EFFICIENT COMBINATIONAL CIRCUITS DESIGN THROUGH FUZZIFIED ANT COLONY OPTIMIZATION ALGORITHM

Mostafa Abd-El-Barr, Sadiq M. Sait, Bambang A. B. Sarif, and Uthman Al-Saiari

Computer Engineering Department
KFUPM, Dhahran-31261, Saudi Arabia
{mostafa, sadiq, sarif, saiarios} @ccse.kfupm.edu.sa

ABSTRACT

With the increasing demand for high quality, more efficient and less area circuits, the problem of logic circuit design has become a multiobjective optimization problem. In this paper, a 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. It is shown that the circuits produced by the proposed algorithm are better as compared to those obtained by SIS.

1. INTRODUCTION

In conventional logic design, circuit designers begin with a precise specification in the form of truth tables or Boolean expressions. These expressions are manipulated by applying logic synthesis algorithms to obtain the optimal representations. These will be either in two-level, multi-level or Reed Muller representation forms. Iterative heuristics work on a larger space, and through the process of assemble and test, candidate solutions are built and evaluated. An optimal solution could evolve from this process.

The first work in evolutionary design of logic circuits was proposed by Louis [1]. Later, the work of Thompson [2] that produced a tone discriminator circuit without input clock hinted to the possibility of a new way of designing circuits. The work of Miller [3] built some arithmetic circuits that cannot be produced by human designer’s conventional methods. Coello et.al [4] proposed a similar approach to evolve a circuit, which they showed was better than that of Miller’s. A complete review and taxonomy of the field can be found in [5].

Ant Colony Optimization (ACO) algorithm [6] is a new meta-heuristic that combines distributed computation, autocatalysis (positive feedback) and constructive greediness in finding optimal solutions for combinatorial optimization problems. Unlike Genetic Algorithms (GAs), ACO involves cooperating agents. 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.

2. ANT COLONY OPTIMIZATION ALGORITHM

The ACO algorithm [6] has been inspired by the behavior of real ants. It was observed that real ants were able to select the shortest path between their nest and food source, in the existence of alternate paths between the two. The search is made possible by an indirect communication known as stigmergy amongst the ants. While traveling their way, ants deposit a chemical substance, called pheromone, on the ground. When they arrive at a decision point, they make a probabilistic choice that is biased by the intensity of pheromone they smell. This behavior has an autocatalytic effect because of the very fact that choosing a path will increase the probability that it will be chosen again by future ants. When they return back, the probability of choosing the same path is higher (due to the increase of pheromone). New pheromone will be released on the chosen path, which makes it more attractive for future ants. Shortly, all ants will select the shortest path.

In the ACO algorithm, the optimization problem is formulated as a graph \( G = (C, L) \), where \( C \) is the set of components of the problem, and \( L \) is the set of possible connections or transitions among the elements of \( C \). The solution is expressed in terms of feasible paths on the graph \( G \), with respect to a set of given constraints.

3. PROPOSED APPROACH

Consider the Boolean function \( f = \overline{x}yz + xy \overline{z} + x \overline{y}z \). Figure 1 shows a graph of some possible paths leading to the intended function \( f \) starting from literal \( x \). The ant will traverse the paths by selecting the edges through a probabilistic process.

The number of possible paths leading to function \( f \) shown in Figure 1 is more than eleven. The number of all possible
paths could be huge. It is impractical to traverse all those paths. We need to modify the ACO algorithm to handle the search space.

3.1. Circuit Encoding and Representation

A circuit is modelled as a matrix $M$ of size $n \times m$. Each cell of the matrix contains triplet of attributes consisting of the row indices of the preceding column (as input1 and input2) and the type of gate used. There are 10 types of gate available. Table 1 shows these gates.

<table>
<thead>
<tr>
<th>Gate ID</th>
<th>Gate</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>WIRE1</td>
<td>$a$</td>
</tr>
<tr>
<td>1</td>
<td>WIRE2</td>
<td>$b$</td>
</tr>
<tr>
<td>2</td>
<td>NOT1</td>
<td>$\bar{a}$</td>
</tr>
<tr>
<td>3</td>
<td>NOT2</td>
<td>$\bar{b}$</td>
</tr>
<tr>
<td>4</td>
<td>AND</td>
<td>$a \cdot b$</td>
</tr>
<tr>
<td>5</td>
<td>OR</td>
<td>$a + b$</td>
</tr>
<tr>
<td>6</td>
<td>XOR</td>
<td>$a \oplus b$</td>
</tr>
<tr>
<td>7</td>
<td>NAND</td>
<td>$a \cdot \bar{b}$</td>
</tr>
<tr>
<td>8</td>
<td>NOR</td>
<td>$a + \bar{b}$</td>
</tr>
<tr>
<td>9</td>
<td>XNOR</td>
<td>$a \oplus \bar{b}$</td>
</tr>
</tbody>
</table>

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

Consider the example shown in Figure 2. Cell(1,2) whose attribute is (0,3,4) is an AND gate (according to Table 1). The first input of the AND gate of 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).

3.2. Fitness Function Calculation

The fitness of a solution contains two parts, namely functional fitness and objective 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 ($FF$) function calculation are reported in the literature [5]. The most commonly used one is the ratio of the number of hits to the length of the truth table. This can be formulated as follows.

$$FF(i) = \frac{\text{Number of hits of cell } i}{\text{Length of truth table}}$$

The number of hits is defined as the number of correct matchings between the output patterns obtained at cell $i$ and the truth table of the intended function. The solution has to be ‘inverted’ if the value of $FF(i)$ is less than 0.5. Therefore, the formulation normalized $FF(i)$ below is used.

$$FF_n(i) = \max\{FF(i), 1 - FF(i)\} \quad (1)$$

The objective fitness ($OF(i)$) of cell $i$ is a measure of the quality of solution in terms of optimization objectives such as area, delay, gate count, and power consumption. It considers two aspects: constraints satisfaction and multiobjective optimization. In this paper, fuzzy logic is used to represent the cost function for area, delay and power. For this purpose, SIS tools [7] are used to estimate the target value (area, delay, and power) of the intended circuits. The shape of the membership function for area is depicted in Figure 3.

$$F(i) = W_f \cdot FF_n(i) + (1 - W_f) \cdot OF(i) \quad (2)$$
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.

### 3.3. Solution Construction

At first, the cells of the matrix $M$ are filled with randomly generated attributes. Then, each ant will traverse the matrix. These ants originate from a dummy cell called $nest$, and traverse each state (a cell in a column) until it reaches the last column or a cell that has no successor.

The selection of edges to traverse is determined by a stochastic probability function. It depends on the pheromone value ($\tau$) and the heuristic value ($\eta$) of the edge. The probability of selecting next cell is formulated below \[6\]:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{j \in \Omega_i} [\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}$$

(3)

The value of $\alpha$ and $\beta$ 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 ($\eta$) between cell $i$ and $j$ is formulated as follows:

$$\eta = 0.5 + (FF(j) - FF(i))$$

(4)

The addition of 0.5 in the calculation of $\eta$ is meant to normalize the value of $\eta$ into $[0,1]$. A decrease in functional fitness means that the value of $\eta$ is in the range of $[0,0.5]$, while an increase in the functional fitness makes the value of $\eta$ in the range of $[0.5,1]$. When all ants finish their tour, pheromone update is performed. The pheromone update is performed using the following equation:

$$\tau(t) = (1 - \rho) \cdot \tau(t) + \lambda \cdot FF(t)$$

(5)

where $FF(t)$ denotes the overall fitness of the solution that the ants built, $\rho$ is pheromone evaporation rate and $\lambda$ is a constant.

When all ants finish their movement, the matrix $M$ is checked to see which cells of the matrix deserve to be kept. The cells that are not included in the best solution in the current iteration will be removed. These empty cells will be filled at the beginning of the next iteration. If it has not reached the maximum iteration, the procedure will be repeated. Otherwise, the best solution is returned. Figure 4 shows the pseudocode of the approach.

**Modified ACO algorithm**

```plaintext
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
```

**Fig. 4. Modified Ant Colony Algorithm.**

### 4. EXPERIMENTS AND RESULTS

In this section, comparison of the results obtained using the proposed algorithm with the results obtained using SIS is presented. It should be noted that SIS does not consider capacitance load in delay calculation and power optimization. Therefore, the results obtained from SIS are in the form of netlist files. These files are imported to the cost function calculation procedures of the proposed algorithm to determine the area, delay and power of the circuits.

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 2 shows the results for area optimization for both techniques. The table shows that for single-output circuits, the best 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 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.

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. As can be seen from Table 3, in contrast with area optimization, the results of delay optimization is very positive. The reason behind this is the following. ACO can be easily modelled as a shortest path problem. Since delay can be said proportional to the length of the path, ACO algorithm, which is the basis of the proposed algorithm, provides a good computational tool for delay optimization problem.
Table 2. Comparison with SIS in area optimization.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Proposed Algorithm</th>
<th>SIS</th>
<th>% Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area</td>
<td>Delay</td>
<td>Power</td>
</tr>
<tr>
<td>majority</td>
<td>13851</td>
<td>4.57</td>
<td>5.06</td>
</tr>
<tr>
<td>xor8</td>
<td>20655</td>
<td>5.90</td>
<td>9.32</td>
</tr>
<tr>
<td>xor9</td>
<td>23232</td>
<td>8.84</td>
<td>10.65</td>
</tr>
<tr>
<td>add2</td>
<td>24300</td>
<td>11.48</td>
<td>9.96</td>
</tr>
<tr>
<td>mul2</td>
<td>12636</td>
<td>3.56</td>
<td>4.66</td>
</tr>
<tr>
<td>add3</td>
<td>49086</td>
<td>21.96</td>
<td>18.47</td>
</tr>
<tr>
<td>mul3</td>
<td>59292</td>
<td>15.03</td>
<td>17.54</td>
</tr>
</tbody>
</table>

Table 3. Comparison with SIS in delay optimization.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Proposed Algorithm</th>
<th>SIS</th>
<th>% Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area</td>
<td>Delay</td>
<td>Power</td>
</tr>
<tr>
<td>majority</td>
<td>16038</td>
<td>4.19</td>
<td>5.02</td>
</tr>
<tr>
<td>xor8</td>
<td>20655</td>
<td>5.90</td>
<td>9.32</td>
</tr>
<tr>
<td>xor9</td>
<td>27216</td>
<td>8.84</td>
<td>11.48</td>
</tr>
<tr>
<td>add2</td>
<td>31347</td>
<td>8.957</td>
<td>11.463</td>
</tr>
<tr>
<td>mul2</td>
<td>18225</td>
<td>2.96</td>
<td>5.99</td>
</tr>
<tr>
<td>add3</td>
<td>53703</td>
<td>12.979</td>
<td>21.484</td>
</tr>
<tr>
<td>mul3</td>
<td>74358</td>
<td>13.138</td>
<td>21.645</td>
</tr>
</tbody>
</table>

5. CONCLUSION

In this paper, we have proposed an ACO-based evolutionary logic design technique. 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 heuristic for the Design of Combinational Logic Circuits”.

6. REFERENCES


