Ant Colony System for the Design of Combinational Logic Circuits

Carlos A. Coello Coello†, Rosa Laura Zavala G.‡, Benito Mendoza García‡,
and Arturo Hernández Aguirre§

†Laboratorio Nacional de Informática Avanzada
Rébsamen 80, A.P. 696
Xalapa, Veracruz, México 91090
`ccoello@xalapa.lania.mx`
‡MIA, LANIA-UV
Sebastián Camacho 5
Xalapa, Veracruz, México
`{rzavala,bmendoza}@mia.uv.mx`
§EECS Department
Tulane University
New Orleans, LA 70118, USA
`hernanda@eecs.tulane.edu`

Abstract. In this paper we propose an application of the Ant System (AS) to optimize combinational logic circuits at the gate level. We define a measure of quality improvement in partially built circuits to compute the distances required by the AS and we consider as optimal those solutions that represent functional circuits with a minimum amount of gates. The proposed methodology is described together with some examples taken from the literature that illustrate the feasibility of the approach.

1 Introduction

In this paper we extend previous work on the optimization of combinational logic circuits [1, 2], by experimenting with a metaheuristic: the ant system (AS) [6, 3].

The AS is a multi-agent system where low level interactions between single agents (i.e., artificial ants) result in a complex behavior of the whole ant colony. The idea was inspired by colonies of real ants, which deposit a chemical substance on the ground called *pheromone* [5]. This substance influences the behavior of the ants: they will tend to take those paths where there is a larger amount of pheromone. The AS was originally proposed for the traveling salesman problem (TSP), and according to Dorigo [6], to apply efficiently the AS, it is necessary to reformulate our problem as one in which we want to find the optimal path of a graph and to identify a way to measure the distances between nodes. This might not be an easy or obvious task in certain applications like the one presented in this paper. Therefore, we will provide with a detailed discussion of how to reformulate the circuit optimization problem as to allow the use of the AS, and we will present several examples to illustrate the proposed approach.

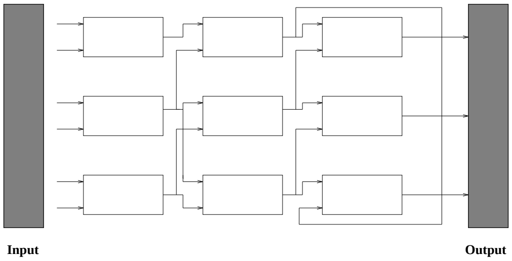

Fig. 1. Matrix used to represent a circuit to be processed by an agent (i.e., an ant). Each gate gets its inputs from either of the gates in the previous column.

2 Description of the Aproach

Since we need to view the circuit optimization problem as one in which we want to find the optimal path of a graph, we will use a matrix representation for the circuit as shown in Fig. 1. This matrix is encoded as a fixed-length string of integers from 0 to $N-1$, where N refers to the number of rows allowed in the matrix.

More formally, we can say that any circuit can be represented as a bidimensional array of gates $S_{i,j}$, where j indicates the *level* of a gate, so that those gates closer to the inputs have lower values of j. (Level values are incremented from left to right in Fig. 1). For a fixed j, the index i varies with respect to the gates that are "next" to each other in the circuit, but without being necessarily connected. Each matrix element is a gate (there are 5 types of gates: AND, NOT, OR, XOR and WIRE) that receives its 2 inputs from any gate at the previous column as shown in Fig. 1.

We have used this representation before with a genetic algorithm (GA) [1, 2]. The path of an agent (i.e., an ant) will then be defined as the sub-portion of this matrix (of a certain pre-defined maximum size) representing a Boolean expression (i.e., an ant will build a circuit while traversing a path). Each state within the path is a matrix position of the circuit and the distance between two states of the path is given by the increase or decrease of the cost of the circuit when moving from one state to the following. Such cost is defined in our case in terms of the number of gates used—i.e., a feasible circuit that uses the minimum amount of gates possible is considered optimal. The aim is to maximize a certain payoff function. Since our code was built upon our previous GA implementation, we adopted the use of fixed matrix sizes for all the agents, but this need not be the case (in fact, we could represent the Boolean expressions directly rather than using a matrix). The matrix containing the solution to the problem is built in a column-order fashion following the steps described next.

The gate and inputs to be used for each element of the matrix are chosen randomly from the set of possible gates and inputs (a modulo function is used when the relationship between inputs and matrix rows is not one-to-one). Each

Input 1	Input 2	Gate Type

Fig. 2. Encoding used for each of the matrix elements that represent a circuit.

state is, therefore, a triplet in which the first 2 elements refer to each of the inputs used (taken from the previous level or column of the matrix) and the third is the corresponding gate (chosen from AND, OR, NOT, XOR, WIRE (WIRE basically indicates a null operation, or in other words, the absence of gate) as shown in Fig. 2 (only 2-input gates were used in this work). For the gates at the first level (or column), the possible inputs for each gate were those defined by the truth table given by the user (a modulo function was implemented to allow more rows than available inputs).

One important difference between the statement of this problem and the TSP is that in our case not all the states within the path have to be visited, but both problems share the property that the same state is not to be visited more than once (this property is also present in some routing applications [4]).

When we move to another state in the path, a value is assigned to all the states that have not been visited yet and the next state (i.e., the next triplet) is randomly selected using a certain selection factor p. This selection factor determines the chance of going from state i to state j at the iteration t, and is computed using the following formula that combines the pheromone trail with the heuristic information used by the algorithm:

$$p_{i,j}^{k}(t) = f_{i,j}(t) \times h_{i,j} \tag{1}$$

where k refers to the ant whose pheromone we are evaluating, t refers to the current iteration, $f_{i,j}(t)$ is the amount of pheromone between state i and state j, and $h_{i,j}$ is the score increment between state i and state j. This score is measured according to the number of matches between the output produced by the current circuit and the output desired according to the truth table given by the user. This score increment ($h_{i,j}$) is analogous to the distance between nodes used in the TSP. No normalization takes place at this stage, because in a further step of the algorithm a proportional selection process is performed.

The amount of pheromone is updated each time an agent builds an entire path (i.e., once the whole circuit is built). Before this update, the pheromone evaporation is simulated using the following formula:

$$f_{i,j}(t+1) = (1-\alpha) \times f_{i,j}(t) + \sum_{k=1}^{m} f_{i,j}^{k}(t) \tag{2}$$

where $0 < \alpha < 1$ ($\alpha = 0.5$ was used in all the experiments reported in this paper) is the trail persistence and its use avoids the unlimited accumulation of pheromone in any path, m refers to the number of agents (or ants) and $\sum_{k=1}^{m} f_{i,j}^{k}(t)$ corresponds to the total amount of pheromone deposited by all the ants that went through states (i, j). Furthermore, the pheromone trail is updated according to the circuit built by each agent. The pheromone of the gates of the first row of each column is increased using the following criteria:

1) If the circuit is not feasible (i.e., if not all of its outputs match the truth table), then:

$$f_{i,j}^{k} = f_{i,j}^{k} + \text{payoff} \tag{3}$$

2) If the circuit is feasible (i.e., all of its outputs match the truth table), then:

$$f_{i,j}^{k} = f_{i,j}^{k} + (\text{payoff} \times 2) \tag{4}$$

3) The best individual (from all the agents considered) gets a larger reward:

$$f_{i,j}^{k} = f_{i,j}^{k} + (\text{payoff} \times 3) \tag{5}$$

The value of "payoff" is given by the number of matches produced between the output generated by the circuit built by the agent and the truth table given by the user (a bonus is added for each WIRE found in the solution only in those cases in which the circuit is feasible—i.e., it matches all the outputs given in the truth table).

To build a circuit, we start by placing a gate (randomly chosen) at a certain matrix position and we fill up the rest of the matrix using WIREs. This tries to compute the effect produced by a gate used at a certain position (we compute the score corresponding to any partially built circuit). The distance is computed by subtracting the hits obtained at the current level (with respect to the truth table) minus the hits obtained up to the previous level (or column). When we are at the first level, we assume a value of zero for the previous level.

Table 1. Truth table for the circuit of the first example.

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

3 Results

We used several examples taken from the literature to test our AS implementation. Our results were compared to those obtained by two human designers and a genetic algorithm with binary representation (see [2] for details).

3.1 Example 1

Table 2. Comparison of results between the AS, a binary GA (BGA), and two human designers for the circuit of the first example.

BGA	**Human Designer 1**
$F = Z(X+Y) \oplus (XY)$	$F = Z(X \oplus Y) + Y(X \oplus Z)$
4 gates	5 gates
2 ANDs, 1 OR, 1 XOR	2 ANDs, 1 OR, 2 XORs
AS	**Human Designer 2**
$F = (Z \oplus XY)(X+Y)$	$F = X'YZ + X(Y \oplus Z)$
4 gates	6 gates
2 ANDs, 1 OR, 1 XOR	3 ANDs, 1 OR, 1 XOR, 1 NOT

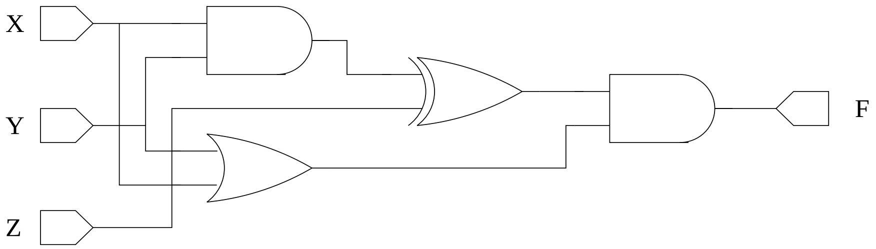

Fig. 3. Circuit produced by the AS for the first example.

Our first example has 3 inputs and one output, as shown in Table 1. In this case, the matrix used was of size 5×5, and the length of each string representing a circuit was 75. Since 5 gates were allowed in each matrix position, then the size of the intrinsic search space (i.e., the maximum size allowed as a consequence of the representation used) for this problem is 5^l, where l refers to the length required to represent a circuit ($l = 75$ in our case). Thefore, the size of the intrinsic search space is $5^{75} \approx 2.6 \times 10^{52}$. The graphical representation of the circuit produced by the AS is shown in Fig. 3. The AS found this solution after 13 iterations using 30 ants.

The comparison of the results produced by the AS, a genetic algorithm with binary representation (BGA) and two human designers are shown in Table 2. As we can see, the AS found a solution with the same number of gates as the BGA. In this case, human designer 1 used Karnaugh Maps plus Boolean algebra identities to simplify the circuit, whereas human designer 2 used the Quine-McCluskey Procedure.

The parameters used by the BGA were the following: crossover rate = 0.5, mutation rate = 0.0022, population size = 900, maximum number of generations

= 400. The solution reported for the BGA in Table 2 was found in generation 197. The matrix used by the BGA was of size 5 × 5.

3.2 Example 2

Table 3. Truth table for the circuit of the second example.

Z	W	X	Y	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

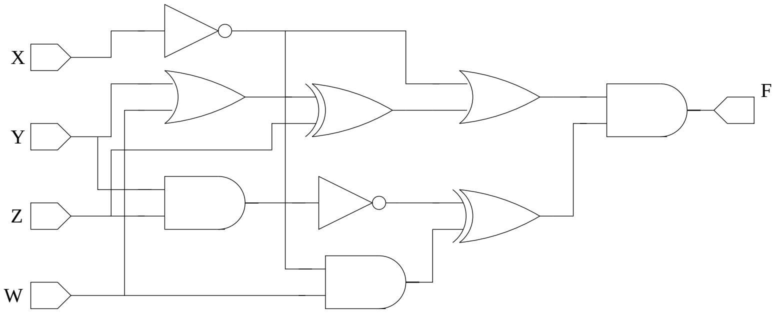

Fig. 4. Circuit produced by the AS for the second example.

Our second example has 4 inputs and one output, as shown in Table 3. In this case, the matrix used was of size 10 × 8. The size of the intrinsic search space for this problem is then 5^{240}. The graphical representation of the circuit

Table 4. Comparison of results between the AS, a binary GA (BGA), a human designer and Sasao's approach for the circuit of the second example

BGA
$F = (WYX' \oplus ((W+Y) \oplus Z \oplus (X+Y+Z)))'$
10 gates
2 ANDs, 3 ORs, 3 XORs, 2 NOTs
Human Designer 1
$F = ((Z'X) \oplus (Y'W')) + ((X'Y)(Z \oplus W'))$
11 gates
4 ANDs, 1 OR, 2 XORs, 4 NOTs
AS
$F = (((W+Y) \oplus Z) + X')((YZ)' \oplus (X'W))$
9 gates
3 ANDs, 2 ORs, 2 XORs, 2 NOTs
Sasao
$F = X' \oplus Y'W' \oplus XY'Z' \oplus X'Y'W$
12 gates
3 XORs, 5 ANDs, 4 NOTs

produced by the AS is shown in Fig. 4. The AS found this solution after 15 iterations using 30 ants.

The comparison of the results produced by the AS, a genetic algorithm with binary representation (BGA), a human designer (using Karnaugh maps), and Sasao's approach [7] are shown in Table 4. In this case, the AS found a solution slightly better than the BGA. Sasao has used this circuit to illustrate his circuit simplification technique based on the use of ANDs & XORs. His solution uses, however, more gates than the circuit produced by our approach.

The parameters used by the BGA for this example were the following: crossover rate = 0.5, mutation rate = 0.0022, population size = 2000, maximum number of generations = 400. Convergence to the solution shown for the BGA in Table 4 was achieved in generation 328. The matrix used by the BGA was of size 5×5.

3.3 Example 3

Our third example has 4 inputs and one output, as shown in Table 5. In this case, the matrix used was of size 15×15. The size of the intrinsic search space for this problem is then 5^{675}. The graphical representation of the circuit produced by the AS is shown in Fig. 5. The AS found this solution after 8 iterations using 30 ants.

The comparison of the results produced by the AS, a genetic algorithm with binary representation (BGA), and 2 human designers (the first using Karnaugh maps and the second using the Quine-McCluskey procedure), are shown in Table 6. In this example, the BGA found a solution slightly better than the AS.

Table 5. Truth table for the circuit of the third example.

A	B	C	D	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Table 6. Comparison of results between the AS, a binary GA (BGA), and two human designers for the circuit of the third example.

BGA
$F = ((A \oplus B) \oplus AD) + (C + (A \oplus D))'$
7 gates
1 AND, 2 ORs, 3 XORs, 1 NOT
Human Designer 1
$F = ((A \oplus B) \oplus ((AD)(B + C))) + ((A + C) + D)'$
9 gates
2 ANDs, 4 ORs, 2 XORs, 1 NOT
AS
$F = ((B \oplus D) \oplus (A + D)) \oplus ((B + C) + (A \oplus D))'$
8 gates
3 ORs, 4 XORs, 1 NOT
Human Designer 2
$F = A'B + A(B'D' + C'D)$
10 gates
4 ANDs, 2 ORs, 4 NOTs

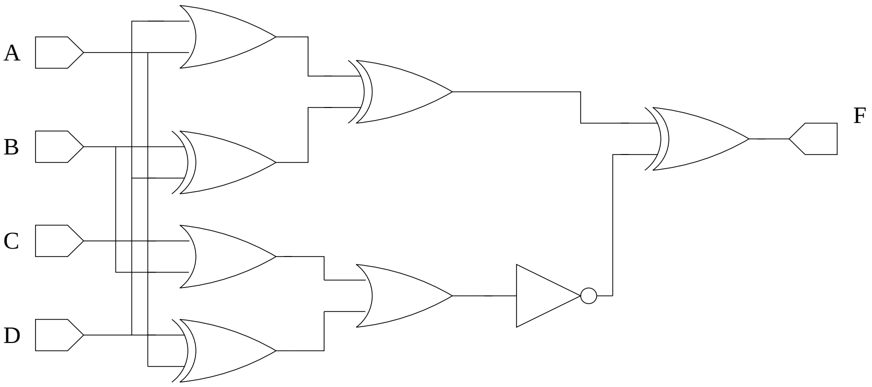

Fig. 5. Circuit produced by the AS for the third example.

The parameters used by the BGA for this example were the following: crossover rate = 0.5, mutation rate = 0.0022, population size = 2600, maximum number of generations = 400. Convergence to the solution shown for the BGA in Table 6 was achieved in generation 124. The matrix used by the BGA was of size 5 × 5.

4 Conclusions and Future Work

In this paper we have presented an approach to use the ant colony system to optimize combinational logic circuits (at the gate level). The proposed approach was described and a few examples of its use were presented. Results compared fairly well with those produced with a BGA (a GA with binary representation) and are better than those obtained using Karnaugh maps and the Quine-McCluskey Procedure.

Some of the future research paths that we want to explore are the parallelization of the algorithm to improve its performance (each agent can operate independently from the others until they finish a path and then they have to be merged to update the pheromone trails) and the hybridization with other algorithms (e.g., local search).

We also want to experiment with other metaheuristics such as tabu search to scale up the use of AS to larger circuits (our current implementation is limited to circuits of only one output) without a significant performance degradation. Finally, we are also interested in exploring alternative (and more powerful) representations of a Boolean expression in an attempt to overcome the inherent limitations of the matrix representation currently used to solve real-world circuits in a reasonable amount of time and without the need of excessive computer power.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments that helped them improve this paper.

We also thank Carlos E. Mariano for supplying the initial code for the AS on which this work is based. The first author acknowledges support from CONACyT through project No. I-29870 A.

The second and third authors acknowledge support from CONACyT through a scholarship to pursue graduate studies at the Maestría en Inteligencia Artificial of LANIA and the Universidad Veracruzana.

The last author acknowledges support for this work to DoD EPSCoR and the Board of Regents of the State of Louisiana under grant F49620-98-1-0351.

References

1. Carlos A. Coello, Alan D. Christiansen, and Arturo Hernández Aguirre. Automated Design of Combinational Logic Circuits using Genetic Algorithms. In D. G. Smith, N. C. Steele, and R. F. Albrecht, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 335–338. Springer-Verlag, University of East Anglia, England, April 1997.
2. Carlos A. Coello, Alan D. Christiansen, and Arturo Hernández Aguirre. Use of Evolutionary Techniques to Automate the Design of Combinational Circuits. *International Journal of Smart Engineering System Design*, 1999. (accepted for publication).
3. A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In F. J. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life*, pages 134–142. MIT Press, Cambridge, MA, 1992.
4. G. Di Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.
5. M. Dorigo and G. Di Caro. The Ant Colony Optimization Meta-Heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.
6. M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.
7. Tsutomu Sasao, editor. *Logic Synthesis and Optimization*. Kluwer Academic Press, 1993.