Extraction of Design Patterns from Evolutionary Algorithms using Case-Based Reasoning

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

[1] Instituto de Investigaciones Eléctricas
Av. Reforma #113, Col. Palmira
62490 Temixco Morelos, MEXICO
eislas@iie.org.mx

[2] 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

[3] Tulane University
Department of Electrical Engineering and Computer Science
211 Stanley Thomas Hall
New Orleans, Louisiana 70118, USA
hernanda@eecs.tulane.edu

Abstract. In this paper we show a scheme based on case-based reasoning to extract design patterns from a genetic algorithm used to optimize combinational circuits at the gate-level. The approach is able to rediscover several of the traditional Boolean rules used for circuit simplification and it also finds new simplification rules. Also, we illustrate how the approach can be used to reduce convergence times of a genetic algorithm using previously found solutions as cases to solve similar problems.

1 Introduction

Although it is difficult for an evolutionary algorithm to suggest directly new design principles, it is feasible to infer such principles through a careful study and analysis of its behavior on a set of examples. That is precisely the focus of this paper. We propose that by employing case-based reasoning (CBR) techniques, we can extract and reuse design patterns that emerge from the evolutionary process of a genetic algorithm (GA). In fact, we will see how some of these rules are really the same traditionally used by human designers. However, others are entirely new simplification rules which, in some cases, may not even be intuitive to a human designer.

Another interesting aspect of this work is that we show how CBR can be used to solve more efficiently similar circuits to those previously solved. This idea, although intuitive, is not completely straightforward in practice, since the selection pressure of an evolutionary algorithm may destroy partial solutions to a problem. 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 (similarity has to be defined according to certain criteria in this context) and incorporate it in the population of another evolutionary algorithm, as to reduce convergence times. The system will be illustrated with the design of a full adder.

2 Related Work

Apparently, the first attempt to combine case-based reasoning (CBR) and GAs was done by Louis et al. [6]. In this paper, the authors use CBR-principles to explain solutions found by a GA. This same idea was also discussed in Louis' dissertation [4], where he proposed a system that combined CBR with GAs to improve performance of the GA. These ideas were further developed by Louis & Johnson [5] and by Liu [3]. Although Louis [4] and Louis & Johnson [5] 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 [9, 10]. 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. [7], 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. However, no CBR is used in this work. Recently, Thomson [11] explored the potential of evolving larger systems more quickly via a method of visualizing the subcomponents of the final solution when they appear. Taking these partially evolved solutions from short runs and feeding them to another GA, the convergence time of the GA can be improved. This work is closer to our own, but unlike our proposal, Thomson does not use CBR in his system.

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). Finding a similar past case, and reusing it in the new problem situation helps to solve a new problem. A second important difference is that CBR is also an approach to incremental, sustained learning, since a new experience is retained each time a problem has been solved, making it immediately available for future problems. A CBR system can be divided in the following main stages (see Fig. 1):

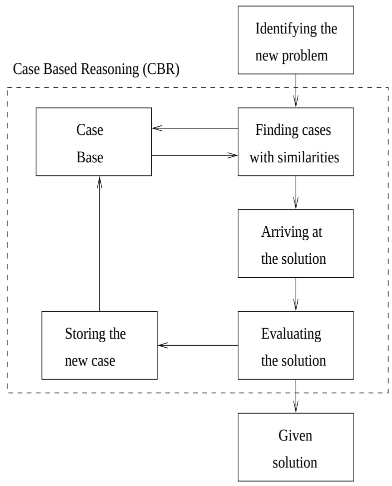

Fig. 1. General structure of a CBR system.

1. **Identifying the new problem:** The system receives the input case (new problem) and analyzes 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 and the design is a critical and important aspect when the case base is sufficiently large. The selection of cases from the case base could be considered as analogous to natural selection due to the fact that it is based only on the distance measure (similarity rather than fitness) between the new case and each case in the case base.
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. In most applications it is better if the system explains how it finds the new case.
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. This evaluation should show the qualities and weaknesses of the solution for the evaluation of its usefulness.
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 analyzes the information in order to avoid making the same mistakes. If something unusual happens, the system should try to explain it. Subsequently, the system repairs the solution based on the explanation and returns to the evaluation stage.

4 Statement of the Problem

We 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 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, 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. 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 (or past experiences) that can be useful to solve the problem at hand. For the experiments described next, we use a GA with integer representation and matrix encoding (encoded as fixed-length linear chromosomes) that we have adopted in previous work [1].

5 Proposed System

The proposed system that combines a GA with CBR is depicted in Fig. 2.

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**: 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

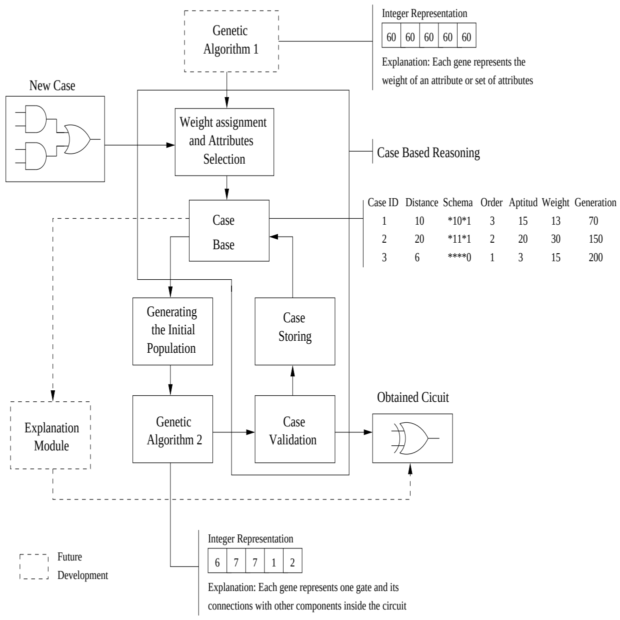

Fig. 2. Proposed system to optimize combinational logic circuits using GAs and CBR.

problem will be considered as the attributes in the case base and the solution will be the output of the case. The system will assign other attributes, in order to have indexes that help retrieving 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 [4]. The GA records data for each individual in the population as it is created and evaluated. Such data include 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. An abstract case is computed for each internal node based on the information contained in the leaves and nodes beneath.

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[1]: case ID, number of inputs, number of outputs, output values, fitness, and genotype. Some examples of these cases are shown in Table 1.

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

Case ID	No. Inputs	No. Outputs	Output Vals	Fitness	Genotype
1	3	2	01101001000010111	39	3230132431232134103231
2	3	2	01101001000010111	38	3230132431232144133204
3	3	2	01101001000010111	39	0200234133241431231130

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: 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	163*14*41	7	15	4	67

Additionally, we also performed 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 some of the examples presented next, the GA was 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.

[1] This scheme presents certain resemblance with the one proposed by Louis [4].

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 subtractor).

A	**B**	**C**	**X**	**Y**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

We want 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: population size=100, maximum number of generations = 2000, crossover rate = 0.5, mutation rate = 0.006. The best solution found from these runs has 7 gates and its corresponding Boolean expression is shown (under "AG Setup 1") in Table 4. This Boolean expression can be contrasted with the best solution found by a Human Designer using Karnaugh maps (this solution has 16 gates). Additionally, we performed 10 more runs using a population size of 700 and a maximum number of generations of 400. The best solution found from these runs has 6 gates and its corresponding Boolean expression is shown (under "AG Setup 2") in Table 4.

Analysis The next step was to analyze (by hand) the solutions produced by our GA with respect to those generated by the human designer: If we take the solution found by the human designer and we factorize C and C' in Y, we have that:

$$Y = A'B'C + A'BC' + AB'C' + ABC = C(A'B' + AB) + C'(A'B + AB') \quad (1)$$

To transform this equation in terms of an XOR gate:
If $S = C$ then it is necessary that $S' = C'$
and if $T = A'B' + AB$ then it is necessary that $T' = A'B + AB'$
If we apply the DeMorgan's theorem to T, we have that:

$$T' = (A'B' + AB)' = (A'B')'(AB)' = (A + B)(A' + B') \quad (2)$$

Applying the distributive law and some basic theorems:

$$T' = (A + B)(A' + B') = AA' + AB' + A'B + BB' = A'B + AB' \quad (3)$$

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

Human Designer
$X = A'B + A'C + BC$
$Y = A'B'C + A'BC' + AB'C' + ABC$
16 gates
8 ANDs, 5 ORs, 3 NOTs
AG Setup 1
$X = (B \oplus (A \oplus C)) \oplus (A \oplus AC)$
$Y = (B \oplus (A \oplus C)) \oplus (B(A \oplus C)$
7 gates
2 ANDs, 1 OR, 4 XORs
AG Setup 2
$X = (A'(B \oplus (A \oplus C)) + BC$
$Y = B \oplus (A \oplus C)$
6 gates
2 XORs, 2 ANDs, 1 OR, 1 NOT

Verifying that $T' = A'B + AB'$ is the negation of $T = A'B' + AB$. Then we can rewrite the eq. (1) as follows:

$$Y = C(A'B' + AB) + C'(A'B + AB') = C \oplus (A'B + AB') \tag{4}$$

If we apply the operation of a XOR gate to the operand between parentheses, we have:

$$Y = C \oplus (A'B + AB') = C \oplus (A \oplus B) \tag{5}$$

Applying the commutative and associative laws, we have:

$$Y = C \oplus (A \oplus B) = B \oplus (A \oplus C) \tag{6}$$

We have the same result obtained from the solution found by the second set of runs of the GA for Y, so we can store this equality in the case base and we will have a reduction in the number of gates. The following can be easily seen from the previous equations:

- In eq. (1) we rediscovered the distributive law
- In eq. (2) we rediscovered the DeMorgan's theorem
- In eq. (3) we rediscovered the distributive law and some basic theorems
- In eq. (4) we rediscovered the operation of an XOR gate
- In eq. (5) we rediscovered the operation of an XOR gate
- In eq. (6) we rediscovered the commutative and associative laws

The cases stored in the case base as a product of the analysis at the end of the evolutionary process are summarized 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 a "new" DeMorgan

Table 5. Cases 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
Case 1	$A'B'C + A'BC'$ $+AB'C' + ABC$	$(A \oplus B) \oplus C$	Comparison between a human designer and the best solution found by the GA	13 - 2 = 11
Case 2	$(A'B' + AB)'$	$A'B + AB'$	Case obtained with the comparison between a human designer and the best solution found by the GA	6 - 5 = 1

theorem applied to XOR gates: $(X \oplus Y')' = X \oplus Y = X' \oplus Y'$. This "new" law says that if we apply the DeMorgan's theorem to an XOR gate, then we only need to negate one of the components of the original expression. We also discovered some more complex 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.

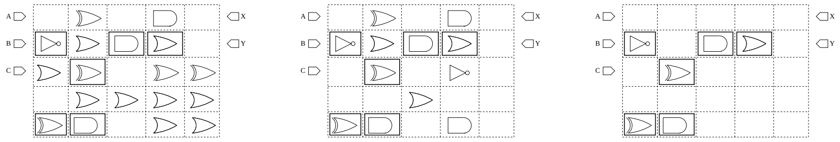

Fig. 3. Solutions obtained by our GA at generations 325 (left), 366 (middle), and 398 (right) for the circuit whose truth table is shown in Table 3.

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. 3 show several snapshots of the solutions produced by our GA with the second set of parameters previously described (population size = 700, maximum number of generations = 400). From these pictures, we can see that the circuit has a fitness value of 23 at generation 325 and we were able to recognize the building blocks used by the GA (such building blocks are indicated with a thicker box). At generation 366, the maximum fitness has increased, reaching 29, and we can observe that the building blocks previously mentioned remain in their same position. Finally, when reaching generation 398, we have a fitness of 35 (i.e., a feasible circuit with only 6 gates). Since the building blocks previously mentioned remain in the same position, we proceed to store them in our case base. The building block found will be stored using integers (since our GA used an integer representation), using asterisks (i.e., 'don't care' symbol) for those positions in the circuit different from the building block. This same process was applied to several other circuits, including a parity bit checker, a 2-bit magnitude comparator, a half-adder and a full adder. The details of these experiments are available at [8].

6.1 A Case Study: Use of CBR to Design a 2×2 bit 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 half 2×2 adder) to produce a full 2×2 adder. We were interested in analyzing different possibilities regarding the use of CBR to improve the performance of the GA. Therefore, we decided to perform three experiments:

- **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 [5]. 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. 4. 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 analyze 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

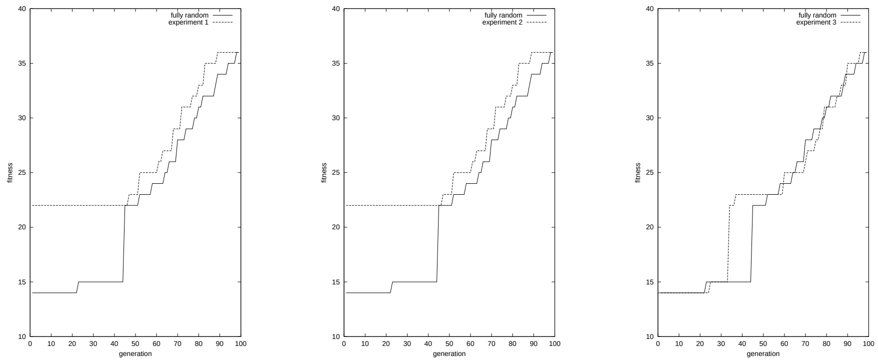

Fig. 4. Comparison of results for the first (left), second (middle) and third (right) experiments. The label "experiment" indicates the runs in which we used cases previously generated by other runs of our GA (i.e., use of the case base).

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.

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 [8] (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 [7, 11], since we show not only how these patterns can be extracted, but also how can they be reused by a GA to design other circuits. More important yet, is the fact that this sort of system can be applied to other domains, and that is precisely one of the future research paths that we would like to explore.

8 Acknowledgements

The second author acknowledges partial support from CINVESTAV through project JIRA'2001/08, and from CONACyT through NSF-CONACyT project No. 32999-A. The third author states that support for this work was provided in part by 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. 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. X. Liu. Combining Genetic Algorithms and Case-based Reasoning for Structure Design. Master's thesis, Department of Computer Science, University of Nevada, 1996.
4. S.J. Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University, August 1993.
5. 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.
6. S.J. Louis, G. McGraw, and R. Wyckoff. Case-based reasoning assisted explanation of genetic algorithm results. *Journal of Experimental and Theoretical Artificial Intelligence*, 5:21–37, 1993.
7. 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.
8. 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.
9. 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.
10. J.W. Sheppard and S.L. Salzberg. Combining Genetic Algorithms with Memory Based Reasoning. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 452–459, San Francisco, California, July 1995. Morgan Kaufmann.
11. 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.