A Modified Ant Colony Algorithm for Evolutionary Design of Digital Circuits

Mostafa Abd-El-Barr, Sadiq M. Sait, Bambang A. B. Sarif, Uthman Al-Saiari
Computer Engineering Department
KFUPM Box 673, Dhahran-31261
{mostafa, sadiq, sarif, saiarios}@ccse.kfupm.edu.sa

Abstract- Evolutionary computation presents a new paradigm shift in hardware design and synthesis. According to this paradigm, hardware design is pursued by deriving inspiration from biological organisms. The new paradigm is expected to radically change the synthesis procedures in a way that can help discovering novel designs and/or more efficient circuits. In this paper, a multiobjective optimization of logic circuits based on a modified Ant Colony (ACO) algorithm is presented. The performance of the proposed algorithm is evaluated using a set of randomly generated circuits. The results obtained using the proposed algorithm are compared to those obtained using existing ACO-based techniques. It is shown that the designed circuits using the proposed algorithm outperform those of the existing techniques.

1 Introduction

Conventional logic design techniques tend to depend on domain-specific knowledge, which is somewhat limited both by the training and experience of the designer. While iterative heuristics, with little domain knowledge, allow the search for solutions in a much larger, and often richer, design space beyond the realms of conventional design techniques.

The Ant Colony Optimization (ACO) algorithm is a new meta-heuristic that combines distributed computation, auto-catalysis (positive feedback) and constructive greedy heuristic in finding optimal solutions for combinatorial optimization problems [1]. Unlike Genetic Algorithms (GAs), which are blind, ACO involves cooperating agents (ants). In ACO, interaction between components of a designed system can be easily analyzed. Some daemon actions or other heuristics can also be easily incorporated to further improve the quality of solutions. In this context, it is possible to use some rules from the logic synthesis domain to guide the search process to obtain not only better quality results, but also faster ones.

In the early 1990's, Hugo de Garris suggested the establishment of a new field of research called Evolvable Hardware (EHW) [2]. The first work in evolutionary design of digital circuits, Designer Genetic Algorithms (DGA), was proposed by Louis [3]. Later, the work of Thompson [4] that produced a tone discriminator circuit without input clock showed the emergence of a new way of designing circuits.

In a recent development [5, 6], much attention is given to the evolutionary design of arithmetic circuits as they provide the essential building blocks needed for larger DSP applications. Such effort has resulted in the development of arithmetic circuits that range from a simple sequential adder to the more complex 3-bit multiplier. The work of Fogarty [7] and Miller [5, 8] claimed to build some arithmetic circuits that cannot be produced by human designer's conventional methods. Coello et al. [9] proposed a similar approach to evolve a circuit, which they claimed was better than that of Miller's. Several other algorithms such as Cartesian Genetic Programming, Ant Colony Optimization, and Particle Swarm Optimization have also been used for evolutionary logic design [6, 10, 11]. A complete review and taxonomy of the field could be found in [12, 13, 14].

Although most of the existing techniques in evolutionary design were able to arrive at solutions that are difficult to obtain using conventional methods, there exist many open problems that were still not addressed. A number of these problems are described below.

Circuit representations: Most of the published work in evolutionary logic design used a two-dimensional matrix of $n \times m$ to represent a circuit. The position of circuit's outputs will most likely be placed at cell(0,$m-1$). However, it may happen that the best solution can be found at cell(i, j), $0 < i < n$, $0 < j < m$. But some redundant gates existing between this cell and the output cell may degrade the quality of the solution. The problem becomes complicated even further when the number of circuit's output is more than one. Figure 1 illustrates the problem arises from circuit representation.

Functional fitness calculation: The value of functional fitness depends on the number of correct matchings between the output's pattern of the obtained solution and the truth table of the intended circuit. The higher the number of hits achieved, the higher the value of the functional fitness. This argument is not always true in logic design. A solution that has low functional fitness can be inverted to have a high functional fitness (see Figure 2).

Objectives of the optimization: Most of the existing techniques use gate count as their objective for optimization. With the increasing need for high performance and low

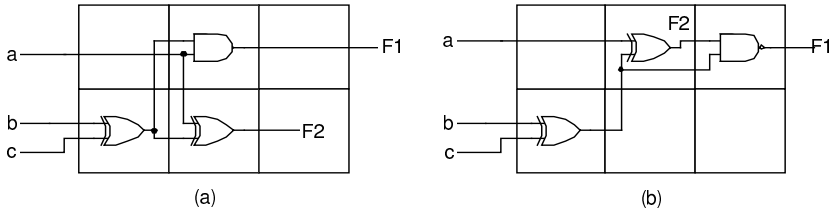

Figure 1: Problems that may appear in matrix representation.

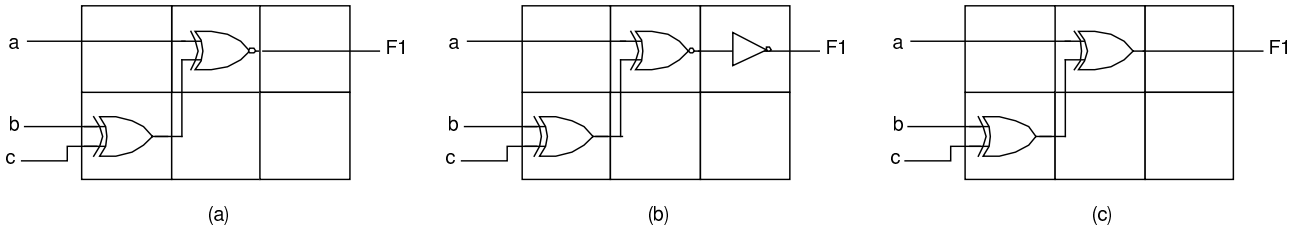

Figure 2: An evolved two-bit odd parity circuit. (a) Fitness of F1 = 0 (b) Adding an inverter, fitness of F1 = 1 (c) Toggle the type of gate (XNOR $\rightarrow$ XOR), fitness of F1 = 1.

power circuits, the objective of only minimizing gate count is not anymore acceptable.

This paper is organized as follows: first, problem and cost function formulation are presented. Then, the modified ACO for logic design is discussed. Finally, performance evaluation and comparison with existing techniques are given.

2 Problem and Cost Function Formulation

Evolutionary computation views the problem of logic design as a search task. The methodology explores a solution space larger than that of the desired function, but gradually pulls the specification of the circuit towards the target truth table. However, the design space of digital circuits is huge. There are 2^n ($C_1^{2^n}$) possible solutions that satisfy $2^n - 1$ out of 2^n truth table's pattern for an n inputs single output function. In addition to that, the number of possible structures representing each of these solutions is many. These different structures represent different design objectives and/or constraints. Exploring the whole search space is impractical. Therefore, the search space sampled by the algorithm must have its size limited.

In this paper, we use the structure proposed in [3]. Each cell of the $n \times m$ matrix contains the information of the gate type and its corresponding inputs. However, unlike the fixed interconnection rules used in [3], we allow the output of each cell in column j to be connected to any of the cells in column $j+1$ ($j > 0,\ j+1 < m$). Thus, it is possible that cell$(i,k),\ 0 < i < n,\ 0 < k < m$, is not connected to any of the cells in column $k+1$.

It is known that each type of gate has different characteristics for different technology. These characteristics include area, base delay and capacitance input (output) of the gate. Although we can build any logic circuit using AND, OR and NOT gates, we need to have a rich (but limited) gate library to be able to obtain different structures of the circuits. The best circuit can then be chosen based on the multiobjective criteria applied in the algorithm. Therefore, ten types of gates are considered. Table 1 shows these gates.

Gate ID	Inputs	Gate	Output
0	a, b	WIRE1	a
1	a, b	WIRE2	b
2	a, b	NOT1	$\overline{a}$
3	a, b	NOT2	$\overline{b}$
4	a, b	AND	$a \cdot b$
5	a, b	OR	$a + b$
6	a, b	XOR	$a \oplus b$
7	a, b	NAND	$\overline{a \cdot b}$
8	a, b	NOR	$\overline{a + b}$
9	a, b	XNOR	$\overline{a \oplus b}$

Table 1: Gate types, Gate ID, and its corresponding Boolean function.

2.1 Fitness Calculation

The fitness of a solution consists of two parts: functional fitness and objective fitness. These are explained below.

Functional Fitness: The functional fitness deals with the functionality of the solution, i.e., how good the solution is in satisfying the truth table of the intended Boolean function. Several functional fitness formulations are reported in the literature [13]. The commonly used one is the ratio of the number of correct hits to the length of the truth table. If FF denotes the functional fitness, then the formulation below is applied.

$$FF = \frac{Number\ of\ hits}{Length\ of\ the\ truth\ table} \tag{1}$$

The solution has to be 'inverted' if the value of FF is less than 0.5. Therefore, the formulation of *normalized* FF (FF_n) below is applied.

$$FF_n = Max\{FF, 1 - FF\} \tag{2}$$

Objective Fitness: The objective fitness (OF) is the measure of the quality of solution in terms of optimization objectives such as area, delay, gate count and power consumption. Formulation of cost functions used to estimate these values is given as follows.

If G is the set of possible gate types and $g_i \in G$, the cost for gate count is formalized as follows:

$$Cost_{gate\_count} = \sum_{i \in G, i \neq WIRES} g_i \tag{3}$$

The cost for area of VLSI circuits is stated as follows.

$$Cost_{area} = \sum_{i \in G, i \neq WIRES} A(g_i) \tag{4}$$

Where $A(g_i)$ is the area of gate $g(i)$.

The propagation delay of signals in VLSI circuit consists of two elements, switching delay of gates and interconnect delay. If a path π consists of n gates $\{v_1, v_2, ..., v_n\}$, then, the delay T_π along π is expressed by the following equation:

$$T_\pi = \sum_{i=1}^{n-1} (CD_i + ((LF_i + R_i) \times C_i))$$

Where CD_i is the switching delay of the cell driving gate v_i. LF_i is the load factor of the driving block, R_i is the interconnect resistance of net v_i, and C_i is the load capacitance of cell i given by Equation 5. Since the value of R_i is constant, it can be neglected. The overall circuit delay is determined by the delay along the longest path (the most critical path).

The total capacitance C_i of gate i consists of the interconnect capacitance at the output node of gate i and the sum of the capacitances of the input nodes of the gates driven by gate i.

$$C_i = C_i^r + \sum_{j \in M_i} C_j^g \tag{5}$$

Where C_j^g is the capacitance of the input node of a gate j driven by gate i and C_i^r represents the interconnect capacitance at the output node of cell i.

The total power consumption can be approximated by the following equation [15].

$$P_t \simeq \sum_{i \in M} \frac{1}{2} \cdot C_i \cdot V_{DD}^2 \cdot f \cdot S_i \cdot \beta \tag{6}$$

Where P_t is the total power consumption, V_{DD} is the supply voltage, S_i is the switching probability at the output node of cell i, i.e., the average number of transitions per clock cycle at the output of gate i, f is the clock frequency and β is a technology dependent constant.

The cost of the overall power consumption in VLSI circuits can then be estimated as follows.

$$Cost_{power} = \sum_{i \in M} S_i \cdot C_i \tag{7}$$

In order to indicate whether a given solution is satisfying a certain constraint, objective fitness is formulated as follows. Note that the constraint values states the upper bound for a specific objective.

$$Obj = \frac{Cost}{Cost + Constraint} \tag{8}$$

With this formulation, the solution that satisfies the area constraint will have Obj_{area} greater than 0.5. Any solution that has Obj_{area} less than 0.5 will not be considered. The objective fitness is then calculated as follows.

$$OF = \frac{\sum_{k \in obj} w_k \cdot Obj_k}{\sum_{k \in obj} w_k} \tag{9}$$

The weights for each elements of OF decide the emphasis of the optimization process. It is also possible to consider more than one objective.

Overall Fitness Calculation: The formulation of overall fitness function is shown in Equation 10. W_f is the weight for functional fitness. The value of W_f determines the trade-off in the searching process, whether to have solutions better in terms of FF (or OF). Since the goal is to obtain working circuits, the value of W_f must be large enough. However, it should not be too large in order to accommodate the OF part. Initial experiments showed that the setting of W_f = 0.5 is appropriate for small circuits (up to four variables). However, these values cannot be used for bigger circuit.

$$OvF = W_f \cdot FF + (1 - W_f) \cdot OF \tag{10}$$

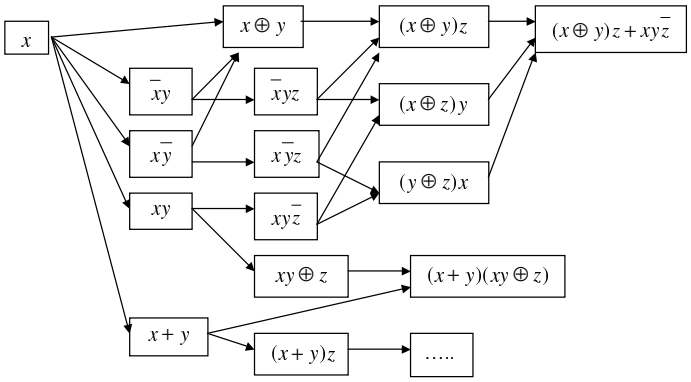

Figure 3: Some of the possible paths in the function f.

3 Proposed Approach

Consider the Boolean function $f = \overline{x}yz + x\overline{y}z + xy\overline{z}$. Figure 3 shows a graph of some possible paths connecting literal x to the intended function f. Assume that the ants start the tour from literal x. The ant will traverse the paths by selecting the branches through a probabilistic process. Assume that the goal is to find the shortest path to represent function f. Therefore, the ants that found the path $x \rightarrow (x + y) \rightarrow (x + y)(xy \oplus z)$ would return the best representation for function f.

The number of paths in Figure 3 is more than eleven. Traversing all possible paths is however, impractical. We need to modify the ACO algorithm to handle this huge search space.

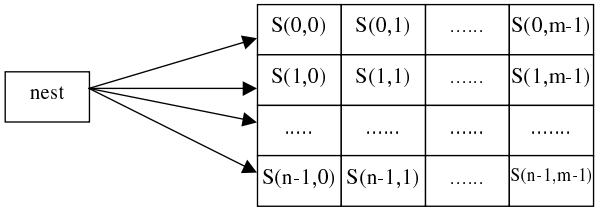

Figure 4: Nest cell and matrix M for ant to be traversed.

A circuit is modelled as a matrix M of size $n \times m$. The content of matrix M is dynamically filled. At first, matrix M is filled with randomly generated cells. Then, each ant will traverse the matrix. These ants originate from a dummy cell called *nest* (see Figure 4), and traverse each state (a cell in a column) until it reaches the last column or a cell that has no successor.

After the ants finish traversing the matrix, all cells are checked to see whether to be kept or not. Each cell can assume two different status, namely: 'l' (locked) or 'r' (removed). The cells that are included in the best path(s) will assume the status of 'l' (locked). And the cells that are feeding locked cells will be locked as well. All other cells will assume the status 'r' (removed). The cells that assume status 'r' will be removed at the end of each iteration. These empty cells will be then filled up again at the beginning of the next iteration. Figure 5 shows the pseudocode of the proposed algorithm.

Modified ACO algorithm
  For MAXITER number of iteration do
    Fill the matrix
    ACO algorithm
      Ant activity
      Pheromone update
    Remove unfit cells
  End For
  Return the best path
end Algorithm

Figure 5: Modified Ant Colony Algorithm.

3.1 Pheromone Trail Calculation and Update

The selection of which edge to traverse is determined by a stochastic process, e.g. *Roulette Wheel*. Therefore, the probability of choosing each edge must be calculated in advance. This probability depends on the pheromone value (τ) and the heuristic value (η) of the corresponding edge (or the next cell), or can be formulated below.

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in \aleph_i^k} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}} \qquad (11)$$

The value of α and β imply the preference of the search, whether it depends more on the pheromone value or the heuristic value. Every newly created cell will be given an initial and small amount of pheromone value. This value will be updated every iteration by the ant.

The heuristic value (η) depends on the distance of FF_n values between cells. The distance d between cells is formulated as follows.

$$d = FF_n(i+1) - FF_n(i) \tag{12}$$

$$\eta = d + 0.5 \tag{13}$$

The addition of 0.5 in the calculation of η is meant to normalize the value of η into [0,1]. A decrease of the functional fitness means that the value of η is in the range of [0,0.5), while an increase of the functional fitness makes the value of η in the range of (0.5, 1]

While traversing the matrix, every ant carries the information of the paths taken so far, e.g., the row index of all cells that are visited. If an ant reaches a cell that has no successor, the overall fitness of the solution built by the ant will be evaluated.

When all ants finish their tour, pheromone update is performed. The pheromone update consists of two procedures, pheromone addition and pheromone evaporation. However, as has been shown in [16], it is better to limit the number of ants that can put additional pheromone. Thus, only certain number of 'the best' ants can track their path(s) back and put some additional pheromone on it. The pheromone addition is performed using the following equation:

$$\Delta\tau = \lambda \cdot OvF(t) \tag{14}$$

$$\tau(t) = \tau(t) + \Delta\tau \tag{15}$$

where $OvF(t)$ denotes the overall fitness of the solution that the ants built, $\Delta\tau$ is the additional pheromone and λ is a constant.

Next, pheromone evaporation will take place using the following formula.

$$\tau = (1-\rho) \times \tau, \;\; with \;\; \rho = (0,1] \tag{16}$$

While traversing the matrix, the ants will memorize the best cell visited along the path. The paths from nest to the best cell will be returned. The remaining part of the path will be discarded by the ant.

When the maximum number of iterations is reached, the best solution is returned. In case of multiple output circuits, multiple colony of ants are used. In this context, each colony of ants is assigned to find a specific output of the circuit. All colonies will share the same matrix. The possibility of using the same sub-functions is established by sharing the pheromone value among different colonies.

4 Experimental Setup

The technology parameters are obtained using CMOS 0.25 micron library from MOSIS [17]. The parameters used for experiments are as follows:

1. α = 1
2. β = 2
3. ρ = 0.1
4. Wf = [0.5, 0.75]
5. Number of ants = 10 * number of inputs
6. Number of generation = 10
7. Maximum iteration = 2000 (four input circuits) or 3000 (five input circuits)
8. Number of runs = 10-15
9. Size of the matrix = (2 * number of inputs) $\times$ (2 * number of inputs)

It should be noted that we avoid using gate count as the measure of objective for several reasons. Firstly, the term 'gate' or basic module for the evolutionary logic design depends on the definition of the gate library that is used. One may use NAND gates, or a set of AND, OR and XOR gates, or MUXes, or a combination of all these. Secondly, each of the aforementioned gates has different characteristics. We can assume that an XOR gate as an atomic gate. However, this may not be the case for all target implementations. For example, in standard cell design, an XOR gate requires more area compared to an AND gate, and an AND gate requires more area than an NAND gate. This is in contrast with FPGA, in which all types of gate can fit into one cell. Nevertheless, if the target implementation is an FPGA, calculating the area is proportional to calculating the gate count. In this context, the proposed algorithm is more general as compared to the existing techniques. However, since most of the published work in evolutionary logic design use the gate count as a measure of quality, we provide a comparison of our results in terms of gate count as well.

5 Performance Evaluation and Comparison

Several circuits of different complexity have been used to test the proposed algorithm. For the sake of simplicity, the truth table of the circuits will be represented as a string of zeros and ones. Table 2 shows the circuits used for performance evaluation. Note that these circuits represent single output Boolean functions.

Name	Inputs	Truth Table
Circuit1	4	0101110100100111
Circuit2	4	0010000011111110
Circuit3	4	0111101001110001
Circuit4	5	10001111001011110011010011110101
Circuit5	5	01000101010010100111010000001100
Circuit6	3	00010110
Circuit7	4	1101001110100100
Circuit8	4	1000111111100101

Table 2: Circuits used to test the performance of the proposed approach.

Figure 6 shows the behavior of the proposed algorithm for area optimization of Circuit1 for the first 100 iterations.

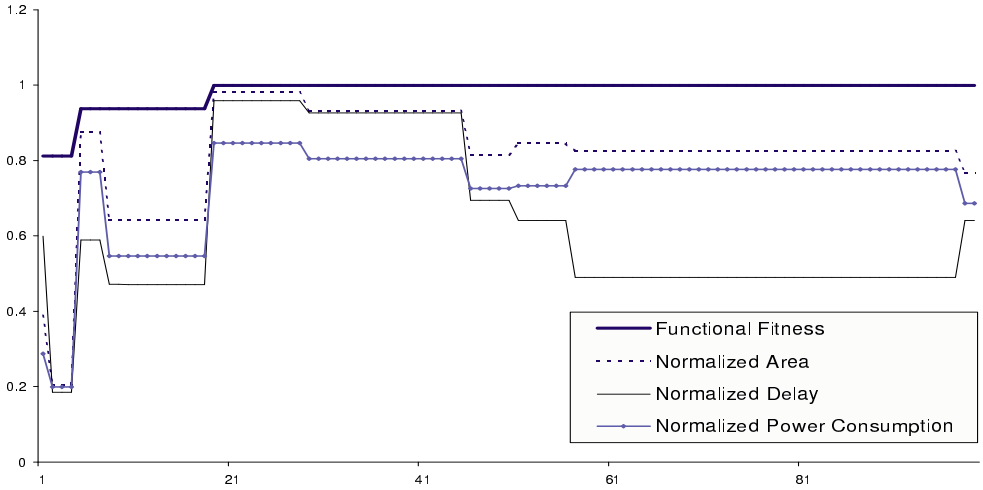

Figure 6: Functional fitness and satisfaction of objectives for the first 100 iterations.

In this figure, the area, delay and power consumption are normalized so that the behavior of the proposed approach can be observed easily. As can be seen, an increase in functional fitness mostly requires an increase in some of the objectives. But then, if there is no increase in the functional fitness, minimization of objectives is observed. The proposed algorithm is able to arrive at a functionally correct circuit for the intended truth table after only 20 iterations. Around iteration 100, a significant increase in the delay of the solution is observed. However, since the emphasize of the current course of action is optimization of area, then new solution that has less area but higher delay is accepted.

Circuit	Objectives	Delay Optimized		Power Optimized	
		Best	Average	Best	Average
Circuit1	area	1.14	1.13	1.00	1.09
	delay	0.81	0.99	1.00	1.04
	power	1.07	1.10	1.00	1.06
Circuit2	area	1.27	1.20	0.92	1.00
	delay	0.99	1.02	1.00	1.14
	power	1.29	1.17	0.92	0.97
Circuit3	area	0.96	1.04	0.87	0.90
	delay	1.00	0.92	0.87	0.94
	power	0.94	1.00	0.87	0.91
Circuit4	area	1.44	1.04	1.18	1.09
	delay	0.83	0.97	1.16	0.98
	power	1.41	1.04	1.05	1.07
Circuit5	area	0.94	1.00	0.75	0.97
	delay	1.05	0.94	1.54	1.09
	power	0.94	0.97	0.78	0.96

Table 3: Results for the delay and power optimization, normalized with respect to the results of area optimization.

The results of delay optimized and power optimized circuits are given in Table 3. Note that these values are normalized with respect to the results obtained from the area optimized circuits. The normalized area for each circuit in Table 3 should be less than or equal to one. As can be seen, except for Circuit5, the delay optimization scheme produced better circuits in terms of delay (normalized delay ≤ 1), at the expense of larger area and/or power consumption. These results show the effectiveness of the proposed algorithm.

In order to compare our algorithm with known published work, some circuits are tested and compared to the results reported in [9, 10]. Some selected circuits from Table 2 with the addition of 2-bit multiplier and 2-bit adder with carry circuits are used for comparison. An overall comparison is shown in Table 4.

For multiple output circuits, multiple colony of ants are used. In a 2-bit adder circuit for example, we need three colony of ants to find all the three outputs of the circuits. Each colony will find the assigned circuit's output. The sharing of pheromone value between colonies makes the sharing of sub-functions possible. However, this sharing can indeed bias the search, since the second colony of ants 'smell' the pheromone that was updated by the first colony of ants, and the third colony will 'smell' the pheromone that will be updated by the second colony, and so on. This is the reason why the value of β of Equation 11 is set equal to 2 during the course of the experiment. With higher value of β, the search is more dependent on the heuristic value η, which emphasizes finding the functionally correct circuits. Figure 7 shows the functional fitness value of each outputs of a 2-bit adder circuit. Notice that the ants will find outputs of the circuit one by one. In analogy with the process of ants finding the food, the simplest function (the closest) will be obtained first and the most complext function (the furthest) will most likely be obtained the last. In this context, for 2-bit adder circuit, the first sum will be obtained first while the carry out will be obtained the last.

Circuit	Algorithm	Resulting Function	Gate Count	Area (micron)
Circuit6	MGA [9]	$F = Z(X+Y) \oplus XY$	4	9477
	Ant System [10]	$F = (Z \oplus XY)(X+Y)$	4	9477
	Proposed Approach	$F = \overline{(X+Y)Z} \odot \overline{XY}$	4	7776
Circuit7	MGA [9]	$F = ((W \oplus WX) \oplus ((Z+X+Y) \oplus Z))'$	7*	17496
	Ant System [10]	$F = (((W+Y) \oplus Z) + X')((YZ)' \oplus (X'W))$	9	19197
	Proposed Approach	$F = ((\overline{X+Z}) + (Y \oplus Z)) \odot (\overline{W\overline{XY}})$	6	12393
Circuit8	MGA [9]	$F = ((A \oplus B) \oplus AD) + (C + (A \oplus D))'$	6*	15552
	Ant System [10]	$F = ((B \oplus D) \oplus (A+D)) \oplus ((B+C) + (A \oplus D))'$	7*	18468
	Proposed Approach	$F = \overline{((\overline{A \odot D}) \cdot C') \cdot (\overline{AD} \odot (A \odot B))}$	7	14337
mul2	MGA [9]	$F_0 = AB;\ \ F_1 = AD \oplus BC$ $F_2 = CD \oplus ABCD;\ \ F_3 = ABCD$	7	17253
	Ant System [10]	$F_0 = AB;\ \ F_1 = AD \oplus BC$ $F_2 = CD \oplus ABCD;\ \ F_3 = ABCD$	7	17253
	Proposed Approach	$F_0 = AB;\ \ F_1 = \overline{AD} \oplus \overline{BC}$ $F_2 = \overline{\overline{CD} \cdot AB};\ \ F_3 = \overline{\overline{BC} \cdot \overline{AD}}$	7	12636
add2	Proposed Approach	$F_0 = A \odot B \odot C$ $F_1 = \overline{(A \odot B) + C} \odot (D \odot E \odot (\overline{A+B}))$ $F_2 = \overline{\overline{DE} \cdot ((\overline{A+B}) + (D \odot E))} + (\overline{(A \odot B) + C})$	11	24300

* inverter at the output(s) is not considered

Table 4: Comparison with existing techniques in terms of gate count and area of the circuits.

6 Conclusion

In this paper, we have presented a modified ant colony algorithm for evolutionary logic design. The modification is performed in order to suit the problem instance and to handle some of the problems that are not addressed by existing techniques. Performance of the proposed approach and comparison with the existing techniques are shown. In all cases, the results obtained using the proposed approach outperformed those obtained using existing techniques.

Acknowledgment

We would like to acknowledge the continued support for our research from King Fahd University of Petroleum & Minerals. This work was supported under project entitled "Iterative Heuristics for the Design of Combinational Logic Circuits".

Bibliography

[1] M. Dorigo, M. Maniezzo, and A. Colorni. The Ant Systems: An Autocatalytic Optimizing Process. Revised 91-016, Dept. of Electronica, Milan Polytechnic, 1991.

[2] Hugo de Garis. Evolvable Hardware: Genetic Programming of a Darwin Machine. *Proceedings of the International Conference in Innsbruck, Austria*, pages 441–449, Springer-Verlag, 1993.

[3] Sushil J. Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University, Aug 1993.

[4] Adrian Thompson. Silicon Evolution. *Proceedings of the First Annual Conference on Genetic Programming*, pages 444–452, MIT Press, 1996.

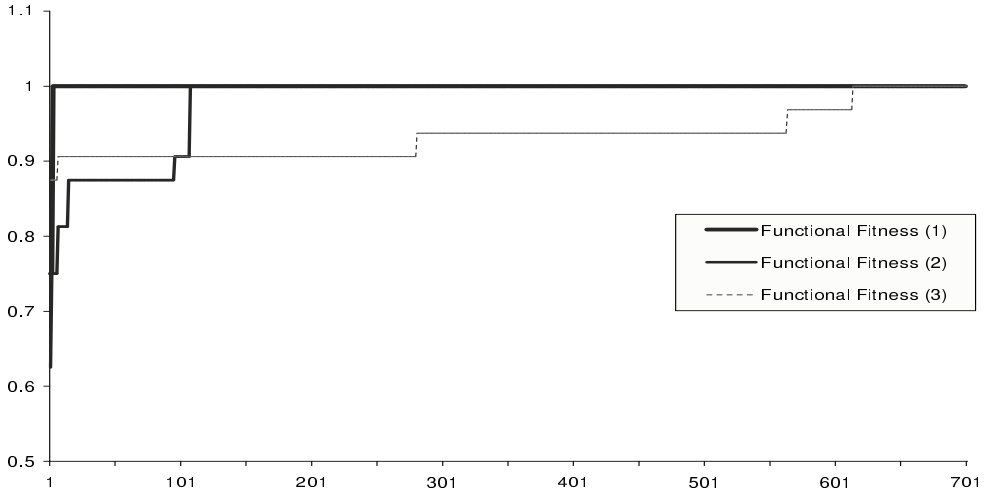

Figure 7: Functional fitness of 2-bit adder.

[5] J. F. Miller, D. Job, and Vassilev V. K. Principles in the Evolutionary Design of Digital Circuits - Part I. *Journal of Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.

[6] J. F. Miller and P. Thomson. A Developmental Method for Growing Graphs and Circuits. *Fifth International Conference on Evolvable Systems: From Biology to Hardware*, 2606:93–104, Mar 2003.

[7] T. Fogarty, J. F. Miller, and P. Thomson. Evolving Digital Logic Circuits on Xilinx 6000 Family FPGAs. *The 2nd Online Conference on Soft Computing in Engineering Design and Manufacturing*, pages 299–305, Springer-Verlag, London, 1998.

[8] J. F. Miller, T. Fogarty, and P Thomson. Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science, John Wiley and Sons, Chichester*, pages 105–131, 1998.

[9] C. A. Coello and A. H. Aguirre,. Evolutionary Multiobjective Design of Combinational Logic Circuits. *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware*, pages 161–170, Jul 2000.

[10] C. A. Coello, A. D. Christiansen, and A. H. Aguirre. Ant Colony System for the Design of Combinational Logic Circuits. *Evolvable Systems: From Biology to Hardware, Edinburgh, Scotland*, pages 21–30, April Springer Verlag, 2000.

[11] C. A. Coello and Hernandez Luna and Erika and A.H. Aguirre. Use of Particle Swarm Optimization to Design Combinational Logic Circuits. *Evolvable Systems: From Biology to Hardware. 5th International Conference, ICES 2003*, 2606:398–409, Mar 2003.

[12] R. S. Zebulum, M. A. Pacheco, and Marley Vellasco. Evolvable Systems in Hardware Design: Taxonomy, Survey and Applications. *Evolvable System: From Biology to Hardware. Proceeding of the First International Conference, ICES 96 Tsukba, Japan, Lecture Notes in Computer Science*, 1259:344–358, Oct. 1997.

[13] R. S. Zebulum, M. A. Pacheco, and Maria Vellasco. *Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*. CRC Press, 2002.

[14] Xin Yao and Tetsuya Higuchi. Promises and Challenges of Evolvable Hardware. *IEEE Trans. on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, 29(1):87–97, 1999.

[15] Srinivas Devadas and Sharad Malik. A Survey of Optimization Techniques Targeting Low Power VLSI Circuits. *32nd ACM/IEEE Design Automation Conference*, pages 242–247, 1995.

[16] T. Stutzle and H. Hoos. Improvements of the ant system: Introducing MAX-MIN ant system. *Proceeding of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 245–249, 1997.

[17] MOSIS Standard Cell Library for CMOS, http://www.mosis.org/Technical/Designsupport/std-cell-library-scmos.html.