Fuzzified Ant Colony Optimization Algorithm for Efficient Combinational Circuits Synthesis

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

Abstract- With the increasing demand for high quality, more efficient, less areaand less power circuits , the problem of logic circuit design has become a multiobjective optimization problem. In this paper, multiobjective optimization of logic circuits based on a fuzzified Ant Colony (ACO) algorithm is presented. The results obtained using the proposed algorithm are compared to those obtained using SIS in terms of area, delay and power for some known circuits. It is shown that the circuits produced by the proposed algorithm are better as compared to those obtained by SIS.

1 Introduction

Synthesis of digital circuits can be stated as the process of assembling a collection of logic components to perform a specified function using a target technology. The obtained circuits are optimized for a number of objectives and subject to some constraints, such as area, delay and power.

The classical logic synthesis algorithms include the optimization of two quality measures, namely: area and performance [1]. The design objective can be either minimizing the area or maximizing the performance. Optimization can be subject to constraints, such as upper bound on area, as well as upper bounds on performance and lower bound on delay.

The possible configurations of a circuit are many. These different feasible implementations of a circuit define its design space. Figure 1 shows an example design space of a 2-bit adder circuit obtained using SIS, considering delay and area of the circuit.

The design space consists of a finite set of design points. If the size of the circuit as well as the design objectives are increased, the number of design points could be huge. This will increase the difficulty in finding the optimal structure for a given circuit. Hence, current available techniques divide the circuit design problem into a number of subproblems with lower dimensionality. However, this approach is somehow constrained both by the training and experience of the designer and by the amount of domain specific knowledge available. On the other hand, iterative heuristics can work on a larger space, and through the process of assemble and test, candidate solutions can be built and evaluated. An optimal solution could evolve from this process.

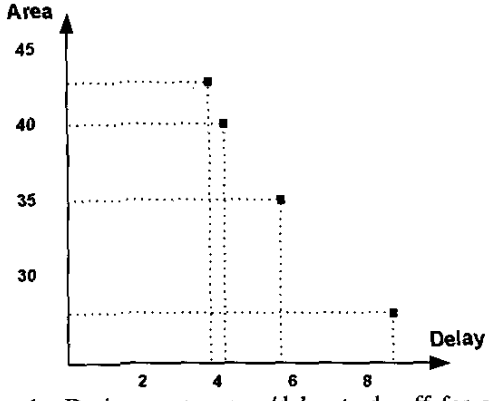


Figure 1: Design space: area/delay trade-off for a 2-bit adder.

A number of researchers have worked on evolutionary logic design. Louis [2], Miller [3, 4] and Colleo [5], to name a few. They have used different heuristics such as: Genetic Algorithm, Ant Colony and Simulated Evolution. This paper is a continuation of our previous work in [6]. In this paper, a multiobjective evolutionary logic design based on Ant Colony Optimization (ACO) is proposed. Fuzzy logic is used to model the multiobjective cost function. The goal is to find functionally correct circuits optimized in terms of area, delay and power. This paper is organized as follows. Section 1 gives a brief introduction to the problem of evolutionary logic synthesis. Section 2 describes some background material on fuzzy logic. The proposed fuzzy fitness function is given in Section 3. Section 4 describes the proposed approach. Experimental results and comparison are given in Section 5. Finally, conclusion is given in Section 6.

2 Fuzzy Logic

Fuzzy Logic was introduced by Lofti A. Zadeh in [7]. During the past decades, fuzzy logic has found numerous applications in the field of engineering and control [8]. In the field of VLSI design, several techniques based on fuzzy logic are reported in the literature [9, 10].

A fuzzy set A of universe of discourse X is defined as $A = \{(x, \mu_A(x)) \mid all\ x \in X\}$, where X is a space point and $\mu_A(x)$ is a membership function of x being an element

0-7803-8515-2/04/$20.00 ©2004 IEEE

of A. A membership function $\mu_A(x)$ is a mapping of x in A that maps X to the membership space M. The range of the membership function is a subset of the non-negative real numbers whose boundaries are finite. Elements with zero degree of membership are normally not listed.

Fuzzy logic establishes approximate truth value of propositions based on linguistic variables and inference rules [11]. A linguistic variable is a variable whose values are words or sentences in natural or artificial language. It is concerned with the use of fuzzy values that captures the meaning of words, human reasoning and decision-making. An example of linguistic variable is circuit's area. This variable can be expressed by linguistic values like very small, small, average, large and very large circuit, rather than crisp values such as 20 μm^2, 30 μm^2, 50 μm^2, 75 μm^2, and 100 μm^2.

2.1 Multiobjective Optimization Using Fuzzy Logic

Approximate reasoning can be made based on linguistic variables and their values. Rules can be generated based on previous experience. The rules are expressed as **If** ... **Then** statements. Connectives such as AND and OR can be used in approximate reasoning to join two or more linguistic values.

In optimization problems, the linguistic value used in the consequent part identifies the fuzzy subset of good solutions. Therefore, the result of evaluation of the antecedent part identifies the degree of membership in the fuzzy subset of good solutions according to the fuzzy rule in question. If more than one rule is used to perform decision-making, each rule can be evaluated to generate a numerical value. Then, these numerical values from various evaluations of different rules can be combined to generate a crisp value on a higher level of hierarchy.

Consider, for example, the circuit design problem targeting minimization of area, delay, and power consumption. Three linguistic variables *area, delay* and *power* introduced. Good solutions can be characterized by the following fuzzy rule.

> *If the circuit has (small area) and (less delay) and (less power consumption) then it is a good solution.*

2.2 Ordered Weighted Averaging (OWA) Operator

In the traditional fuzzy logic, the minmax operators are used to build the above fuzzy rule. However, it was shown in [12] that these operators can lead to undesirable behavior. This behavior has led to the development of other fuzzy operators such as the *Ordered Weighted Averaging (OWA)* operator. This operator allows easy adjustment of the degree of "AND-ing" and "OR-ing" embedded in the aggregation. According to [13], "OR-like" and "AND-like" OWA for two fuzzy sets A and B are implemented as given in Equations 1 and 2 respectively.

$$\mu_{A\cup B}(x) = \lambda \times \max(\mu_A, \mu_B) + (1-\lambda) \times \frac{1}{2}(\mu_A + \mu_B) \quad (1)$$

$$\mu_{A\cap B}(x) = \lambda \times \min(\mu_A, \mu_B) + (1-\lambda) \times \frac{1}{2}(\mu_A + \mu_B) \quad (2)$$

where λ is a constant parameter in the range [0,1] and represents the degree to which OWA operator resembles a pure "OR" or pure "AND" respectively.

3 Fuzzy Fitness Function

In this section, a fuzzy-based fitness function is formulated. Similar to the weighted sum approach proposed in [6], the overall fitness of a solution consists of two parts: functional fitness and objective fitness. In this approach, membership functions are used and these membership functions will be aggregated into a single function using a fuzzy operator.

Recall to the formulation of functional fitness used in [6], FF lies in the range [0.5, 1]. Thus, the membership function for functional fitness is shown in Equation 3.

$$\mu_{FF} = \begin{cases} FF & if\ 0.5 \leq FF \leq 1 \\ 0 & otherwise \end{cases} \quad (3)$$

Area as Optimization Objective

The lower bound on area can be estimated by referring to the VLSI circuit design and logic synthesis principles. For any n-input single-output circuit, the minimum area for the circuit is equal to the area of $(n-1)$ 2-input gates representing binary tree structure. Since any circuit can be implemented using NAND gates and NAND gates happen to be the smallest among other primitives gates (except NOT gate), then the minimum area is:

$$min_{area} = (n-1) \times Area(NAND\ gate)$$

In order to guide the search intelligently, a maximum value must be carefully estimated. For this purpose, SIS tools [1] are used to obtain circuits with minimum area. In this context, *rugged.script* is used to generate the circuits' netlist files. These files are then fed to our own tool to obtain the estimated value for area, delay and power consumption. The reason behind this is twofold. Firstly, because the delay optimization in SIS does not consider switching delay. Secondly, SIS does not consider power optimization.

Since our objective is to obtain circuits with better performance than those obtained using SIS, the estimated values of area, delay and power of circuits obtained using SIS are used as the target values. In the case of area as optimization objectives, the target area is equal to the area of circuits

obtained using SIS and denoted as tg_{area1} (see Figure 2). Thus, the membership function for area as optimization objectives is:

$$\mu_{area\_obj} = \begin{cases} 1 & 0 \leq area < min_{area} \\ 1 - p1 & min_{area} \leq area < tg_{area1} \\ 0 & otherwise \end{cases} \quad (4)$$

with

$$p1 = \frac{(area - min_{area})}{tg_{area1} - min_{area}}$$

The shape of the membership function is depicted as the bold line shown in Figure 2.

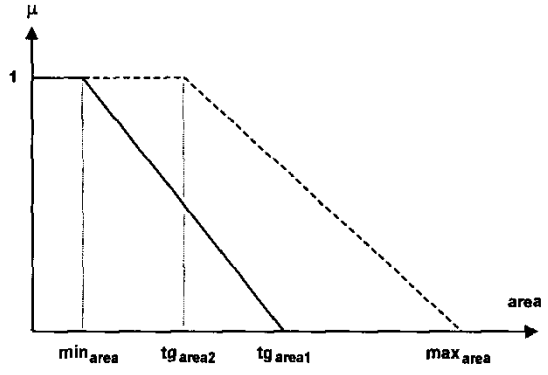


Figure 2: Membership function for area.

Area as Constraint

In this case, the area of a circuit obtained from SIS is used as target value. For this purpose, the max_{area} and tg_{area2} should be defined. The following settings are applied, $tg_{area2} = k_1 \times tg_{area1}$ and $max_{area} = k_2 \times tg_{area1}$, k_1, $k_2 \in \Re$, $0 < k_1 \leq 1$, $k_2 \geq 1$. In this case, the membership function is given by:

$$\mu_{area\_con} = \begin{cases} 1 & 0 \leq area < tg_{area1} \\ 1 - p2 & 1 \leq area < max_{area} \\ 0 & otherwise \end{cases} \quad (5)$$

with

$$p2 = \frac{area - k_1}{max_{area} - k_1}$$

The shape of the membership function is depicted as dashed line shown in Figure 2.

Delay as Optimization Objective

The minimum delay (min_{delay}) is estimated as the delay of two-level logic consisting of NAND gates without considering the switching delay. The tg_{delay1} is estimated from circuit generated using SIS with *delay.script* executed. The membership function for delay as optimization objectives is:

$$\mu_{delay\_obj} = \begin{cases} 1 & 0 \leq delay < min_{delay} \\ 1 - p1 & min_{delay} \leq delay < tg_{delay1} \\ 0 & otherwise \end{cases} \quad (6)$$

with

$$p1 = \frac{delay - min_{delay}}{tg_{delay1} - min_{delay}}$$

The shape of the membership function is depicted as bold line shown in Figure 3.

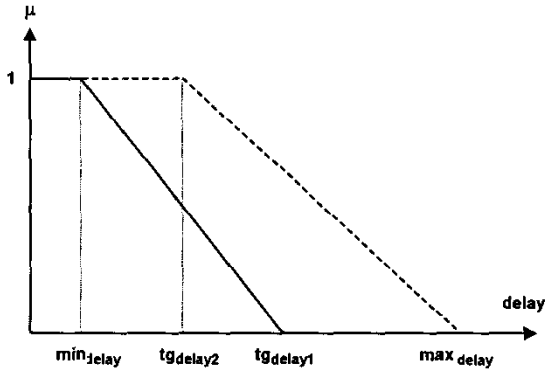


Figure 3: Membership function for delay.

Delay as Constraint

In this case, the following settings are applied, $tg_{delay2} = k_1 \times tg_{delay1}$ and $max_{delay} = k_2 \times tg_{delay1}$, k_1, $k_2 \in \Re$, $0 < k_1 \leq 1$, $k_2 \geq 1$. In this case, the membership function is given by:

$$\mu_{delay\_con} = \begin{cases} 1 & 0 \leq delay < tg_{delay1} \\ 1 - p2 & 1 \leq delay < max_{delay} \\ 0 & otherwise \end{cases} \quad (7)$$

with

$$p2 = \frac{delay - 1}{max_{delay} - 1}$$

The shape of the membership function is depicted as dashed line shown in Figure 3.

Power as Optimization Objective

The minimum power (min_{pow}) is estimated as the power consumption of minimum area circuit in which each gate has the least switching activity. It is assumed that for a given truth table, the output of each gate will be '1' only once.

With $L = length\ of\ truth\ table$, the minimum power consumption (switching activity) can be estimated as follows.

$$min_{power} = 2 \cdot \frac{L-1}{L^2} \cdot capacitance(NAND)$$

The tg_{pow1} is estimated from minimum area circuit generated by SIS. The membership function for power as optimization objectives is:

$$\mu_{power\_obj} = \begin{cases} 1 & 0 \leq power < min_{pow} \\ 1 - p1 & min_{pow} \leq power < tg_{pow1} \\ 0 & otherwise \end{cases} \quad (8)$$

with

$$p1 = \frac{power - min_{pow}}{tg_{pow} - min_{pow}}$$

The shape of the membership function is depicted as bold line shown in Figure 4.

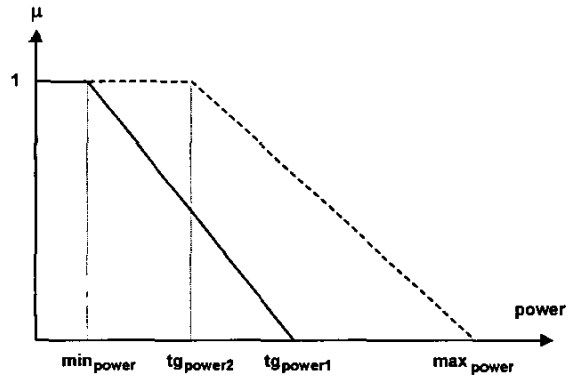


Figure 4: Membership function for power.

Power as Constraint

The following settings are applied, $tg_{pow2} = k_1 \times tg_{pow1}$ and $max_{pow} = k_2 \times tg_{pow1}$, $k_1, k_2 \in \Re$, $0 < k_1 \leq 1$, $k_2 \geq 1$. In this case, the membership function is given by:

$$\mu_{power\_con} = \begin{cases} 1 & 0 \leq power < tg_{pow1} \\ 1 - p2 & 1 \leq power < max_{pow} \\ 0 & otherwise \end{cases} \quad (9)$$

with

$$p2 = \frac{power - 1}{max_{pow} - 1}$$

The shape of the membership function is depicted as dashed line shown in Figure 4.

The type of membership function (as objective or constraint) for each merit determine the goal of the heuristics. For example, optimization or area can be performed by using μ_{area_obj} (area as objective), μ_{delay_con} (delay as constraint) and μ_{power_con}. Then, these three membership functions are aggregated into one unit (the objective fitness) using the OWA operator [12] as follows:

$$\mu_{OF} = \lambda \times Min(\mu_{area\_obj}, \mu_{delay\_con}, \mu_{power\_con}) + (1-\lambda) \times \tfrac{1}{3}(\mu_{area\_obj} + \mu_{delay\_con} + \mu_{power\_con}) \quad (10)$$

The overall fitness of a cell is formulated as follows.

$$Fit = W_f \cdot \mu_{FF} + (1 - W_f) \cdot \mu_{OF} \quad (11)$$

Where W_f is the weight for functional fitness. The value of W_f must be large enough in order to have better functionality of the circuit. However, it should not be too large in order to get better quality solutions in terms of design objectives.

4 Proposed Approach

A circuit is modelled as a matrix M of size $n \times m$. Each cell of the matrix contains a triplet of *attributes* consisting of the type of gate used and its corresponding inputs, i.e., the row indices of the preceding column (see Figure 5).

Input 1	Input 2	Gate type

Figure 5: Representation of a cell in the matrix.

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

Table 1: Gate ID, gate name and output of the gate, considering input a and b.

The values in input 1 and input 2 indicate the row indices from which the current cell is getting its input from. The value of the gate type indicates the type of the gate being assigned to that cells assuming a predetermined set of gate types (see Table 1). The input of a gate at position (i, j) can only be connected to the output of a cell

at $(i', (j-1))$ and i' can be any row index in column $(j-1)$.

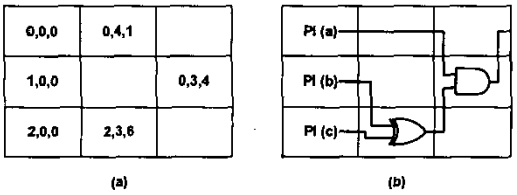


Figure 6: Example of a circuit and its encoding.

Consider the example shown in Figure 6. Cell(1,2) whose attribute is (0,3,4) is an AND gate (according to Table 1). The first input of the AND gate in this cell is connected to the output of cell(0,1), which is a WIRE, and the second input is connected to the output of cell(2,1).

4.1 Solution Construction

In the beginning, the cells of the matrix M are filled with randomly generated attributes. The ants originate from a dummy cell called *nest* (see Figure 7), and traverse each state (a cell in a column) until it reaches the last column or a cell that has no successor.

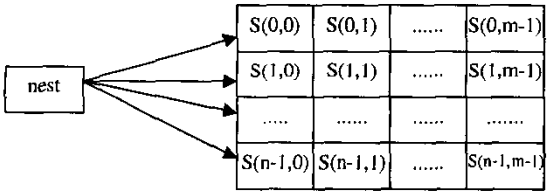


Figure 7: Nest cell and matrix M for ant to be traversed.

The selection of edges to traverse is determined by a stochastic probability function. It depends on the pheromone value (τ) and the heuristic value (η) of the edge. The probability of selecting next cell is formulated below [14]:

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in \aleph_i^k} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}} \qquad (12)$$

The value of α and β imply the preference of the search, whether it depends more on the pheromone value or the heuristic value, respectively. 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) between cell i and j is formulated as follows.

$$\eta = 0.5 + (\mu_{FF}(j) - \mu_{FF}(i)) \qquad (13)$$

Algorithm Modified ACO (MACO)

Begin
  For $0 < i < iteration$
    $Filling\_the\_Matrix$
    $Ant\_Activity$
    $Removing\_Unfit\_Cells$
  EndFor
End

Figure 8: Modified ACO algorithm for logic design.

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

After the ant finish its tour, pheromone update is performed using the following equation:

$$\tau(t) = (1-\rho) * \tau(t) + Fit(t) \qquad (14)$$

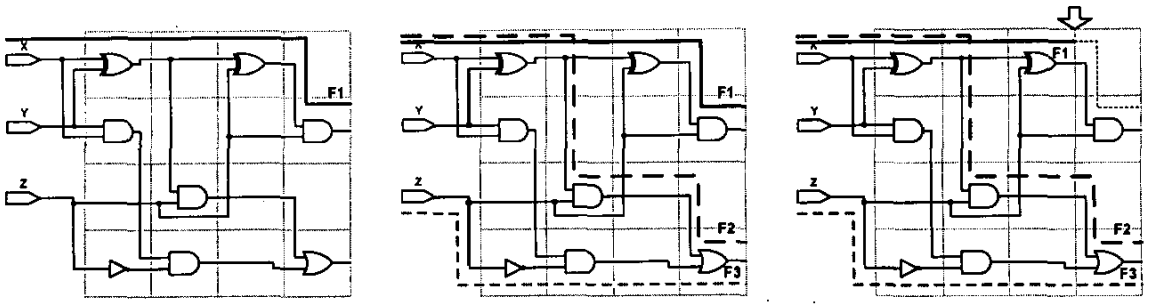
where $Fit(t)$ denotes the overall fitness of the solution that the ants built, ρ is pheromone evaporation rate.

When all ants finish their tours, the solutions provided will be evaluated. All cells that are included in the best solution of the current matrix will be kept. Note that, this solution may not represent the intended function. All unneeded cells will be removed. These empty cells will be filled up again in the next iteration. The ants will then traverse the new matrix and return the best possible solution. If the stopping criteria is not met, the same procedure will be repeated. Figure 8 shows the pseudocode of the approach.

4.2 The Intelligent Ant

The *Filling* and *Removing* cells procedures in MACO algorithm shown in Figure 8 are performed to handle the limitation of ACO algorithm due to the huge search space of circuit design problem. To further accommodate some improvements the *Intelligent Ant* is proposed.

The original ACO algorithm works on a clearly defined graph where the number of nodes and/or edges is mostly static and the quality of best solution is unknown. On the other hand, the result of evolutionary logic synthesis must be a functionally correct circuit optimized according to the cost function. While traversing the matrix, each ant must seek good solution in terms of circuit's functionality first. Since the length of the tour is limited by the size of the matrix, the ant should have intelligence to select which part of its tour that provides the best solution in terms of functional fitness. The remaining path will be removed from its memory. Using this approach, the ant will provide better partial



(a) (b) (c)

Figure 9: Example of intelligent ant (a) First solution found, F1 (b) F1,F2, and F3 are good solutions. F1 is the best solution (c) Quality of F1 can be improved by discarding the path after the arrow sign

Table 2: Comparison with SIS in area optimization.

Circuit	Proposed Algorithm			SIS			% Improvement		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
majority	13851	4.57	5.06	14823	6.28	5.41	6.56	27.18	6.48
xor8	20655	5.90	9.32	27945	27.69	10.82	26.09	78.70	13.89
xor9	23328	8.84	10.65	33048	33.25	12.65	29.41	73.40	15.83
add2	24300	11.48	9.96	29889	17.22	11.38	18.70	33.31	12.48
mul2	12636	3.56	4.66	18225	6.59	5.56	30.67	45.94	16.21
add3	49086	21.96	18.474	42282	24.99	15.68	-16.09	12.13	-17.79
mul3	59292	15.03	17.541	112752	43.39	37.75	47.41	65.36	53.53

solution in every iteration and that the best solution would emerge at the end of the iterative process.

Consider the example shown in Figure 9. The required Boolean function is a 3-bit odd parity circuit. Figure 9 (a) shows one possible solution in the current matrix, denoted as $F1 = ((X \oplus Y) \oplus Z) \cdot Z$. However, there are 2 other existing solutions in the matrix, namely $F2 = ((X \oplus Y) \cdot Z) + XYZ'$ and $F3 = ((X \oplus Y) \cdot Z) + XYZ'$. Note that $F2$ and $F3$ are basically the same function, but since they are found by different ants (hence different path), they are treated as different solutions. From these three solutions, we can easily find that $F1$ is the best solution. However, the quality of $F1$ can be improved by discarding some part of its path, since the 3-bit odd parity circuit is $F = X \oplus Y \cdot Z$. Hence, the intelligent ant will detect the solution and record the required path only (from input until the arrow sign) shown in Figure 9 (c).

5 Experiments and Results

In this section, comparison of the results obtained using the proposed algorithm with the results obtained using SIS is presented. However, since SIS does not perform power optimization, the comparison is made only for area and delay optimization.

5.1 Case 1: Area Optimization

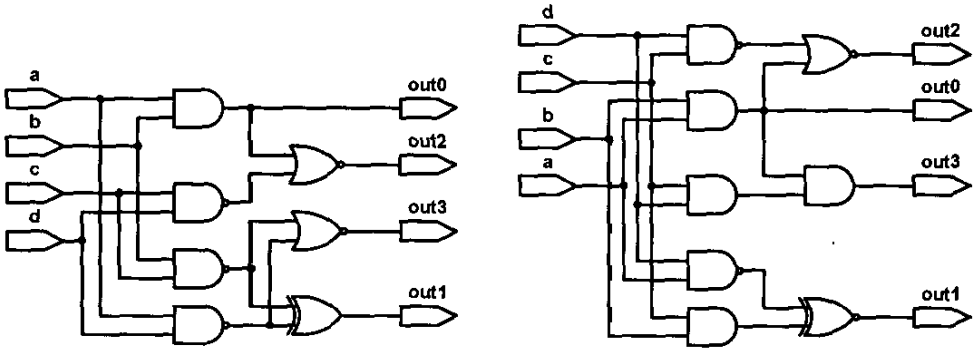
The *rugged.script* is used in order to get the area minimized circuits in SIS. The obtained circuits are then mapped for area minimization. Table 4.2 shows the results for area optimization for both techniques. The table shows that for single-output circuits, the highest improvements are obtained in the case of 8-bit and 9-bit odd parity circuits. The parity circuits are best represented using XOR (XNOR) gates. Unfortunately, SIS is unable to perform XOR decomposition. Thus, the parity circuits obtained by SIS requires larger area as compared to the ones obtained using the proposed algorithm. For multiple-output circuits, the improvement in area varies. The highest improvements are observed in the case of 2-bit and 3-bit multiplier circuits. However, the proposed algorithm failed to deliver better circuit in terms of area in the case of add3 circuit, which is the largest circuit used as test case.

5.2 Case 2: Delay Optimization

For delay optimization, the results from SIS are obtained by executing *delay.script* mapped for delay minimization. The test cases used are the same circuits used for area optimization.

Table 3: Comparison with SIS in delay optimization.

Circuit	Proposed Algorithm			SIS			% Improvement		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
majority	16038	4.19	5.02	18711	7.53	5.40	14.29	44.34	7.11
xor8	20655	5.90	9.32	32805	9.53	11.65	37.04	38.11	20.04
xor9	27216	8.84	11.48	41067	15.42	14.15	33.73	42.64	18.85
add2	31347	8.957	11.463	50787	11.77	14.63	38.28	23.90	21.64
mul2	18225	2.96	5.99	25272	4.33	7.16	27.88	31.57	16.30
add3	53703	12.979	21.484	118827	19.20	35.21	54.81	32.40	38.98
mul3	74358	13.138	21.645	174231	31.66	47.16	57.32	58.51	54.10



(a) (b)

Figure 10: 2-bit multiplier circuit (a) Using area optimization (b) Using delay optimization

5.3 Case Study: 2-bit Multiplier Circuit

Figure 10 shows the result obtained using area and delay optimization for a 2-bit multiplier circuit. As we can see from the figure, in area optimization, the proposed approach prefers the use of NAND and NOR gates (Figure 10 (a)). This results to circuit with minimum area. However, the longest delay of this circuit is bigger as compared to the one produced using delay minimization (see Figure 10 (b)). This delay is attributed to the load factor caused by fan-out existing in the circuit (see [6] for delay cost function).

6 Conclusion

In this paper, an ACO-based evolutionary logic synthesis technique have been proposed. Comparison of the proposed approach with SIS is shown. The proposed approach has shown that it is capable of producing optimized combinational circuits. In addition, the results obtained by the proposed algorithm are better in terms of area, delay and power as compared to SIS.

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

Bibliography

[1] E. M. Sentovic, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, University of California, Berkeley, May 1992.

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

[3] 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.

[4] 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.

[5] C. A. Coello, A. D. Christiansen, and A. H. Aguirre. Towards Automated Evolutionary Design of Combinational Circuits. *Computers and Electrical Engineering, Pergamon Press*, 27(1):1–28, Jan. 2001.

[6] Bambang A. B. Sarif. A Modified Ant Colony Algorithm for Evolutionary Design of Digital Circuits. *IEEE 2003 International Conference on Evolutionary Computation*, pages 708–715, December 2003.

[7] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, June 1965.

[8] J. M. Mendel. Fuzzy logic systems for engineering: A tutorial. *IEEE Proceeding*, 83(3):345–377, Mar. 1995.

[9] E. Shragowitz, Jun-Yong Lee, and E. Q. Kang. Application of fuzzy logic in computer aided design. *IEEE Trans. on Fuzzy Systems*, 6(1):163–172, Feb. 1998.

[10] Sadiq M. Sait and Youssef, H. and J. A. Khan and El-Maleh, A. Fuzzy simulated evolution for power and performance optimization of VLSI placement. *Proceedings of International Joint Conference on Neural Networks, IJCNN '01.*, 1:738 –743, July 2001.

[11] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transaction Systems Man. Cybern*, SMC-3(1):28–44, 1973.

[12] Ronald R. Yager. On Ordered Weighted Averaging Aggregation Operators in Multicriteria Decision Making. *IEEE Transaction on Systems, MAN, and Cybernetics*, 18(1):183–190, Jan 1988.

[13] R. Yager. Second Order Structures in multi-criteria decision making. *International Journal of Man-Machine Studies*, pages 36:553–570, 1992.

[14] M. Dorigo and G. Di Caro. *New Ideas in Optimisation*. McGraw Hill, London, UK, 1999.