Automated Design of Combinational Logic Circuits using the Ant System

Carlos A. Coello Coello*
CINVESTAV-IPN
Depto. de Ingeniería Eléctrica
Sección de Computación
Av. Instituto Politécnico Nacional No. 2508
Col. San Pedro Zacatenco
México, D. F. 07300, MEXICO
ccoello@cs.cinvestav.mx

Rosa Laura Zavala Gutiérrez & Benito Mendoza García
MIA, LANIA-UV
Sebastián Camacho 5
Xalapa, Veracruz 91090, MEXICO
{rzavala,bmendoza}@mia.uv.mx

Arturo Hernández Aguirre
EECS Department
Tulane University
New Orleans, LA 70118, USA
hernanda@eecs.tulane.edu

March 20, 2001

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.

Keywords: circuit design, ant colony system, evolvable hardware, circuit optimization.

*Most of this work was performed while the first author was at the Laboratorio Nacional de Informática Avanzada (LANIA), in Xalapa, Veracruz, México.

1 Introduction

The design of digital circuits is a difficult task that is normally associated with certain human attributes (i.e., creativity). Its automation is therefore a challenging problem.

There are several standard graphical design aids for combinational circuit synthesis (e.g., Karnaugh Maps and the Quine-McCluskey method). Although some of these techniques are fairly limited, others can handle truth tables with hundreds of inputs. In comparison, heuristics such as the genetic algorithm (GA) are normally restricted to relatively small truth tables [19]. Such restrictions also apply to other heuristics such as the ant system (which is the subject of this paper).

But evolutionary design has triggered the study of other aspects of design that have been normally disregarded, for example, emergent design patterns [19, 3]. It is therefore clear that when using heuristics, we aim not only to synthesize a circuit (using a certain metric), but also to produce novel designs that do not correspond to the solutions that a human designer would typically produce [19, 17, 3]. Additionally, the existence of techniques (such as the one discussed in this paper) that allow us to produce compact circuits (i.e., with few gates) for relatively small truth tables could be of great use for function-level design (where these compact circuits would be used as building blocks for designing more complex circuits). Such a divide-and-conquer approach to circuit design has been suggested in the past [20, 19] and we believe that it constitutes a viable alternative to deal with scalability issues related to evolvable hardware (i.e., circuit design using heuristics).

The remainder of this paper is organized as follows: first, we provide a short description of the ant system. Then, we describe some of the previous related work on automated combinational circuit design. After that, we introduce our approach, giving several examples of its performance. Results are compared against those produced by a GA and a human designer. Then, we present a short discussion of our results, our conclusions and some of the possible paths of future research.

2 The Ant System

The ant system (AS) is a meta-heuristic inspired by colonies of real ants, which deposit a chemical substance on the ground called *pheromone* [8]. This substance influences the behavior of the ants: they will tend to take those paths where there is a larger amount of pheromone. Pheromone trails can be seen as an indirect communication mechanism among ants. From the computer science perspective, 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. Figure 1 shows graphically an example of the typical behavior of a colony of real ants. When the ants leave initially the nest, (1) they follow random patterns. (2) Over time, they start following a common path. (3,4)

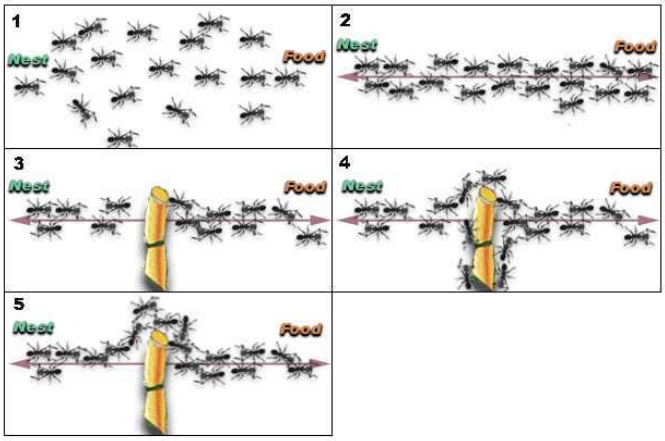

Figure 1: Behavior of a colony of real ants.

When faced with an obstacle, some choose to go around it through the left side of the obstacle and others avoid it going through the right. (5) Over time, the whole colony will follow a common path (the shortest way) due to the pheromone trials.

There are three main ideas from colonies of real ants that have been adopted in the AS:

1. The indirect communication through pheromone trials.
2. Shortest paths tend to have a higher growth rate of pheromone values.
3. Ants have a higher preference (with a certain probability) for paths that have a higher amount of pheromone.

Additionally, the AS has certain capabilities nonexistent in colonies of real ants. For example:

1. Each ant is capable of estimating how far it is from a certain state.
2. Ants have information about the environment and use it to make decisions. Therefore, their behavior is not only adaptive, but also exhaustive.
3. Ants have memory, since this is necessary to make sure that only feasible solutions are generated at each step of the algorithm.

The AS was originally proposed for the traveling salesman problem (TSP), and according to Dorigo [9], 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.

In fact, the main contribution of this paper is precisely our proposal regarding how to reformulate the circuit optimization problem so as to allow the use of the AS. We will use several examples to compare the solutions generated by our approach against those produced by a human designer and by a GA with binary representation previously developed by us for this problem [4].

3 Related Work

We could not find any previous work on the design of circuits using the ant system. Therefore, we will briefly discuss some related work using evolutionary techniques (genetic algorithms and genetic programming).

In the contemporary literature, the attempt to use evolutionary-based techniques to design electrical circuits has been called "evolvable hardware" [15, 6]. Within evolvable hardware, we can distinguish three types of evolutionary processes [13]: extrinsic evolution (we use software models of the circuit and evaluations are performed with a simulator), intrinsic evolution (we use a physical model of the circuit and evaluations are performed with test equipment), and mixtrinsic evolution (a mixture of the two previous types). Our work uses extrinsic evolution.

There are also several levels at which evolution can be performed. In this work, we are only considering the lowest, which is called gate-level evolvable hardware, because the primitives used to design circuits are gates such as AND, OR and NOT. It is known that gate-level evolution is only suitable for small circuits [13]. However, our belief is that if these small circuits can be highly optimized, they will be more useful at the following level of evolvable hardware (the so-called "function-level"), at which these small circuits will be used as primitives to design more complex circuits. In fact, similar mechanisms as those adopted in this work can be used for function-level evolvable hardware, although such design is beyond the scope of this paper.

Despite the limitations of gate-level design, several researchers have worked in this area [16, 18, 11, 13]. Furthermore, besides the normal use of gate-level design (e.g., two-bit adders, two- and three-bit multipliers, decoders, etc.), there have been a few more complex applications reported in the specialized literature (see for example [12, 14]).

Our goal in this paper is to show the feasibility of using the ant system for gate-level design of circuits. We will describe how to adapt the ant system algorithm to design combinational circuits, and we will show how the resulting approach is competitive with a traditional GA in terms of performance and quality of the solutions produced. The approach described in this paper is an extension of an approach previously reported [5] in which the main limitation was the fact that only circuits with one output could be designed by the system. Our current approach also shows a significant improvement in terms of performance with respect to our previous version.

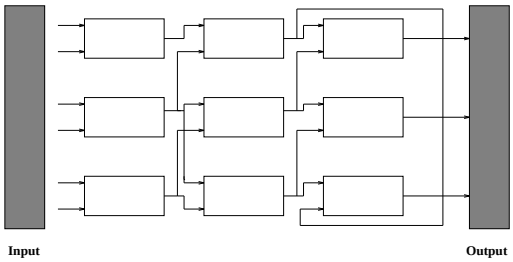

Figure 2: Matrix used to represent a circuit to be processed by an agent (i.e., an ant). Each gate gets its inputs from either of the gates at the previous column.

4 Description of the Approach

In this section, we will describe the way in which the circuit design problem had to be reformulated in order to be able to use the AS to solve it. The main problem that we faced was how to make an analogy (as much as possible) between circuit design and the TSP. The main issues are: the representation to be adopted, the notion of state in that representation, the way in which a path would be built, and the way of updating the trails of each ant. Each of these issues will be discussed in this section

4.1 Representation

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 Figure 2. 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 Figure 2). 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 (five types of gates were considered in our work: AND, NOT, OR, XOR and WIRE[1]) that receives its 2 inputs from any gate at the previous column as shown in Figure 2. We have used this representation before with a GA [2, 4].

A chromosomic string encodes the matrix shown in Figure 2 by using triplets in which the 2 first elements refer to each of the inputs used, and the third is the corresponding gate as shown in Figure 3 (only 2-input gates were used in this work).

[1]WIRE basically indicates a null operation, or in other words, the absence of gate.

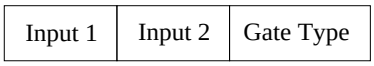

Figure 3: Encoding used for each of the matrix elements that represent a circuit.

4.2 Building a path

The path of an ant in our case is a full circuit. In other words, each ant traverses a path and, in the process, it builds a circuit. In the TSP, the ants also traverse a path and try to find the shortest way to the goal. In our case, "shortest" relates to "less gates". However, in the TSP, any permutation is a valid solution, whereas in our case, an arbitrary string encodes a circuit that may or may not be feasible. We only try to minimize the number of gates of feasible circuits.

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, and other representations are currently a matter of further research). The matrix containing the solution to the problem is built in a column-order fashion as indicated next.

Each state is, in our case, a column of the matrix, which is composed of several elements. A certain state is selected element by element (gate by gate). Each of these column elements is called a substate. A substate is a triplet in which the first two 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) as shown in Figure 3. 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). 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).

The distance (between cities or states), which we denote by h, is measured in our case as the increment or decrement in the fitness value of the circuit when we move from one level to the next. By level, we refer to a column in the matrix. Since our algorithm builds the circuit progressively (starting from the leftmost column), as we move to the right, levels increase and fitness values change. Fitness in this domain is measured according to the amount of hits achieved (i.e., matches between the outputs of the circuit and the outputs defined in the truth table). Feasible circuits get an extra increase in their fitness measured as the amount of WIREs that they contain. This allows us to perform a fair comparison between feasible and infeasible designs (i.e., feasible designs always get a higher reward than infeasible designs).

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 [7]).

When we move from one substate to another in the path, a value is assigned to all the substates that have not been visited yet and the next substate (i.e., the next triplet) is randomly selected using a certain selection factor p^k. 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:

$$p^k_{i,j,l} = f_{j,l} \times h_{i,j,l} \tag{1}$$

where k refers to the ant whose pheromone we are evaluating (the ant that is building the path), $f_{j,l}$ is the amount of pheromone at state j at row l (this value is initialized to zero), and $h_{i,j,l}$ is the score increment between substate i and substate j for row l (each row is associated with an output in the truth table). 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. The value of $h_{i,j,l}$ is given by the amount of hits that the partially-built circuit produces so far with respect to the l output of the truth table provided by the user. This value is therefore a score increment analogous to the distance between nodes used in the TSP.

Once every combination has been assigned a selection factor, we choose one of them. At this point, we apply roulette-wheel selection[2]. We do this for every substate that belongs to one of the rows representing an output of the circuit. The other substates are randomly chosen.

The previous process is repeated until we finish a path (i.e., until we reach the last state of the circuit, or the last column of the matrix).

4.3 Updating the trails

The amount of pheromone is updated each time an agent builds an entire path (i.e., once the whole circuit is built). This is done in two steps:

1. First, we simulate the evaporation of the pheromone trails in all substates, such as they occur with real ants (over time). For the simulation, we adopt the following formula:

$$f_{i,l} = (1 - \alpha) \times f_{i,l} \tag{2}$$

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, and $f_{i,l}$ is the amount of pheromone at state i at row l.

[2]Roulette-wheel selection belongs to the so-called "proportional selection methods". In these methods, the probability of selecting an individual is proportional to its fitness contribution (with respect to the total fitness of the population) [10]. Normally, the probability of selecting option i is given by $f_i/(\sum_{j=1}^{m} f_j)$, where m is the amount of options under consideration and fj is the fitness of individual j.

2. Then, we deposit the pheromone in the substates through which the ants passed, using the following formula:

$$f_{i,l} = f_{i,l} + \sum_{k=1}^{m} f_{i,l}^{k} \tag{3}$$

where m refers to the number of agents (or ants), $f_{i,l}^{k}$ corresponds to the amount of pheromone deposited by ant k at state i at row l. This value is obtained in the following way:

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

$$f_{i,l}^{k} = \text{payoff} \tag{4}$$

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

$$f_{i,l}^{k} = \text{payoff} \times 2 \tag{5}$$

- If it is the circuit with the highest fitness (i.e., the best path found):

$$f_{i,l}^{k} = \text{payoff} \times 3 \tag{6}$$

- If the ant k did not pass through substate i of row l:

$$f_{i,l}^{k} = 0 \tag{7}$$

The value of payoff is given by the following expression:

$$\text{payoff} = hits + ((Cols \times Rows) - TotCirc) \tag{8}$$

where: $hits$ is the number of matches produced between the outputs generated by the circuit produced by the AS and the truth table given by the user; $Cols$ is the amount of columns in the matrix; $Rows$ is the number of rows in the matrix, and $TotCirc$ is the amount of gates used by the circuit generated by the AS.

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.

The pseudo-code of our approach is the following:

```
Program Ant System for Circuit Design
  Open input and output files
  Initialize random numbers seed
  Read input data
  For i = 1 to Max_Cic
      // Loop until reaching maximum number of iterations
          For j = 1 to popsize
            // For each ant do
            Build_Solution(j)
            Evaluate_Solution(j)
          End
          Update_Trails
          Print Report
  End
  Print Global Best
  Close Files
End
```

5 Comparison of Results

We used several examples taken from the literature to test our AS implementation. Our results were compared to those obtained by a human designer (using Karnaugh maps plus simplification using Boolean rules) and by a genetic algorithm using binary representation (BGA). In all the examples presented, 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 all of the examples 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 $5^{75} \approx 2.6 \times 10^{52}$. Also, the parameters of the BGA were chosen so that they approximated the total number of fitness function evaluations required by the AS[3]. For each of the following examples, we performed 20 runs with each technique.

The experiments described next were performed on a PC with a Pentium III processor (running at 550 Mhz), with 128 Mbytes in RAM and a 13 Gbytes hard disk. The code was implemented using Borland C++ Builder 4. To allow a fair comparison, the binary genetic algorithm was tested under Red Hat Linux (version 7) and ran on the same computer. We prefer to use the amount of fitness function evaluations required by each approach to perform a comparison of performance, since different architectures and software platforms can provide different running times for the same program. However, for completeness, we will mention the CPU required for a single run in each of the following examples

[3]In this work, the term "fitness function evaluation" refers to a unit used to compare the performance of our algorithm against others. In terms of computational effort, a fitness function evaluation is the amount of time required to evaluate a solution (i.e., a circuit).

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

Table 2: Comparison of the Boolean expressions produced by the AS, a GA with binary representation (BGA), and a human designer for the circuit of the first example.

Human Designer
$F = Z(X \oplus Y) + Y(X \oplus Z)$
5 gates
2 ANDs, 1 OR, 2 XORs
BGA
$F = (XZ)' \oplus ((X + Z)Y)$
6 gates
2 ANDs, 1 OR, 1 XOR, 2 NOTs
Ant System
$F = (Z \oplus XY)(X + Y)$
4 gates
2 ANDs, 1 OR, 1 XOR

(and using the hardware and software platforms previously mentioned).

5.1 Example 1

Our first example has 3 inputs and 1 output as shown in Table 1.

The parameters used by the Ant System and the BGA are shown in Table 3 (α is the evaporation factor). In this case, the BGA performed 25,000 fitness function evaluations per run, and the AS performed 20,600 fitness function evaluations per run.

The summary of the results produced is shown in Table 4. The AS was able to find a solution with a fitness of 29 (4 gates) 85% of the time, and in all cases it converged to a feasible solution. The graphical representation of this circuit is shown in Figure 4.

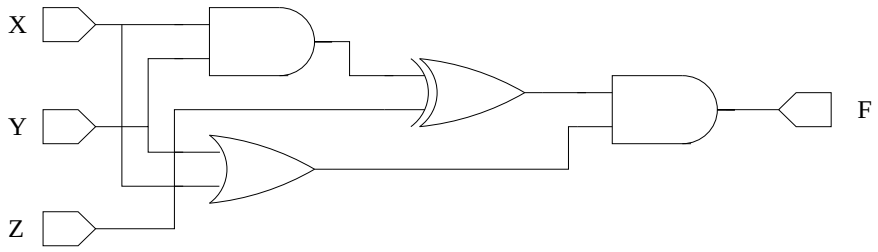

Figure 4: Circuit produced by the AS for the first example.

Table 3: Parameters used by the AS and the BGA for the first example.

AS		**BGA**	
No. of ants	10	Pop. size	100
Max. iters.	10	Max. gen.	250
α	0.5	Cross. rate	0.5
		Mut. rate	0.5/L
		Chrom. length	225

The best solution that the BGA could find had a fitness of 26 (this circuit really had only 6 gates, but the BGA could not eliminate a gate that did not have any impact on the solution) and it appeared only once in the 20 runs performed. For 30% of the runs, the BGA converged to an infeasible solution.

The comparison of the Boolean expressions produced by the AS, a genetic algorithm with binary representation (BGA), and a human designer are shown in Table 2. The solution produced by the AS is better (i.e., it uses less gates) than those produced by the human designer and the BGA.

Table 4: Summary of results produced by the Ant System (AS) and a Genetic Algorithm with binary representation (BGA) for the first example.

	AS	**BGA**
Best fitness	29	26
Average	28.85	18.25
Std. dev.	0.366347549	7.663138013
Mode	29	7
Lowest fitness	28	7
CPU time	0.1 secs.	9 secs.

Table 5: Truth table for the circuit of the second example.

A	**B**	**W**	**X**	**Y**	**Z**
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Table 6: Comparison of the Boolean expressions produced by the AS, a GA with binary representation (BGA), and a human designer for the circuit of the second example.

Human Designer
$X = A'B'$, $Y = A'B$, $Z = AB'$, $W = AB$
6 gates
4 ANDs, 2 NOTs
BGA
$W = (AB)A$, $X = A \oplus (AB)$, $Y = ((A \oplus B) + A) \oplus A$, $Z = ((A \oplus B) + A)'$
7 gates
3 XORs, 1 OR, 2 ANDs, 1 NOT
Ant System
$X = AB \oplus A$, $Y = BA'$, $Z = A' \oplus BA'$, $W = AB$
5 gates
2 XORs, 2 ANDs, 1 NOT

Table 7: Parameters used by the AS and the BGA for the second and third examples.

AS		BGA	
No. of ants	30	Pop. size	200
Max. iters.	30	Max. gen.	1000
α	0.5	Cross. rate	0.5
		Mut. rate	0.5/L
		Chrom. length	225

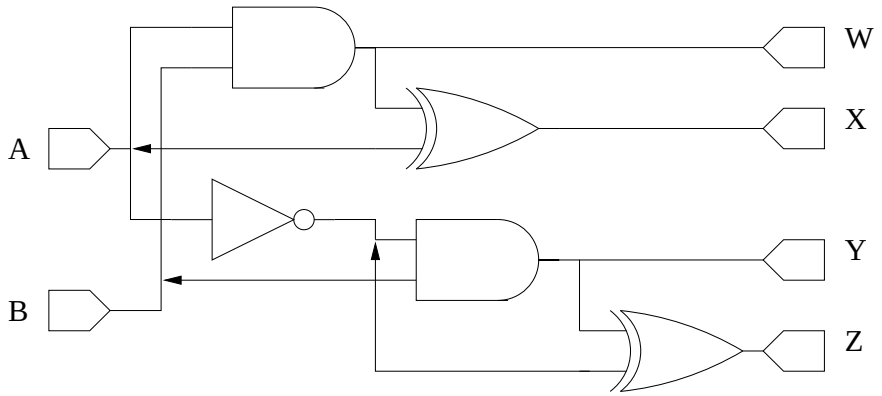

Figure 5: Circuit produced by the AS for the second example.

5.2 Example 2

Our second example has 2 inputs and 4 outputs (it is a decoder 2-4) as shown in Table 5.

The parameters used by the Ant System and the BGA are shown in Table 7. In this case, the BGA performed 200,000 fitness function evaluations per run, and the AS performed 185,400 fitness function evaluations per run.

The summary of the results produced is shown in Table 8. The AS was able to find a solution with a fitness of 36 (i.e., a circuit with 5 gates) in all the runs performed. The graphical representation of this circuit is shown in Figure 5.

The best solution that the BGA could find had a fitness of 34 (i.e., a feasible circuit with 7 gates). In 10% of the runs, the BGA converged to an infeasible solution.

The comparison of the Boolean expressions produced by the AS, a genetic algorithm with binary representation (BGA), and a human designer are shown in Table 6. It can be clearly seen that the AS produced better solutions than both the human designer and the BGA for this example. However, note in Table 6, that some of the Boolean expressions generated by the BGA can be easily simplified (e.g., $W = (AB)A = AB$). Nevertheless, we were interested in

Table 8: Summary of results produced by the Ant System (AS) and a Genetic Algorithm with binary representation (BGA) for the second example.

	AS	**BGA**
Best fitness	36	34
Average	36	28.45
Std. dev.	0.0	5.072889761
Mode	36	30
Lowest fitness	36	15
CPU time	13 secs.	30 secs.

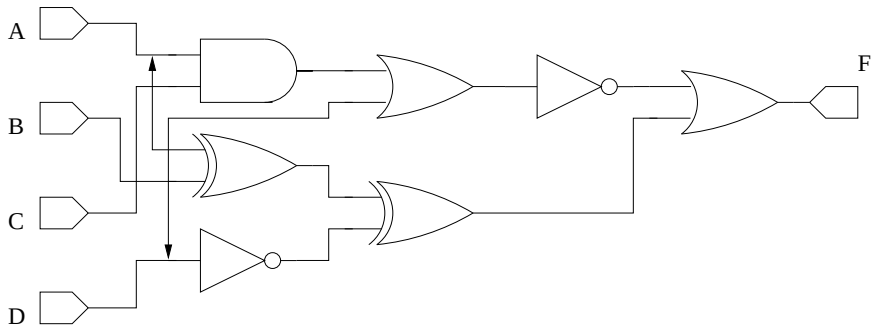

Figure 6: Circuit produced by the AS for the third example.

comparing the solutions generated by the AS and the BGA without any extra human intervention.

5.3 Example 3

Our third example has 4 inputs and 1 output, as shown in Table 9. The parameters used by the Ant System and the BGA are the same shown in Table 7. As in the previous example, the BGA performed 200,000 fitness function evaluations per run, and the AS performed 185,400 fitness function evaluations per run.

The summary of the results produced is shown in Table 11. The best solution that the AS could find had a fitness of 34 (i.e., a circuit with 7 gates). The graphical representation of this circuit is shown in Figure 6. In all cases, the AS converged to a feasible circuit and 25% of the time a fitness value of 34 was achieved.

The best solution that the BGA could find had a fitness of 34 (i.e., a feasible circuit with 7 gates), but it appeared only once in the 20 runs performed. The BGA converged to an infeasible solution 60% of the time.

The comparison of the Boolean expressions produced by the AS, a genetic algorithm with binary representation (BGA), and a human designer are shown

Table 9: 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	1
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	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Table 10: Comparison of the Boolean expressions produced by the AS, a GA with binary representation (BGA), and a human designer for the circuit of the third example.

Human Designer
$F = (D' + (A \oplus B))((AC)' + (B \oplus D))$
8 gates
2 XORs, 2 ANDs, 2 ORs, 2 NOTs
BGA
$F = (AC + (B + D))' + (AD \oplus B)$
7 gates
2 ANDs, 3 ORs, 1 XOR, 1 NOT
Ant System
$F = (AC + D)' + (A \oplus B) \oplus D'$
7 gates
2 XORs, 1 AND, 2 ORs, 2 NOTs

Table 11: Summary of results produced by the Ant System (AS) and a Genetic Algorithm with binary representation (BGA) for the third example.

	AS	**BGA**
Best fitness	34	34
Average	33.15	21.3
Std. dev.	0.587142949	8.398621441
Mode	33	15
Lowest fitness	32	13
CPU time	13 secs.	30 secs.

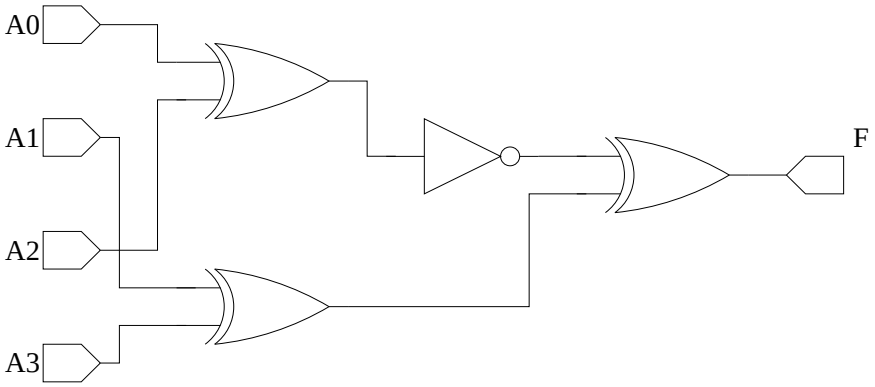

Figure 7: Circuit produced by the AS for the fourth example.

in Table 10.

5.4 Example 4

Our fourth example is an even 4-parity problem. The circuit has 4 inputs and 1 output, as shown in Table 12.

The parameters used by the Ant System and the BGA are shown in Table 13. In this case, the BGA performed 100,000 fitness function evaluations per run, and the AS performed 82,400 fitness function evaluations per run.

The summary of the results produced is shown in Table 15. The AS was able to find a solution with a fitness of 37 (i.e., a feasible circuit with 4 gates) in all the runs performed. The graphical representation of this circuit is shown in Figure 7.

The best solution that the BGA could find had a fitness of 37 (i.e., a feasible circuit with 4 gates), but it appeared only four times in the 20 runs performed (i.e., 20% of the time). For 20% of the runs, the BGA converged to an infeasible solution.

The comparison of the Boolean expressions produced by the AS, a genetic

Table 12: Truth table for the circuit of the fourth example.

$\mathbf{A}_0$	$\mathbf{A}_1$	$\mathbf{A}_2$	$\mathbf{A}_3$	**F**
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Table 13: Parameters used by the AS and the BGA for the fourth example.

AS		**BGA**	
No. of ants	20	Pop. size	100
Max. iters.	20	Max. gen.	1000
α	0.5	Cross. rate	0.5
		Mut. rate	0.5/L
		Chrom. length	225

Table 14: Comparison of the Boolean expressions produced by the AS, a GA with binary representation (BGA), and a human designer for the circuit of the fourth example (an even 4-parity problem).

Human Designer
$F = ((A_0 \oplus A_1)' \oplus (A_2 \oplus A_3)')'$
6 gates
3 XORs, 3 NOTs
BGA
$F = ((A_1 \oplus A_2) \oplus (A_0 \oplus A_3))'$
4 gates
3 XORs, 1 NOT
Ant System
$F = (A_0 \oplus A_2)' \oplus (A_1 \oplus A_3)$
4 gates
3 XORs, 1 NOT

Table 15: Summary of results produced by the Ant System (AS) and a Genetic Algorithm with binary representation (BGA) for the fourth example.

	AS	**BGA**
Best fitness	37	37
Average	37	29.90
Std. dev.	0.0	9.181789758
Mode	37	15
Lowest fitness	37	15
CPU time	6 secs.	68 secs.

Table 16: Truth table for the 2-bit multiplier of the fifth example.

A_1	A_0	B_1	B_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

algorithm with binary representation (BGA), and a human designer are shown in Table 14. The classical human solution to this problem has 3 XNORs. Since we did not use XNORs in our representation, we count each XNOR as 2 gates (1 XOR and 1 NOT). Therefore, the solution produced by a human is considered to have 6 gates. Note how the AS and the BGA found a rearrangement of inputs that allows us to save two gates (the two solutions are equivalent, but not identical).

5.5 Example 5

Our fifth example is the 2-bit multiplier (4 inputs and 4 outputs) whose truth table is shown in Table 16.

The parameters used by the Ant System and the BGA are shown in Table 18. In this case, the BGA performed 800,000 fitness function evaluations per run, and the AS performed 725,400 fitness function evaluations per run.

The summary of the results produced is shown in Table 19. The best solution that the AS could find had a fitness of 82 (i.e., a feasible circuit with 7 gates) and is graphically depicted in Figure 8. In all cases, the AS converged to a feasible circuit and 40% of the time a fitness value of 82 was achieved.

The best solution that the BGA could find had a fitness of 80 (i.e., a feasible circuit with 9 gates), and it appeared only once in the 20 runs performed. For 55% of the runs, the BGA converged to an infeasible solution.

The comparison of the Boolean expressions produced by the AS, a genetic

Table 17: Comparison of the Boolean expressions produced by the AS, a GA with binary representation (BGA), and a human designer for the circuit of the fifth example (a 2-bit multiplier).

Human Designer
$C_0 = A_0B_0$
$C_1 = A_0B_1 \oplus A_1B_0$
$C_2 = A_1B_1(A_0B_0)'$
$C_3 = A_1A_0B_1B_0$
8 gates
6 ANDs, 1 XORs, 1 NOT
BGA
$C_0 = ((A_0B_0)')'$
$C_1 = A_0B_1 \oplus A_1B_0$
$C_2 = A_1B_1(A_0B_0)'$
$C_3 = A_1A_0B_1B_0$
9 gates
1 XOR, 6 ANDs, 2 NOTs
Ant System
$C_0 = A_0B_0$
$C_1 = A_1B_0 \oplus A_0B_1$
$C_2 = A_1A_0B_1B_0 \oplus A_1B_1$
$C_3 = A_1A_0B_1B_0$
7 gates
2 XORs, 5 ANDs

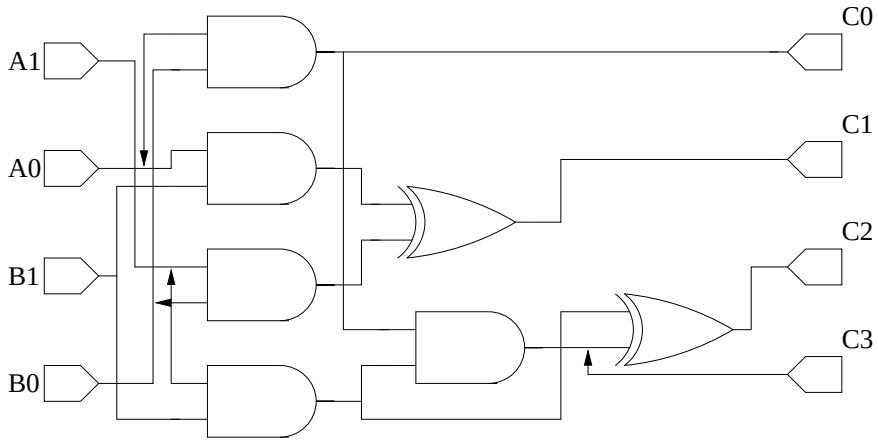

Figure 8: Two-bit multiplier produced by the AS for the fifth example.

Table 18: Parameters used by the AS and the BGA for the fourth example.

AS		**BGA**	
No. of ants	30	Pop. size	800
Max. iters.	30	Max. gen.	1000
α	0.5	Cross. rate	0.5
		Mut. rate	0.5/L
		Chrom. length	225

Table 19: Summary of results produced by the Ant System (AS) and a Genetic Algorithm with binary representation (BGA) for the fifth example.

	AS	**BGA**
Best fitness	82	80
Average	81.4	68.25
Std. dev.	0.50262469	7.731514456
Mode	81	63
Lowest fitness	81	62
CPU time	55 secs.	253 secs.

algorithm with binary representation (BGA), and a human designer are shown in Table 17. The solution produced by the AS is better (i.e., it uses less gates) than those produced by the human designer and the BGA. In fact, these last two solutions are really the same, although the BGA was not able to eliminate a double NOT in the Boolean expression.

6 Discussion of Results

The results presented in the previous section indicate that the AS is very suitable for combinational circuit design at the gate level. In all cases, our approach produced circuits that were, in the worst case, equivalent to those generated by a BGA and better than those produced by a human designer. In fact, in most cases, the AS was able to improve the solutions produced by the BGA for an equivalent amount of fitness function evaluations. Also, the lower standard deviation obtained from the runs of the AS indicate its robustness in this domain (in two examples a standard deviation of zero was achieved).

As in the case of the BGA, the AS tends to use XOR gates to simplify a circuit, and it also tends to degrade (in terms of performance) as we increase the complexity (e.g., the amount of outputs) of a circuit. While it is feasible to use the AS to solve larger circuits than those included in this paper, its performance tends to degrade rapidly as we increase the size of the circuit to be solved.

This problem, however, is also present when using a GA, and it is commonly associated with gate-level design [13]. As indicated at the beginning of this paper, one way to tackle scalability issues of this kind is by using function-level design. However, we believe that the use of techniques such as the AS for gate-level design can produce more compact design units to be used for function-level design and therefore gives relevance to the work reported here.

The AS, like the GA, requires certain parameters to work. To allow a fair comparison, we tried to keep the parameters of the AS fixed for all our experiments. The exception was the first example, for which a lower amount of ants and iterations (ten instead of thirty) made it possible to converge to the best known solution with a low standard deviation (the use of a higher number of ants and iterations significantly reduces the standard deviation). In order to compare the GA against the AS, we used a combination of population size and maximum number of generations such that the total number of fitness function evaluations was approximately equivalent for both techniques (in fact, the AS used always less fitness function evaluations than the GA). We favored lower population sizes for the GA based on the previous experience of other researchers and ourselves [19, 1].

7 Conclusions and Future Work

In this paper we have presented an approach to use the ant system to optimize combinational logic circuits (at the gate level). The proposed approach was described and several 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 by a human designer using Karnaugh maps and Boolean rules for simplification.

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).

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. The first choice that we are considering is to use a tree representation such as in genetic programming [16].

Acknowledgements

The authors thank the anonymous reviewers for their comments which greatly help them to improve the contents of this paper.

The first author acknowledges support from the Consejo Nacional de Ciencia y Tecnología (CONACyT) through project number 32999-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 states that his contribution to this paper describes research done in the Department of Electrical Engineering and Computer Science at Tulane University. He acknowledges partial support for this work through grant NAG5-8570 from NASA/Goddard Space Flight Center, and in part by DoD EPSCoR and the Board of Regents of the State of Louisiana under grant F49620-98-1-0351.

References

[1] Carlos A. Coello Coello, Arturo Hernández Aguirre, and Bill P. Buckles. Evolutionary Multiobjective Design of Combinational Logic Circuits. In Jason Lohn, Adrian Stoica, Didier Keymeulen, and Silvano Colombano, editors, *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware*, pages 161–170. IEEE Computer Society, Los Alamitos, California, July 2000.

[2] Carlos A. Coello 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.

[3] Carlos A. Coello 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*, 2(4):299–314, June 2000.

[4] Carlos A. Coello Coello, Alan D. Christiansen, and Arturo Hernández Aguirre. Towards Automated Evolutionary Design of Combinational Circuits. *Computers and Electrical Engineering. An International Journal*, 27(1):1–28, January 2001.

[5] Carlos A. Coello Coello, Rosa L. Zavala Gutiérrez, Benito Mendoza García, and Arturo Hernández Aguirre. Ant Colony System for the Design of Combinational Logic Circuits. In Julian Miller, Adrian Thompson, Peter Thomson, and Terence C. Fogarty, editors, *Evolvable Systems: From Biology to Hardware*, pages 21–30, Edinburgh, Scotland, April 2000. Springer-Verlag.

[6] Hugo de Garis. Evolvable Hardware: Genetic Programming of a Darwin Machine. In Colin Reeves, R. F. Albrecht, and N. C. Steele, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 117–123, Inssbruck, Austria, 1993. Springer-Verlag.

[7] G. Di Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[8] 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.

[9] 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.

[10] David E. Goldberg and Kalyanmoy Deb. A comparison of selection schemes used in genetic algorithms. In G.J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, San Mateo, California, 1991.

[11] Hitoshi Iba, Masaya Iwata, and Tetsuya Higuchi. Gate-Level Evolvable Hardware: Empirical Study and Application. In Dipankar Dasgupta and Zbigniew Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*, pages 260–275. Springer-Verlag, Berlin, 1997.

[12] I. Kajitani, T. Hoshino, D. Nishikawa, H. Yokoi, S. Nakaya, T. Yamauchi, T. Inuo, N. Kahijara, M. Iwata, D. Keymeulen, and T. Higuchi. A Gate-Level EHW Chip: Implementing GA Operations and Reconfigurable Hardware on A Single LSI. In M. Sipper, D. Mange, and A. Pérez-Uribe, editors, *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES'98)*, volume 1478 of *Lecture Notes in Computer Science*, pages 1–12, Lausanne, Switzerland, 1998. Springer-Verlag.

[13] Tatiana G. Kalganova. *Evolvable Hardware Design of Combinational Logic Circuits*. PhD thesis, Napier University, Edinburgh, Scotland, 2000.

[14] D. Keymeulen, M. Durantez, K. Konaka, J. Kuniyoshi, and T. Higuchi. An Evolutionary Robot Navigation System using a Gate-Level Evolvable Hardware. In *Proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware (ICES'96)*, volume 1259 of *Lecture Notes in Computer Science*, pages 195–209, Tsukuba, Japan, 1996. Springer-Verlag.

[15] Hiroaki Kitano and James A. Hendler, editors. *Massively Parallel Artificial Intelligence*. MIT Press, Cambridge, Massachusetts, 1994.

[16] John R. Koza. *Genetic Programming. On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts, 1992.

[17] J. Miller, T. Kalganova, N. Lipnitskaya, and D. Job. The Genetic Algorithm as a Discovery Engine: Strange Circuits and New Principles. In *Proceedings of the AISB Symposium on Creative Evolutionary Systems (CES'99)*, Edinburgh, UK, 1999.

[18] J. F. Miller, P. Thomson, and T. Fogarty. Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, pages 105–131. Morgan Kaufmann, Chichester, England, 1998.

[19] Julian F. Miller, Dominic Job, and Vesselin K. Vassilev. Principles in the Evolutionary Design of Digital Circuits—Part I. *Genetic Programming and Evolvable Machines*, 1(1/2):7–35, April 2000.

[20] Jim Torresen. A Divide-and-Conquer Approach to Evolvable Hardware. In Moshe Sipper, Daniel Mange, and Andrés Pérez-Uribe, editors, *Proceedings of the Second International Conference on Evolvable Systems (ICES'98)*, pages 57–65, Lausanne, Switzerland, 1998. Springer-Verlag.