Genetic Algorithms and Case-Based Reasoning as a Discovery and Learning Machine in the Optimization of Combinational Logic Circuits

E. Islas Pérez[1], C. A. Coello Coello[2] , A. Hernández Aguirre[3] and A. Villavicencio Ramírez[1]

[1] Instituto de Investigaciones Eléctricas
Av. Reforma 113, Col. Palmira, 62490 Temixco Morelos, México
eislas@iie.org.mx, avilla@iie.org.mx
[2] CINVESTAV-IPN; Depto. de Ingeniería Eléctrica, Sección de Computación,
Av. IPN No. 2508, Col. San Pedro Zacatenco, México, D.F. 07300, México,
ccoello@cs.cinvestav.mx
[3] CIMAT, Depto. de Computación, Guanajuato, Gto. 36240, México
artha@cimat.mx

Abstract. In this paper we show how case-based reasoning techniques can be used to extract and reuse solutions previously found by a heuristic (a genetic algorithm in our case) used to solve problems in a specific domain (MSI and SSI combinational circuit design). This reuse of partially built solutions allows us to improve convergence time of our heuristic since the building blocks of the "good" solutions in design space are incorporated earlier in the search process. Our system is illustrated with the design of a full adder circuit being this circuit the solution of two interconnected half-adder. Furthermore, with the analysis of the obtained results we are able to rediscover several of the traditional Boolean rules used for circuit simplification and we are also able to find a new and interesting simplification rule.

1 Introduction

The focus of this paper is to show how Case-Based Reasoning (CBR) techniques can be used to extract and reuse design patterns that emerge from the evolutionary process of a Genetic Algorithm (GA) used to design combinational circuits. We build a database of solutions previously found (to certain circuits) which we consider having some (potentially) useful information. Then, using CBR techniques, we retrieve this information when solving similar circuits, aiming to improve convergence time of the GA used.

2 Related Work

Apparently, the first attempt to combine CBR and GAs was done by Louis et al. [5]. In this paper, the authors use CBR-principles to explain solutions found by a GA. These ideas were further developed by Louis & Johnson [4]. Although Louis & Johnson [4] used a few examples from circuit design (mainly parity checkers) to illustrate their principles, they did not focus their work specifically on the design of combinational circuits as in our case. Nevertheless, our current proposal has been influenced by this prior work. Several other researchers have proposed approaches that combine CBR and GAs (see for example [8]). However, the emphasis of these papers has been to illustrate the benefits of this sort of hybrid scheme rather than emphasizing a certain application domain like in our case. Also, some researchers in evolvable hardware have pointed out the potential benefits of using GAs as a discovery engine capable of producing novel and even inspirational designs. Miller et al. [6], for example, showed that through the evolution of a hierarchical series of examples, it was possible to rediscover the well-known ripple-carry principle for building adder circuits of any size. Recently, Thomson [9] explored the potential of evolving larger systems more quickly via a method of visualizing the subcomponents of the final solution when they appear.

3 Case-based Reasoning

CBR is a problem-solving paradigm that in many respects is fundamentally different from other major AI approaches [2]. Instead of relying solely on general knowledge of a problem domain, or making associations along generalized relationships between problem descriptors and conclusions, CBR is able to utilize the specific knowledge of previously experienced, concrete problem situations (cases). A CBR system can be divided in the following main stages:

1. *Identifying the new problem:* The system receives the input case (new problem) and analyses its most important attributes and characteristics in order to search amongst the cases that are most similar to the cases in the case base. The attributes used to measure the similarities between the cases are called indexes.

2. *Finding cases with similarities to the new case:* The following step is to find the cases that have more attributes in common with the attributes of the new case using the indexes found in the previous step. Sometimes it is necessary to reduce the subset in order to find the most relevant cases. The algorithm should be fast and efficient due to the fact that cases retrieval is a critical and important aspect when the case base is sufficiently large.

3. *Arriving at the Solution:* Once we have the most similar cases, the system starts the adaptation process, which consists of the combination and modification of the most similar cases to form a new solution, and additionally an interpretation or an explanation depending on the application of the system.

4. *Evaluating the solution:* The solution obtained in the previous stage is a tentative or potential solution. It is necessary to do an evaluation of the proposed solution before giving it to the final user.

5. *Assignment and storing of the new case:* Once the solution has been created and evaluated, it is given to the user and then it is possible to create a new case. This new case is formed from the solution found and the original case (problem). Indexes are assigned to the new case and it is stored in the case base.

6. *Explaining, repairing and testing:* If the solution fails, it is important that the system obtains and analyses the information in order to avoid making the same mistakes. If something unusual happens, the system should try to explain it.

4 Statement of the Problem

We will extract 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 will be the Boolean laws used by the evolutionary algorithm to design a circuit. These laws will be obtained after comparing the results produced from two or more runs of the GA (with different parameters) with the solution produced by a human designer. In the second case, the knowledge extracted will be the building blocks that the circuit structurally maintains during its evolutionary process. When some individuals arrive at a certain threshold (given by the circuit when all the outputs are accomplished) 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. Our approach consists of storing solutions that were previously generated by the same GA and use them as a memory of "past experiences". Then, we can use a mechanism to detect cases similar to the one being solved and retrieve from this "memory" some solutions that can be useful to solve the problem at hand.

In this work we use a matrix encoding (encoded as fixed-length linear chromosomes) that we have adopted in previous work [1] combined with tournament selection, one-point crossover and uniform mutation. For reasons of simplicity we used only five type of gates (AND, OR, NOT, XOR, WIRE [1]). In order to evaluate the fitness value of each circuit we counted the number of WIRES (absence of gates) in the matrix encoding.

5 Proposed System

The proposed system that combines a GA with CBR is depicted in Fig. 1.

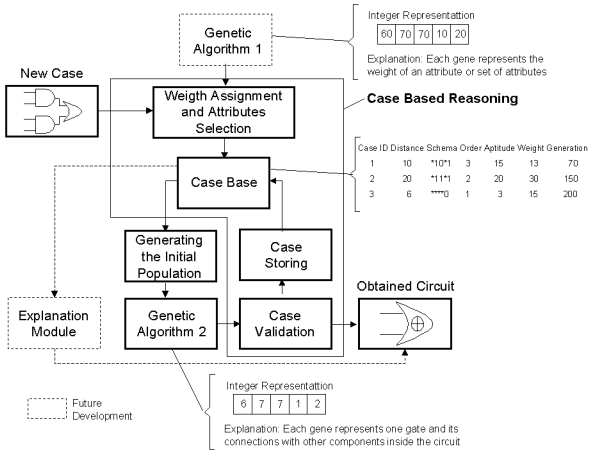

Fig. 1. Proposed system to optimize combinational logic circuits using GAs and CBR

The process of extracting knowledge in the two situations previously mentioned can be described as follows:

1. *At the end of the evolutionary process:* In this case, 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. As a future work we will include in the system other attributes, in order to have indexes that help to retrieve the most similar cases in a more efficient way.

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 GA'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.

5.1 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 cases will be stored from problems that have been solved previously and they will be used for seeding the initial population of a GA. 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.

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

Case ID	Num. Inputs	Num. Outputs	Output Values	Fitness	Genotype
1	3	2	0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1	39	3 2 3 0 1 3 2 4 3 1 2 3 2 1 3 4 1 0 3 2 3 1
2	3	2	0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1	38	3 2 3 0 1 3 2 4 3 1 2 3 2 1 4 4 1 3 3 2 0 4
3	3	2	0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1	39	0 2 0 0 2 3 4 1 3 3 2 4 1 4 3 1 2 3 1 1 3 0

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. Some of the attributes that are contained in this part of the case base are the following [3]: case ID, distance from the root of the tree to the level of the case, schema for the case, scheme order, average fitness, weight (number of leaves or individuals below), and generation info (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

Case ID	Distance	Schema	Order	Fitness	Weight	Generation
1	5	710*13*2*	6	30	6	50
2	2	**4*50*2*	4	60	8	30
3	8	350610*7*	7	65	4	32

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

6 An Example

Next, we will provide an example of how is the knowledge extracted both at the end and during the evolutionary process of a GA with integer representation used to design combinational logic circuits at the gate-level.

Table 3. Truth table for the circuit of the example (a parity checker)

A	B	C	D	X
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

We want to find the Boolean expression that corresponds to the circuit whose outputs must have exactly the same behaviour as the truth table shown 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: 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 (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. In order to say that a solution is better than other it is important to say that in the logic circuit field simplicity depends on the means of measurement that we use. One possibility is the number of gates, which measures circuit area.

Table 4. Comparison of results between a human designer and two setups of our GA

Human Designer
X = ((A ⊕ B)' ⊕ (C⊕D)') '
6 gates
3 NOT, 3 XOR
GA Setup 1
X = ((A⊕C) ⊕ B) ' ⊕ ((B'B) + D')'
9 gates
1 AND, 1 OR,3 XOR, 4 NOT
GA Setup 2
X = ((A ⊕ B) ⊕ (C⊕D)) '
4 gates
1 NOT, 3 XOR

Analysis. The next step was to analyse (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" theorem for XOR gates of the type:

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

A case stored in the case base as a product of the analysis at the end of the evolutionary process is shown in Table 5. Using other circuits, we were able to produce several other cases and we were able to rediscover several simplification laws from Boolean algebra and even discover some new ones. For example, we discovered some non-trivial simplifications, such as $(A + (A \oplus B)) \oplus (A \oplus B) = AB$. All these cases were stored in the case base with the aim of reusing this knowledge with other circuits.

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

Original case	Solution	Description	Number of gates eliminated
$(S' \oplus T)'$	$S \oplus T$	"New" theorem applied to XOR gates obtained from the comparison between the solution found by the second run of the GA and the solution obtained by a human designer.	3 - 1 = 2

Then, we performed an analysis during the evolutionary process, trying to detect the basic building blocks used by the evolutionary algorithm to generate the best solutions produced. Fig. 2 shows several snapshots of the solutions produced by our GA (the wiring is omitted for reasons of clarity) 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 shadowed 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 the circuits in earlier stages. For this reason we proceed to store it in our case base. This same process was applied to several other circuits, including a subtractor, a 2-bit magnitude comparator, a half-adder and a full adder. The details of these experiments are available at [7].

6.1 A Case Study: Use of CBR to Design a 2x2 Bit Full Adder

To provide an insight into some of the possible applications of our work, we chose a second example in which we want to illustrate how can we use previously acquired

knowledge (derived from the design of a 2x2 bit half adder) to produce a 2x2 bit full adder. We were interested in analysing different possibilities regarding the use of CBR to improve the performance of the GA. Therefore, we decided to perform three experiments:

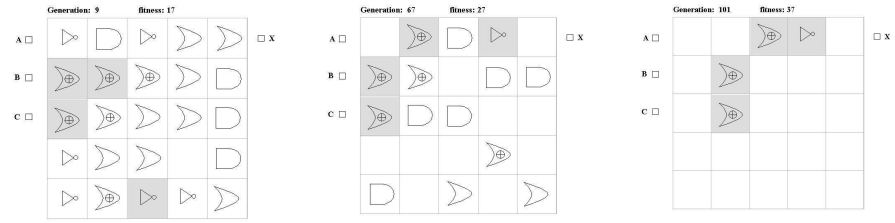

Fig. 2. Solutions obtained by our GA at generations 9 (top left), 67 (top right), and 101 (bottom) for the circuit whose truth table is shown in Table 3

First Experiment: Only previous solutions to the full adder circuit with different fitness values were stored in a case base and some of these individuals were retrieved to seed a percentage of the initial population of a GA before running it. The individuals were taken from different generations with different fitness values in a previous set of runs for the full adder circuit. The initial population was a mixture of previous solutions (10%) and random solutions (90%). This mixture is necessary to avoid an excessive selection pressure that would cause premature convergence. However, the issue of finding the proper number of cases to be injected in the population of a GA is still an open research area [4]. The best-known solution to this circuit has a fitness of 36 (i.e., a feasible circuit with 5 gates), and we stored solutions with a fitness value of up to 22. This also intends to reduce selection pressure in the GA.

Second Experiment: Some solutions to different logic circuits including the full adder, the half-adder, the comparator and other circuits were stored in the case base. The most similar cases would then be used to seed a portion of the initial population of a GA before running it. The same mixture of individuals as before was adopted in this case.

Third Experiment: Some solutions to different logic circuits including all the circuits as in the previous experiment, but without including the full adder circuit were stored in the case base. The most similar cases would then seed a part of the initial population of a GA were retrieved before running it. The same mixture of individuals as before was adopted in this case.

The results produced from the three experiments are shown in Fig. 3. As we expected, when previous knowledge is used, the GA arrives more rapidly to the best-known solution to this circuit. In the first experiment, our GA converges, on average, at generation 87, whereas the GA without knowledge required almost 100 generations

to converge (on average). In the second experiment, the GA performed the same as in the first experiment. The interesting issue to analyse here is that our system decided to extract from the case base the previous solution to the full adder (with fitness of 22), instead of the solution to the half adder. This is explained by the fact that the full adder presents a greater resemblance with the circuit being designed. The experiment showed us the capability of our system to discriminate correctly among several circuits. In our third experiment, we can observe that the GA begins to evolve from a fitness value of 14 in generation one, analogously to the GA with its initial population randomly generated. However, the circuit evolves in a completely different way after that, due to the fact that the system retrieves as the most similar case the previous solution found for the half-adder circuit. In this case, the GA that uses the case base finds a valid circuit at generation 34, whereas the conventional GA finds a valid circuit at generation 45. This illustrates how the use of case base reasoning can actually help the GA to explore the search space in a more efficient way. As an important conclusion we can say that our approach was able to find the solution more rapidly although in the case base only existed a similar solution.

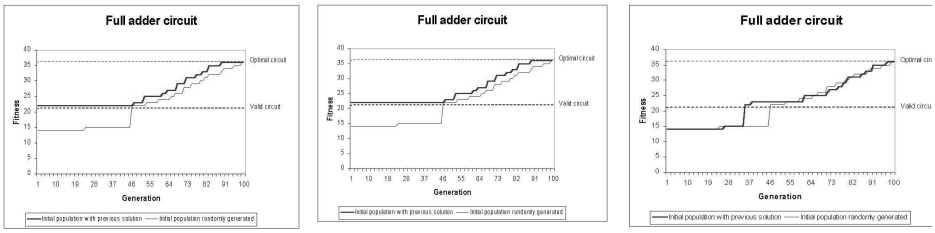

Fig. 3. Comparison of results for the first (top left), second (top right) and third (bottom) experiments. The label "experiment" indicates the runs in which we used cases previously generated by other runs of our GA (i.e., use of the case base)

7. 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 [7] (many details were excluded due to space limitations). However, more experimentation in this direction is still necessary. Our approach extends some of the previous efforts to extract design patterns from a GA used to design circuits [6, 9], since we show not only how these patterns can be extracted, but also how can they be reused by a GA to design more complex circuits using as building blocks the circuits previously solved. Furthermore we have put forward the view that the analysis of

evolutionary algorithms results can be regarded as a discovery engine of well-known laws or theorems, and new principles or theorems. We studied this idea in the context of digital logic. We suggested that new principles can be discovered by examining a series of evolved designs, in our case, for some arithmetic logic circuits. We examined the concept of the space of all circuit representations, but we feel that similar ideas may be well carried over the general field of design.

8. Acknowledgements

The first author acknowledges to some students in the UAEM for doing some of the analysis shown in this paper. The second author acknowledges partial support from CINVESTAV through project JIRA'2001/08, and from CONACyT through the NSF-CONACyT project No. 32999-A. The third author acknowledges partial support for this work by grant No. 01-02-202-111 from Consejo de Ciencia y Tecnología del Estado de Guanajuato (CONCyTEG).

References

1. C.A. Coello Coello, A.D. Christiansen, and A. 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. V. Kolodner. *Case Based Reasoning*. Morgan Kaufmann Publishers, San Mateo, California, (1993).
3. S.J. Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University, (August 1993).
4. S.J. Louis and J. 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. S.J. Louis, G. McGraw, and R. Wycko. Case-based reasoning assisted explanation of genetic algorithm results. *Journal of Experimental and Theoretical Artificial Intelligence*, 5:21-37, (1993).
6. 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).
7. E. 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).
8. C. L. Ramsey and J.J. Grefenstette. Case-Based Initialization of Genetic Algorithms. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84-91, San Mateo, California, (1993). Morgan Kauffman Publishers.
9. P. Thomson. Circuit Evolution and Visualisation. In Julian Miller, Adrian Thompson, Peter Thomson, and Terence C. Fogarty, editors, *Evolvable Systems: From Biology to Hardware*, pages 229-240. Springer-Verlag, Edinburgh, Scotland, (April 2000).