Extracting and Re-Using Design Patterns from Genetic Algorithms using Case-Based Reasoning

Eduardo Islas Pérez
Instituto de Investigaciones Eléctricas
Av. Reforma # 113
Col. Palmira 62490
Temixco, Morelos, MEXICO
eislas@axp18.iie.org.mx

Carlos A. Coello Coello
CINVESTAV-IPN
Dept. Ing. Elec./ Secc. Comp.
Av. IPN No. 2508
México, D. F. 07300, MEXICO
ccoello@cs.cinvestav.mx

Arturo Hernández Aguirre
CIMAT
Área Computación
Callejón Jalisco s/n, Mineral de Valenciana
Guanajuato, Gto 36240, MEXICO
artha@cimat.mx

Abstract

This paper proposes a scheme in which case-based reasoning techniques are employed to extract design patterns from a genetic algorithm used to optimize combinational circuits at the gate level.

1 INTRODUCTION

In this paper we show how an evolutionary algorithm can produce simplification rules that human designers can use. We also show how case-based reasoning can be used to perform modular design of circuits. The idea is to use small components as building blocks to produce more complex circuits. Our approach is therefore, to use a database of solutions previously found that have some (potentially) useful information. Then, using techniques from case-based reasoning, we retrieve this information when designing similar circuits and incorporate it in the population of another evolutionary algorithm, as to reduce convergence times and to encourage modular design. Our work aims then to explore the potential of CBR combined with GAs to design combinational circuits which can be optimized. Optimality will be defined in terms of the number of gates employed by a function circuit (i.e., we wish to produce a circuit that matches all the outputs of the truth table and, at the same time, we want such a circuit to use as few gates as possible).

2 Statement of the Problem

This paper extends our previous work in combinational circuit design using genetic algorithms (GAs) [1], by using case-based reasoning techniques to extract and incorporate knowledge used by a genetic algorithm to design combinational circuits. The main

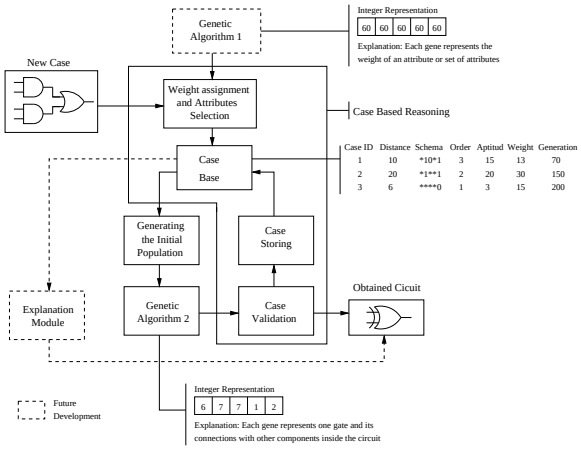

Figure 1: Proposed system to optimize combinational logic circuits using GAs and CBR.

emphasis of this work is to show the potential of incorporating domain-specific knowledge generated by the GA itself into other GAs used to solve similar problems. Other researchers have attempted similar schemes (e.g., [5, 4]), but only in a fairly limited way regarding combinational circuit design.

Our approach extracts knowledge at two stages of the evolutionary process: **at the end of a run**, and **during a run**. In the first case, the knowledge to be extracted are the Boolean laws used by the evolutionary algorithm to design a circuit. These laws are obtained after comparing the results produced from two or more runs of the GA (with different parameters) with the solution produced by a human expert. In the second case (extraction during a run), the knowledge extracted are the building blocks that the circuit structurally maintains during its evolutionary process. When some individuals arrive at a certain (predefined) threshold in their fitness value during the evolutionary process, it means that these circuits have evolved

long enough to contain good building blocks and we can then extract the knowledge that they contain and store it in a case base for further use. For the experiments described next, we use the genetic algorithm with integer representation and matrix representation (encoded as fixed-length linear chromosomes) that we have adopted in previous work [1].

Our GA uses a fitness function that works in two stages: first, it tries to reach the feasible region (i.e., it tries to produce circuits which match all the outputs of the truth table) and then, once feasible circuits are available, it tries to maximize the number of WIRE gates (WIRE means no gate; this type of gate is used to allow variable-length Boolean expressions within our fixed-length representation); this aims to produce feasible circuits that have as few gates as possible.

3 Proposed System

The proposed system that combines a GA with CBR is depicted in Figure 1. To understand better the way in which our system works, we will describe in more detail the process of extracting knowledge in the two situations previously mentioned:

1. **At the end of the evolutionary process**: We perform complete runs of a GA solving a certain circuit. Once a solution is found, a new case is formed with such a solution and the original problem. The original problem will be considered as the attributes in the case base and the solution will be the output of the case.

2. **During the evolutionary process**: In this case, our work is inspired on the research of Louis [3]. The GA records data for each individual in the population as it is created and evaluated. Such data includes a fitness measure, the genotype and chronological data, as well as some information on the individual's parents. This collection of data is the initial case data. Though normally discarded by the time an individual is replaced, all of the case data collected is usually contained in the genetic algorithm's population at some point and it is easy to extract. When a sufficient number of individuals have been created over a number of generations, the initial case data is sent to a clustering program. A hierarchical clustering program clusters the individuals according to both, the fitness and the alleles of the genotype. This clustering constructs a binary tree in which each leaf includes the data of a specific individual. The binary tree structure provides an index for the initial case base. The numbers at the leaves of the tree correspond to the case number (an identification number) of an individual created by the GA. An abstract case is computed for each internal node based on the information contained in the leaves and nodes beneath. The final case base includes: 1) cases corresponding directly to GA individuals (at the leaves) and 2) more abstract cases made up of information generalized from the leaves.

Table 1: Cases for knowledge extraction at the end of the evolutionary process.

ID-I-O	Table	Fit.	Genotype
1-3-2	01101001	39	3230132431232134103231
2-3-2	01101001	38	3230132431232144133204
3-3-2	01101001	39	0200234133241431231130

3.1 Representing circuits in the case base

Depending on the stage at which knowledge is extracted, the representation adopted to store it in the case base can vary:

1. **At the end of the evolutionary process**: The attributes contained in this part of the case base are the following: **Case ID, Number of Inputs, Number of Outputs, Output Values, Fitness, and Genotype**

   Some examples of this sort of cases stored in the case base are shown in Table 1. The attribute **ID-I-O** is the combined case ID, number of inputs, and number of outputs of the circuit. **Table** is used to verify that the outputs indicated in the truth table are satisfied.

2. **During the evolutionary process**: The best individuals are recognized during early generations of the evolutionary process. Afterwards they are stored as cases in the case base and retrieved in later generations. The attributes contained in the case base are the following: Case ID, Distance from the root of the tree to the level of the case, Schema for the case, Schema order, Average fitness, Weight (number of leaves—individuals—below), and Generation information (the earliest and latest leaf occurrence as well as the average in the subtree).

   Some examples of this sort of cases stored in the case base are shown in Table 2.

Table 2: Cases for knowledge extraction during the evolutionary process.

ID	Dist.	Schema	Order	Fit.	Wght.	Gen.
1	5	710*13*2*	6	30	6	50
2	2	**4*50*2*	4	60	8	30
3	8	163*14*41	7	15	4	67

Table 3: Truth table of the Example (a parity checker). Miniterms not shown have $F = 0$

A	B	C	D	F
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	1

Additionally, we also perform some analysis by hand to try to understand the way in which the GA performs the simplification of a circuit. As we will show in the examples presented next, the GA is able to rediscover several of the simplification rules commonly used in Boolean algebra and, furthermore, was able to discover "new" simplification laws that are stored in the case base and can also be used by human designers.

4 Example

The goal is to find the Boolean expression that corresponds to the circuit whose truth table is provided in Table 3.

We will start by providing the steps followed to extract knowledge at the end of the evolutionary process. First, we performed 10 runs using integer representation and the following parameters[1]: population size=600, maximum number of generations = 200, crossover rate = 0.6, mutation rate = 0.001. The best solution found from these runs has 9 gates and its corresponding Boolean expression is shown (under "GA Setup 1") in Table 4. This Boolean expression is not better than the best solution found by a Human Designer using Karnaugh maps (this solution has 6 gates). However, we additionally performed 10 more runs using a population size of 3000 and a maximum number of generations of 120. The best solution found from these runs has 4 gates (i.e., it is better than the solution produced by a human expert) and its corresponding Boolean expression is shown (under "GA Setup 2") in Table 4.

Table 4: Comparison of results between a human designer and two setups of our GA for the first example.

Human Designer
$X = ((A \oplus B)' \oplus (C \oplus D)')'$
6 gates
3 XORs, 3 NOTs
GA Setup 1
$X = ((A \oplus C) \oplus B)' \oplus ((B'B) + D')'$
9 gates
1 AND, 1 OR, 3 XORs, 4 NOTs
GA Setup 2
$X = ((A \oplus B) \oplus (C \oplus D))'$
4 gates
1 NOT, 3 XORs

4.0.1 Analysis

The next step was to analyze (by hand) the solutions produced by our GA with respect to those generated by the human designer:

$$X = ((A \oplus B)' \oplus (C \oplus D)')' = ((A \oplus B) \oplus (C \oplus D)') \quad (1)$$

We have discovered a "new" DeMorgan's theorem[2] for XOR gates of the type:

$$(S' \oplus T')' = (S \oplus T)' \quad (2)$$

A case stored in the case base as a product of the analysis at the end of the evolutionary process is shown in Table 4.0.1.

Then, we performed an analysis of the information extracted during the evolutionary process trying to detect the basic building blocks used by the GA to generate the best solutions produced. Figures 4 show

[1] The parameters indicated were empirically derived after performing a set of experiments.

[2] By "new" we mean that this DeMorgan theorem is not part of the basic set of Boolean algebra simplification rules normally adopted for circuit design.

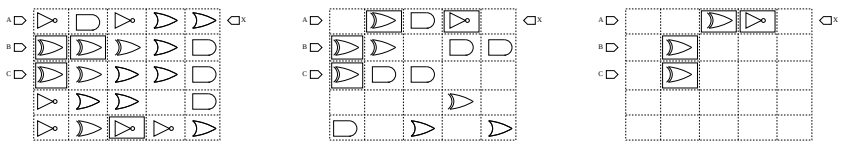

Figure 2: Solutions obtained by our GA at generations 9 (left), 67 (middle), and 101 (right).

Case		Solution	Description	Improved
1	$(S' \oplus T')'$	$(S \oplus T)'$	DeMorgan's XOR	4 - 2 = 2

Table 5: Case stored in the case base at the end of the evolutionary process for the circuit whose truth table is shown in Table 3.

several snapshots of the solutions produced by our GA with the second set of parameters previously described (population size = 3000, maximum number of generations = 120). From these pictures, we can see that the circuit has a fitness value of 17 at generation 9 and we were able to recognize the building blocks used by the GA (such building blocks are indicated with a thicker box). At generation 67, the maximum fitness has increased, reaching 27, and we can observe that the building blocks previously mentioned have moved to a different position. Finally, when reaching generation 101, we have a fitness of 37 (i.e., a feasible circuit with only 4 gates). Although the building blocks are in a different position, the circuit has the same behavior as in earlier stages of the design. For this reason, we proceed to store it in our case base.

5 Conclusions and Future Work

We have illustrated the potential of combining CBR with a GA to improve performance. The introduction of domain-specific knowledge within a GA is not straightforward, and care must be taken of not biasing the search too strongly as to produce premature convergence. The mixture of individuals proposed in this work (10% of the population were taken from the case base and 90% were randomly generated) seems to be a good choice, at least for the small and medium size circuits used in our experiments [2]. However, more experimentation in this direction is still necessary. We are also currently extending our system to use it with more complex circuits.

6 Acknowledgements

The second author acknowledges support from CONACyT through project No. 32999-A. The last author acknowledges partial support for this work from Consejo de Ciencia y Tecnología del Estado de Guanajuato (CONCyTEG) through project No. 01-02-202-111.

References

[1] 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.

[2] Eduardo Islas Pérez. Development of a Learning Platform using Case Based Reasoning and Genetic Algorithms. Case Study: Optimization of Combinational Logic Circuits. Master's thesis, Maestría en Inteligencia Artificial, Facultad de Física e Inteligencia Artificial, Universidad Veracruzana, November 2000. (Available at: `http://delta.cs.cinvestav.mx/~ccoello/`).

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

[4] Sushil J. Louis and Judy Johnson. Solving Similar Problems using Genetic Algorithms Case-Based Memory. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 283–290, San Francisco, California, 1997. Morgan Kaufmann Publishers.

[5] Sushil J. Louis, Gary McGraw, and Richard Wyckoff. Case-based reasoning assisted explanation of genetic algorithm results. *Journal of Experimental and Theoretical Artificial Intelligence*, 5:21–37, 1993.