zebulum, R., M.A.C. Pacheco and M.V.R. Vellasco, 2002. Evolutionary Electronics, Automatic Design of Electronic Circuits and Systems by Genetic Algorithms. CRC Press, USA.

Thompson, A. and P. Layzell, 1999. Analysis of unconventional evolved electronics. Commun. ACM, 42: 71-79.

Torresen, J., 1998. A divide-and-conquer approach to evolvable hardware. Proceedings of the 2nd International Conference on Evolvable Systems: From Biology to Hardware, September 23-25, 1998, Switzerland, pp: 57-65.

Wolpert, D.H. and W.G. Macready, 1997. No free lunch theorems for optimization. IEEE Trans. Evol. Comput., 1: 67-82.

Yan, X., K. Wang, J. Jin, Q. Liang and C. Hu, 2010. Designing electronic circuits using cultural algorithms. Proceedings of the 3rd International Workshop on Advanced Computational Intelligence, August 25-27, 2010, Suzhou, Jiangsu, pp: 299-303.

Yan, X., Q. Wu, C. Hu and Q. Liang, 2011a. Electronic circuits optimization design based on cultural algorithms. Int. J. Inf. Process. Manage., 2: 49-56.

Yan, X., Q.H. Wu, C.Y. Hu and Q.Z. Liang, 2011b. Circuit design based on particle swarm optimization algorithms. Key Eng. Mater., 474: 1093-1098.

Yan, X., W. Wei, Q. Liang, C. Hu and Y. Yao, 2007. Designing electronic circuits by means of gene expression programming II. Proceedings of the 7th International Conference on Evolvable Systems: From Biology to Hardware, September 21-23, 2007, Wuhan, China, pp: 319-330.

Yan, X., W. Wei, R. Liu, S.Y. Zeng and L.S. Kang, 2006. Design electronic circuits by means of gene expression programming. Proceedings of the 1st NASA/ESA Conference on Adaptive Hardware and Systems, Istanbul, Turkey, June 15-18, 2006, pp: 194-199.