Evolutionary Synthesis of Logic Circuits using Information Theory

Arturo Hernández Aguirre[1] and Carlos A. Coello Coello[2]
[1] *Center for Research in Mathematics (CIMAT)*
Department of Computer Science
Guanajuato, Gto. 36240, MEXICO
artha@cimat.mx
[2] *CINVESTAV-IPN*
Evolutionary Computation Group
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

Abstract. In this paper, we propose the use of Information Theory as the basis for designing a fitness function for Boolean circuit design using Genetic Programming. Boolean functions are implemented by only replicating binary multiplexers. Entropy-based measures, such as Mutual Information and Normalized Mutual Information are investigated as tools for similarity measures among circuits. Three fitness functions are built over a primitive one. We show that the landscape of Normalized Mutual Information is more amenable for being used as a fitness function than simple Mutual Information. A comparison of synthesized (through evolution) and minimized circuits through other methods denotes the advantages of the Information-Theoretical approach.

Keywords: evolutionary algorithms, genetic programming, circuit synthesis, computer-aided design, evolvable hardware, information theory

1. Introduction

The implementation of Boolean functions using the minimum number of components is important for ASIC circuit designers and programmable devices since silicon surface on a chip is a limited resource. Classic graphical methods such as Karnaugh Maps become harder to use when the number of variables increases, and it becomes impossible to use for a relatively small number of variables. Tabular methods such as Quine-McCluskey, although amenable for digital computers, have been proved to need memory in the order of 3^n. Such simple automated design tools make use of *a priori* knowledge that human experts have extracted from the problem domain. Knowledge is stored and represented in the form of axioms and rules, for example, from the Boolean algebra. For many design domains, human knowledge encoded in this form suffices to automate the design process. For some other areas such as logic circuit minimization, humans have not yet derived the whole set of rules that would allow to find the smallest circuit that implements an arbitrary Boolean function. Therefore, computation of the smallest circuit is achieved by searching for a solution in a combinatorial space spawned by operators of Boolean algebra.

As noted, the search space humans know (in some domains) is constructed by a deduction process, or repetitive application of the rules of the problem domain over an initial seed. But the search space could contain other elements not known to humans if it is created in some other way. Evolutionary computation methods build the search space in a bottom-up fashion by combining only some sampling elements (Thompson et al., 1999) (called individuals of the population). Hence, solutions found in this space challenge human designers since the deduction rules that lead to them are not common and, in many cases, such rules are in fact unknown.

In this article we propose the use of Genetic Programming to synthesize logic circuits using binary multiplexers ("mux" or "muxes"). Here, muxes are the only element replicated to construct circuits. Since such circuits are equivalent to the Shannon expansion of the Boolean formula implemented, the symbolic representation is equivalent to the real circuit. We also propose the use of Information Theory concepts for the design of fitness functions.

The organization of this paper is the following: In Section 2, we describe some previous related work. Section 3 provides the detailed description of the problem that we wish to solve. Section 4 provides the mathematical foundations for justifying the use of multiplexers as universal logic elements. Some basic concepts of information theory and genetic programming are provided in Sections 5 and 6, respectively. The use of entropy for circuit design is discussed in Section 7. With these ideas in mind, we introduce fitness functions for evolutionary circuit design in Section 8. A set of experiments illustrating the design of logic circuits is described in Section 9. In Section 10, we provide some final remarks and our conclusions and, finally, in Section 11, we briefly discuss some possible paths for future research.

2. Previous Work

Most of the previous work in this domain has been done using Genetic Algorithms (GAs). We could consider Genetic Programming (Koza, 1992) as an extension of GAs in which a tree-based representation is used instead of the traditional linear binary chromosome adopted with GAs.

Louis (1993) introduced the use of GAs for the design of combinational circuits, and the use of a matrix array inside of which a circuit is evolved. A layer or stage of his circuits was constrained to get its inputs only from the previous stage. Thus, he defined a new operator called *masked crossover* that exploits information unused by standard genetic operators over the matrix representation. After Louis, several authors followed his representation, for example, Miller et al. (1998) also evolved logic circuits in a matrix but the position of the circuit output is also considered a variable. In their approach, Miller et al. encode a set of complex Boolean expressions instead of simple gate functions aiming to design more complex circuits given the set of more powerful primitives. In principle, the approach is sound but, unfortunately, its main drawback is its lack of flexibility to handle a large number of inputs. Miller et al. (2000; 2000a), and also Louis and Johnson (1997), studied the combination of GAs with Case-Based Reasoning tools that incorporate and preserve knowledge about the problem domain.

Our own previous work using GAs for circuit design (Coello Coello et al., 1996; Coello Coello et al., 1997; Coello Coello et al., 2000; Coello Coello et al., 2001), shows succesful results when small circuits are evolved inside the previously mentioned matrix representation. From our research using this sort of encoding, we concluded that the matrix causes a strong representation bias since some inputs and gates are favored by the genetic operators in a probabilistic sense. In an attempt to deal with such a biased representation, in more recent work, we proposed another approach based on genetic programming and multiplexers that seems to have a more neutral representation (Hernández Aguirre et al., 1999; Hernández Aguirre et al., 2000; Hernández Aguirre et al., 2000b).

Koza (1992) has used Genetic Programming to design combinational circuits, but his emphasis has been in the generation of functional circuits (in symbolic form) rather than their optimization. Iba et al. (1997), have studied the Boolean function learning problem at gate-level concluding that it is harder for a GA to learn than what it is for a neural network. Note however, that we have showed that it is in fact possible for a GA to learn Boolean functions if we estimate correctly the VC dimension of our design tool (Hernández Aguirre et al., 2000a; Hernández Aguirre et al., 2001).

Claude Shannon suggested the use of information entropy as a measure of the amount of information contained within a message (Shannon, 1948). Thus, entropy tells us there is a limit in the amount of information that can be removed from a random process without information loss. For example, music can be (losslessly) compressed and reduced up to its entropy limit. Further reduction is only possible at the expense of information loss (Weaver and Shannon, 1949). In a few words, entropy is a measure of disorder and it constitutes the basis of Information Theory.

The ID3 algorithm for the construction of classifiers (based on decision trees) probably is the best known classification algorithm among computer scientists that relies on entropy measures (Quinlan, 1983). For ID3, an attribute is more important for concept classification if it provides greater "information gain" than the others.

Information Theory (IT) was early used by Hartmann et al. (1982) to convert decision tables into decision trees. Boolean function minimization through IT techniques has been approached by several authors (Kabakcioglu et al., 1990; Lloris et al., 1993). Some work to the proposal presented in this article can be found in Luba et al. (2000a), which address the synthesis of logic functions using a genetic algorithm and a fitness function based on conditional entropy. Their system (called *EvoDesign*) works in two phases: first, the search space is partitioned into subspaces via Shannon expansions of the initial function. Then the GA is started in the second phase. The authors claim that the partition of the space using entropy measures is the basis for their success. In their domain, a fitness function based on Mutual Information seemed to work well. Note however, that in our case such an approach did not produce good results.

3. Problem Statement

For the purposes of this article, let us consider a Boolean function specified by its truth table. The problem of interest to us is the design of the smallest possible logic circuit, with the minimum number of binary multiplexers (described in Section 4), that implements the given function. The design metric driving the implementation is the number of components. Therefore, the best among a set of circuits with the same functionality is the one with the lowest number of components. Our goal is to find 100% functional circuits, specifying components and connections, instead of a symbolic representation of it. Thus, the approach of this paper could be classified as "gate-level synthesis".

Since the number of circuit components is unknown for most circuits, the use of a heuristic method such as Genetic Programming seems appropriate. Also, the tree-like structure of the circuits makes Genetic Programming the most appropriate evolutionary technique to be adopted. In our approach, multiplexers are controlled by direct

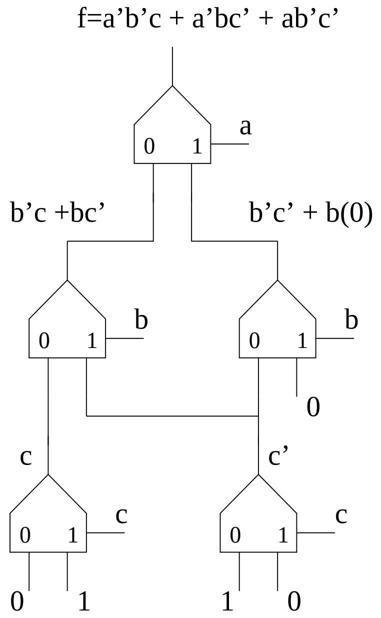

Figure 1. Shannon expansion implemented with binary multiplexers.

variables of the Boolean function, and only 1s and 0s are fed into the multiplexers (the analog of the Shannon's expansion shown in Figure 1).

4. Multiplexers as Universal Logic Elements

In this paper, the binary multiplexer is the only component used to create Boolean functions. This is a sound approach since the binary multiplexer is a basis for Boolean functions, that is, "universal generators" of Boolean functions.

A binary multiplexer (**mux**) is a logic circuit with two inputs a and b, one output f, and one control signal s, related as follows:

$$f = as + bs' \tag{1}$$

In other words, the output is the value a when the selector is "high", and b when the selector is "low".

The Shannon expansion is the representation of a Boolean function through the residues of a Boolean function.

Definition 1. Residue of a Boolean function: The residue of a Boolean function $f(x_1, x_2, \ldots, x_n)$ with respect to a variable x_j is the value of the function for a specific value of x_j. It is denoted by f_{x_j}, for $x_j = 1$ and by $f_{\bar{x}_j}$ for $x_j = 0$. The Shannon expansion in terms of residues of the function is,

$$f = \bar{x}_j f|_{\bar{x}_j} + x_j f|_{x_j} \tag{2}$$

For mapping Boolean expansions into circuits using binary multiplexers, the variable x_j in Equation 2 takes the place of the control variable s in Equation 1. For the sake of an example consider the function $f(a, b, c) = a'b'c + a'bc' + ab'c'$.

The residue of the expansion over the variable a is:

$$\begin{aligned} f(a,b,c) &= a' f|_{a=0} + a f|_{a=1} \\ &= a' \cdot (b'c + bc') + a \cdot (b'c') \end{aligned}$$

The factor $b'c + bc'$ must be taken by the mux when the selector a is "low", and $b'c'$ when a is "high". These factors could also be expanded in the same way. The expansion of the first factor over the variable b is:

$$\begin{aligned} b'c + bc' &= b'(b'c + bc')|_{b=0} + b(b'c + bc')|_{b=1} \\ &= b' \cdot c + b \cdot c' \end{aligned}$$

And the expansion of the second factor over b is:

$$\begin{aligned} b'c' &= b'(b'c')|_{b=0} + b(b'c')|_{b=1} \\ &= b' \cdot c' + b \cdot 0 \end{aligned}$$

Since in our approach the only valid inputs to the muxes are "0" and "1", the variable "c" has to be fed to the circuit through a mux. This is done by the two muxes at the bottom of the "tree". The circuit implementing the function of our example is shown in Figure 1.

5. Basic Concepts of Information Theory

Uncertainty and its measure provide the basis for developing ideas about Information Theory (Cover and Thomas, 1991). The most commonly used measure of information is Shannon's entropy.

Definition 2. Entropy: The average information supplied by a set of k symbols whose probabilities are given by $\{p_1, p_2, \ldots, p_k\}$, can be expressed as,

$$H(p_1, p_2, \ldots, p_k) = -\sum_{s=1}^{k} p_k log_2 p_k \tag{3}$$

The information shared between the transmitter and a receiver at either end of a communication channel is estimated by its Mutual Information,

$$MI(T; R) = H(T) + H(R) - H(T, R) = H(T) - H(T|R) \tag{4}$$

The conditional entropy $H(T|R)$ can be calculated through the joint probability, as follows:

$$H(T|R) = -\sum_{i=1}^{n}\sum_{j=1}^{n} p(t_i r_j) log_2 \frac{p(t_i r_j)}{p(r_j)} \tag{5}$$

An alternative expression of mutual information is

$$MI(T; R) = \sum_{t \in T}\sum_{r \in R} p(t, r) log_2 \frac{p(t, r)}{p(t)p(r)} \tag{6}$$

Mutual information, Equation 4, is the difference between the marginal entropies $H(T) + H(R)$, and the joint entropy $H(T, R)$.

We can explain it as a measure of the amount of information one random variable contains about another random variable, thus it is the reduction in the uncertainty of one random variable due to the knowledge of the other (Cover and Thomas, 1991). Conditional entropy has been used in top-down circuit minimization methods (Cheushev et al., 2001), and also in evolutionary approaches (Luba et al., 2000a; Luba et al., 2000).

Mutual information is not an invariant measure between random variables because it contains the marginal entropies. Normalized Mutual Information is a better measure of the "prediction" that one variable can do about the other (Studholme et al., 1999):

$$NMI(T; R) = \frac{H(T) + H(R)}{H(T, R)} \tag{7}$$

The joint entropy $H(T, R)$ is calculated as follows:

$$H(T, R) = -\sum_{t \in T}\sum_{r \in R} p(t, r) log_2 p(t, r) \tag{8}$$

Table I. Function $f = a'b'c + a'bc' + ab'c'$ used to compute MI(f;c).

a	b	c	f=a'b'c+a'bc'+ab'c'
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Normalized MI has been used in image registration with great success (Maes et al., 1997).

Example: We illustrate these concepts by computing the Mutual Information between two Boolean vectors f and c, shown in Table I. Variable c is an argument of the Boolean function $f(a,b,c) = a'b'c + a'bc' + ab'c'$.

We wish to estimate the description the variable c can do about variable f, that is, $MI(f;c)$.

We use Equations 4 and 5 to calculate $MI(f;c)$. Thus, we need the entropy $H(f)$ and the conditional entropy $H(f|c)$.

Entropy requires the discrete probabilities $p(F=0)$ and $p(F=1)$ which we find by counting their occurrences

$$H(f) = -(\frac{5}{8}log_2\frac{5}{8} + \frac{3}{8}log_2\frac{3}{8}) = 0.9544$$

The conditional entropy, Equation 5, uses the joint probability $p(f_i, c_j)$, which can be estimated through conditional probability, as follows: $p(f,c) = p(f)p(c|f)$. Since either vector f and c is made of two symbols, the discrete joint distribution has four entries. This is:

$$\begin{aligned}
p(f=0, c=0) &= p(f=0)p(c=0|f=0) = \frac{5}{8} \times \frac{2}{5} = 0.25 \\
p(f=0, c=1) &= p(f=0)p(c=1|f=0) = \frac{5}{8} \times \frac{3}{5} = 0.375 \\
p(f=1, c=0) &= p(f=1)p(c=0|f=1) = \frac{3}{8} \times \frac{2}{3} = 0.25 \\
p(f=1, c=1) &= p(f=1)p(c=1|f=1) = \frac{3}{8} \times \frac{1}{3} = 0.25
\end{aligned}$$

Now, we can compute the conditional entropy by using Equation 5. The double summation produces four terms:

$$
\begin{aligned}
H(f|c) &= -(\frac{1}{4}log_2\frac{1}{2} + \frac{3}{8}log_2\frac{3}{4} + \frac{1}{4}log_2\frac{1}{2} + \frac{1}{8}log_2\frac{1}{4}) \\
H(f|c) &= 0.9056
\end{aligned}
$$

Therefore, $MI(f;c) = H(f) - H(f|c) = 0.9544 - 0.9056 = 0.0488$. In fact, for the three arguments of the example function we get $MI(f;a) = MI(f;b) = MI(f;c)$. The normalized mutual information between either argument and the Boolean function is $NMI(f;a) = NMI(f;b) = NMI(f;c) = 1.0256$. Although no function argument seems to carry more information about the function f, we show later that the landscape of NMI contains a region implying information sharing. This region is hard to find on the MI landscape.

6. Genetic Programming Concepts

Genetic Programming is a variation of the genetic algorithm in which a tree-based representation is adopted. Its main aim is to solve automatic programming problems and symbolic regression problems (Koza, 1992). Problems such as the exponential growth of the search space even for some very specific problem domains, and the representation or encoding of computational structures of the objective language, remained unsolved for some years. John R. Koza (1992), used genetic algorithms to tackle the search space problem, and S-expressions which are naturally encoded as trees, to represent programs. The evolutionary operations applied over trees always produce valid trees and, therefore, syntactically correct programs. One of the earliest genetic programming systems ran in LISP, as to take advantage of the native parser provided by this programming language. Thus, the language interpreter "runs" the evolutionary algorithm, and at the same time is the evaluator of the S-expressions produced by the evolutionary algorithm.

Genetic Programming evolves functions encoded as trees. We should see a tree as the abstract semantic view of a program, that is, a parse tree. Therefore, nodes and leaves of the tree represent non-terminal and terminal grammar elements of the objective language. All together, genetic programming has to be initialized with a set of operators and functions that work as a basis for the evolutionary synthesis of programs. For example,

- Arithmetic operations (e.g., $+, -, \times, \div$)
- Mathematical functions (e.g.,sine, cosine, log, exp)
- Boolean operations (e.g., AND, OR, NOT)
- Conditionals (IF-THEN-ELSE)
- Iterators (DO-UNTIL)

are operators and functions commonly adopted in genetic programming. For the evolution of logic circuits we define a pertinent set of grammar elements (note the relationship with Figure 1):

- Terminals= $\{0, 1\}$
- Non-terminals= {a multiplexer}

Therefore, a circuit is represented as a tree using binary multiplexers as node functions, and 0s and 1s for the leaves. An illustration of this kind of tree is shown in Figure 2 (the circuit was derived using the technique described in this article). The circuit is 100% functionally equivalent to the one derived by Shannon expansions in Figure 1. Note that the circuits are also structurally similar but the muxes are controlled by different variables.

The main body of the standard Genetic Programming algorithm is the following:

Genetic Programming Algorithm
t=0;
$P_t \leftarrow$ initial population
$P_t \leftarrow fitness(P_t)$
while (stopcondition = false) {
    $S \leftarrow selection(P_t)$;
    $C \leftarrow crossover(S)$;
    $M \leftarrow mutation(C)$;
    t=t+1;
    $P_t \leftarrow fitness(M)$;
}

Several issues regarding the application of Genetic Programming as a problem solving tool for Boolean function synthesis are discussed next.

a b c	f(a,b,c)
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	0

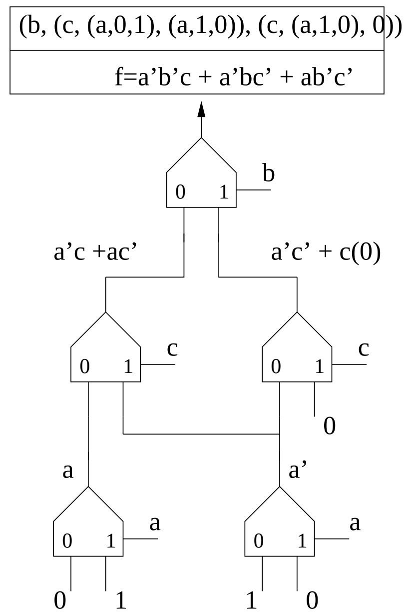

Figure 2. Logic function specification, encoding of the circuit using lists, and the circuit.

– **Implementation language:** Although the implementation language is not relevant for the results, we chose Prolog because lists are natural structures of this language and allow the representation of trees. The evaluation of either circuit just requires one predicate that translates a list into a tree.

– **Initial population:** In genetic programming, the size of the trees plays an important role in the search. In our work, we adopted the following setting for the size of the trees which was experimentally derived: maximum tree height should not exceed half the number of variables of the Boolean function. These trees could be required to be complete binary trees but our strategy was to randomly create them as to have a rich phenotypic blend in the population.

– **Representation:** A circuit is represented using binary trees, and trees are encoded as Prolog lists. A circuit tree is a recursive list of triplets of the form: $(mux, left - child, right - child)$. Mux is assigned a control variable, and either child could be a subtree or a leaf. The muxes are treated as "active high" elements, therefore the left subtree is followed when the control is 0, and the right subtree otherwise. The tree captures the essence of the circuit topology allowing only the children to feed their parent node, as shown

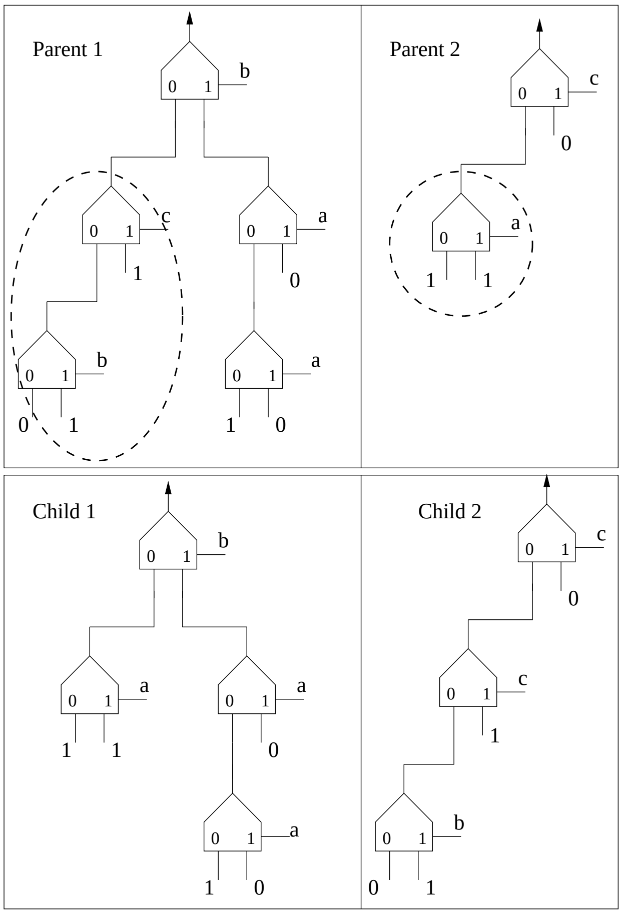

Figure 3. The crossover operator over trees encoding circuits.

in Figure 2. The representation also captures with no bias the requirement for the leaves of being only 0 or 1.

- **Crossover operator:** The exchange of genetic information between two parent trees is accomplished by exchanging subtrees as shown in Figure 3. Crossover points are randomly selected, therefore, node-node, node-leaf, and leaf-leaf exchanges are allowed since they produce correct circuits. The particular case when the root node is selected to be exchanged with a leaf is disallowed, so that invalid circuits are never generated.

  In Figure 3, two parents exchange genetic information (subtrees circled) and produce two children in the way used in this paper.

- **Mutation operator:** Mutation is a random change with very low probability of any gene of a chromosome. The mutation of a tree

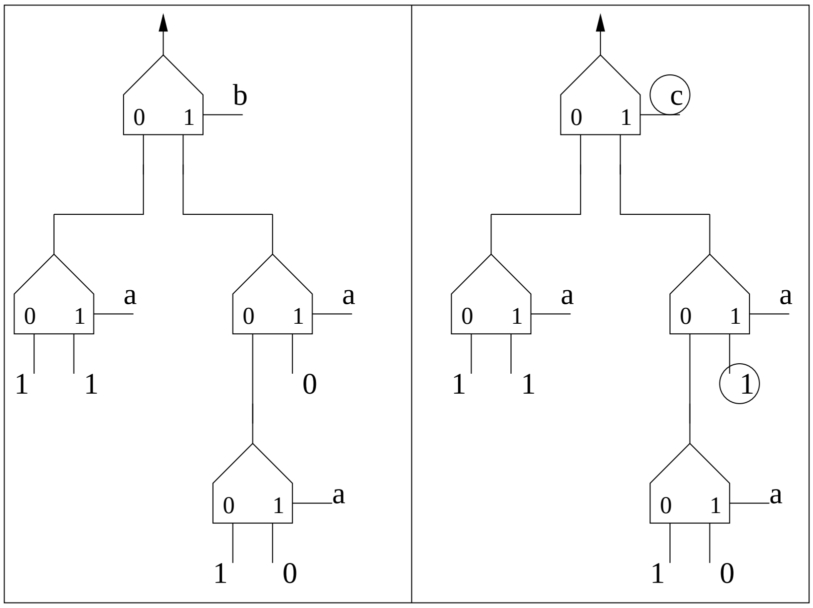

Figure 4. Mutation operation over a tree (two mutations are shown).

can alter a mux or a leaf. If a mux is chosen then a random variable is generated anew and placed as a new gene value. The mutation of a leaf is as simple as the changing of a 0 to 1, and 1 to 0. The mutation of both a node and a leaf of the tree is shown in Figure 4.

– **Fitness function:** The design of the fitness function using entropy principles is explained in Section 7. Nevertheless, every fitness function used in our experiments works in two stages since the goal is twofold: the synthesis of 100% functional circuits, and their minimization. At stage one, genetic programming explores the search space and builds improved solutions over partial solutions until it finds the first 100% functional circuit. The fitness function for this stage uses entropy concepts in order to reproduce the truth table. Once the first functional circuit appears in the population, a second fitness function is activated for measuring the fitness of the new kind of circuit. Thus, if a circuit is not 100% functional its fitness value is estimated through entropy; if the circuit is 100 % functional its fitness value denotes its size and smaller circuits are preferred over larger ones. The fitness value of a 100% functional circuit is always larger than the value of a non functional one, so that circuits are protected from fitness conflicts.

7. Entropy and Circuits

Entropy has to be carefully applied to the synthesis of Boolean functions. Let us assume any two Boolean functions, $F1$ and $F2$, and a third $F3$ which is the one's complement of F2, then $F3 \neq F2$.

$$H(F2) = H(F3)$$

Also Mutual Information shows a similar behavior.

$$MI(F1, F2) == MI(F1, F3)$$

The implications for Evolutionary Computation are important since careless use of mutual information can nullify the system's convergence. Assume the target Boolean function is T. Then, $MI(T, F2) = MI(T, F3)$, but only one of the circuits implementing $F2$ and $F3$ is close to the solution since their Boolean functions are complementary. A fitness function based on mutual information will reward both circuits with the same value, but one is better than the other. Things could get worse as evolution progresses because the mutual information increases when the circuits are closer to the solution, but in fact, two complementary circuits are then given larger rewards. The scenario is one in which the population is driven by two equally strong attractors, hence convergence is never reached.

The fitness function of that scenario is as follows. Let us assume T is the target Boolean function (must be seen as a truth table), and C is the output of any circuit in the population. Our fitness function is either the maximization of mutual information or the minimization of the conditional entropy term. This is,

$$badfitnessfunction\#1 = MI(T, C) = H(T) - H(T|C)$$

The entropy term $H(T)$ is constant since this is the expected target vector. Therefore, instead of maximizing mutual information, the fitness function can only minimize the conditional entropy,

$$badfitnessfunction\#2 = H(T|C) \tag{9}$$

We called *bad* to these fitness functions based on mutual information because we were not able to find a solution with them. Although mutual information has been described as the "common" information shared by two random processes, the search space is not amenable for evolutionary computation. In Figure 5 we show this search space over mutual information for all possible combinations with two binary strings of 8 bits (shown in decimal). The area shown corresponds to about $\frac{1}{4}$

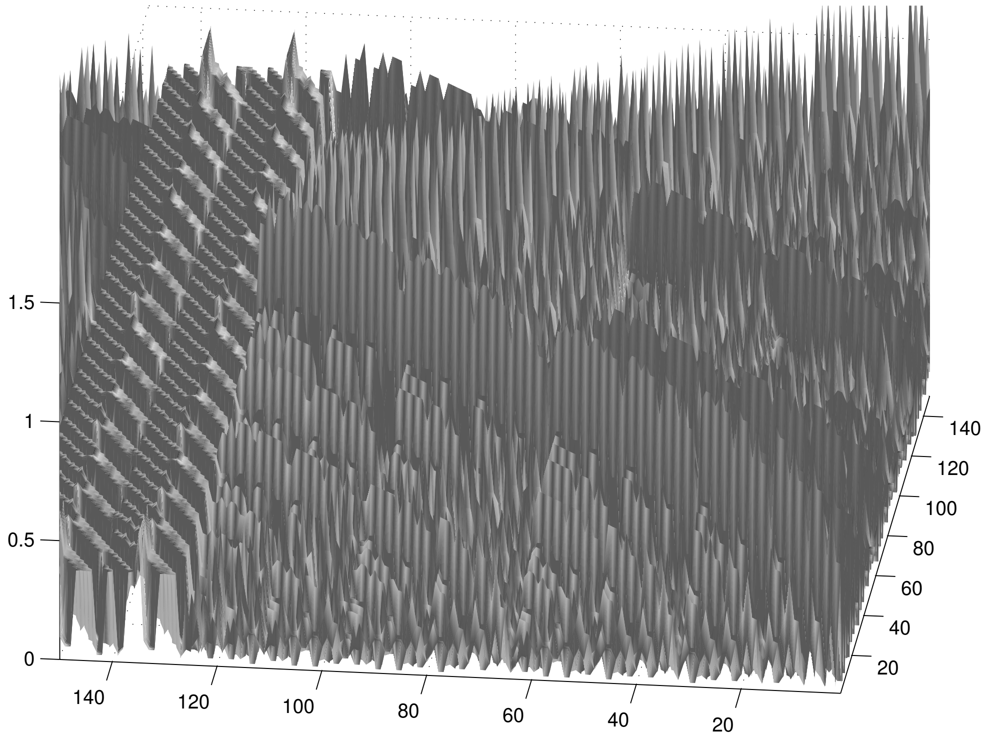

Figure 5. The search space of Mutual Information.

$([1,150] \times [1,150])$ of the whole search space of ($[1,254] \times [1,254]$) (the values 0 and 255 were not used).

The mutual information space, clearly full of spikes, does not favor the area of common information. For any two equal vectors, their Mutual Information lies on the line at 45^o (over points $\{(1,1),(2,2),(3,3) \ldots (n,n)\}$). In the next Section we continue this discussion and design fitness functions whose *landscape* seems more promising for exploration.

8. Fitness Function based on Normalized Mutual Information

So far we have described the poor scenario where the search is driven by a fitness function based on the sole mutual information. We claim that fitness functions based on Normalized Mutual Information (NMI) should improve the performance of the genetic programming algorithm because of the form of the NMI landscape. This is shown in Figure 6 for two 8-bit vectors (as in our previous case). Note on the figure how

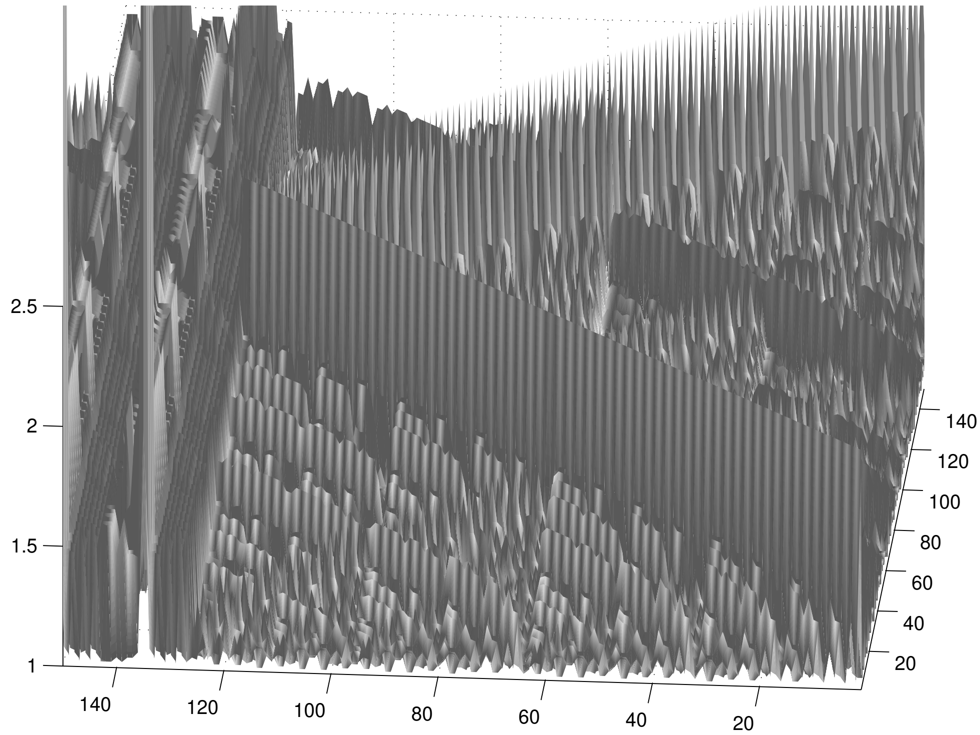

Figure 6. The search space of Normalized Mutual Information.

the search space becomes more regular and, more important yet, note the appearance of the *wall* at 45^o where both strings are equal.

We propose three new fitness functions based on Normalized Mutual Information (Equation 7) and report experiments using the next three fitness functions (higher fitness means better).

Let us assume a target Boolean function of m attributes $T(A_1, A_2, \ldots, A_m)$, and the circuit Boolean function C of the same size. In the following, we propose variations of the basic fitness function of Equation 10, and discuss the intuitive idea of their (expected) behavior.

$$fitness = (Length(T) - Hamming(T, C)) \times NMI(T, C) \qquad (10)$$

We tested Equation 10 in the synthesis of several problems and the results were quite encouraging. Thus, based on this primary equation we designed the following fitness functions. In Figure 7 we show the *fitness landscape* of Equation 10.

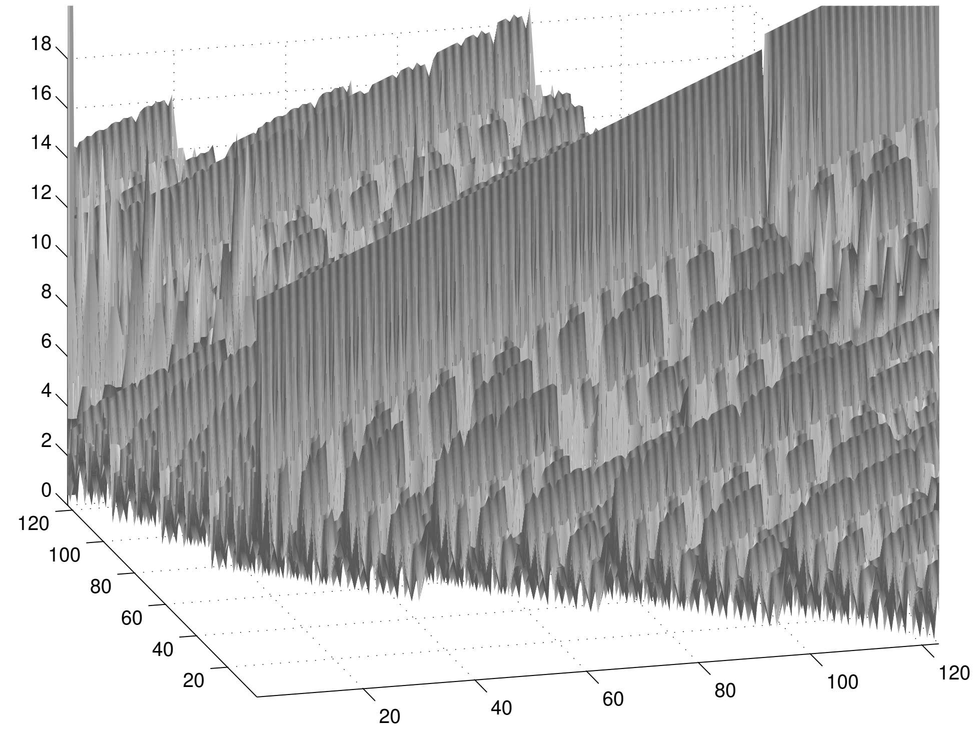

Figure 7. Fitness landscape of: $f = (Length(T) - Hamming(T,C)) \times NMI(T,C)$.

$$fitness1 = \sum_{i=1}^{m} \frac{fitness}{NMI(A_i,C)} \quad (11)$$

$$fitness2 = \sum_{i=1}^{m} fitness \times NMI(A_i,C) \quad (12)$$

$$fitness3 = (Length(T) - Hamming(T,C)) \times (10 - H(T|C)) \quad (13)$$

The function fitness, Equation 10, is driven by NMI(T,C) and adjusted by the factor $Length(T) - Hamming(T,C)$. This factor tends to zero when T and C are far in Hamming distance, and tends to $Length(T)$ when T and C are close in Hamming distance. The effect of the term is to give the correct rewarding of the NMI to a circuit C close to T. Equation 10 is designed to remove the convergence problems described in the previous section.

Table II. Generation number where the first 100% functional circuit is found, and the generation where the optimum is found, for our three proposed fitness functions.

Event	**Gen. at fitness1**	**Gen. at fitness2**	**Gen. at fitness3**
100% Functional	13 ± 5	14 ± 7	18 ± 6
Optimum Solution	30 ± 7	30 ± 10	40 ± 20

$Fitness1$ and $Fitness2$, Equations 11 and 12, combine NMI of T and C with NMI of C and the attributes A_k of the target function. Thus, $fitness1$ and $fitness2$ pretend to use more information available in the truth table in order to guide the search. $Fitness3$ is based on conditional entropy and it uses the mentioned factor to supress the reproduction of undesirable trees. Since conditional entropy has to be minimized we use the factor $10 - H(T|C)$ in order to maximize fitness. Equations 11 and 9 use the conditional entropy term. Nevertheless, only Equation 11 works fine. As a preliminary discussion regarding the design of the fitness function, the noticeable difference is the use of Hamming distance to guide the search towards the aforementioned *optimum wall* of the search space. The Hamming distance destroys elements of the population on one side of the wall, and favors the other side. Thus, there is only one attractor in the search space.

9. Experiments

In the following experiments we find and contrast the convergence of the genetic programming system for the three fitness functions of Equations 11,12,13.

9.1. Experiment 1

Here we design the following (simple) Boolean function:

$$F(a, b, c, d) = \sum(0, 1, 2, 3, 4, 6, 8, 9, 12) = 1$$

We use a population size of 300 individuals, a crossover rate (p_c) of 0.35, a mutation rate (p_m) of 0.65, and a maximum of 100 generations. The optimal solution has 6 nodes, thus we find the generation in which the first 100% functional solution appears, and the generation number where the optimal is found. The problem was solved 20 times for each fitness function. Table II shows the results of these experiments.

Table III. Generation number where the first 100% functional circuit is found, and the generation where the optimum is found, for our three proposed fitness functions.

Event	**Gen. at fitness1**	**Gen. at fitness2**	**Gen. at fitness3**
100% Functional	39 ± 12	40 ± 11	50 ± 12
Optimum Solution	160 ± 15	167 ± 15	170 ± 20

9.2. Experiment 2

Our second test function is:

$$F(a,b,c,d,e,f) = ab + cd + ef$$

In this case, we adopted a population size of 600 individuals, $p_c = 0.35$, $p_m = 0.65$, and a maximum of 200 generations. The optimal solutions has 14 nodes. Each problem was solved 20 times for each fitnesss function. Table III shows the results of these experiments.

A solution found to this problem is shown in Figure 8. We contrast the evolutionary solution against the optimum found by Reduced Order Binary Decision Diagrams.

9.3. Experiment 3

Our third (and last) problem is related with partially specified Boolean functions (Hernández Aguirre et al., 2000). With this experiment we address the ability of the system to design Boolean functions with "large" numbers of arguments and specific topology. For this, we have designed a synthetic problem where the topology is preserved when the number of variables increases.

Boolean functions with $2k$ variables are implemented with $(2*2k)-1$ binary muxes *if* the truth table is specified as shown in Table IV.

We ran experiments for $k = 2, 3, 4$, thus 4,8, and 16 variables and we contrasted these results with the best known so far for this problem (reported in (Hernández Aguirre et al., 2000)). For completeness, all previous results are reported together with the results of the new experiments in Table V, where we use the three fitness functions proposed before (Equations 11,12,13).

All parameters are kept with no changes for similar experiments, average is computed for 20 independent runs. In previous work we used a fitness function based on the sole Hamming distance between the current solution of an individual and the target solution of the truth table (Hernández Aguirre et al., 2000). One important difference is the

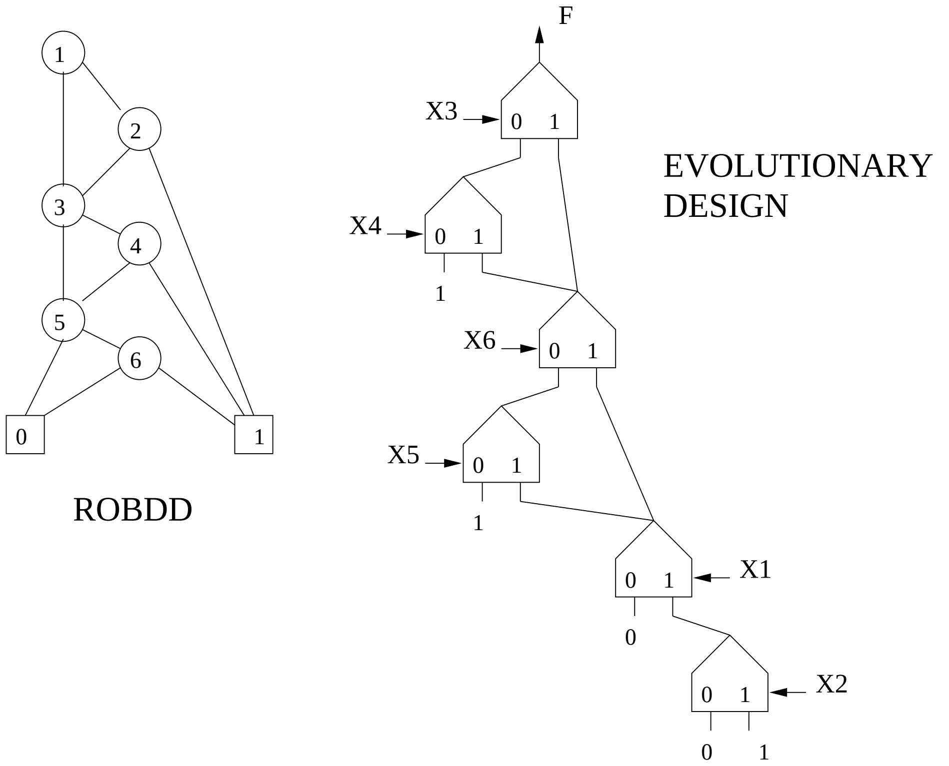

Figure 8. Solution found by Genetic Programming to Experiment 2

Table IV. Partially specified Function of Example 3 needs $(2*2k)-1$ muxes.

ABCD	**F(ABCD)**
0 0 0 0	0
0 0 0 1	1
0 0 1 0	1
0 1 0 0	1
1 0 0 0	1
0 1 1 1	1
1 0 1 1	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	0

Table V. Generation number where the first 100% functional circuit is found, and the generation where the optimum is found, for the three proposed fitness functions.

k	variables	size	Avg(previous)	Avg(fitness1)	Avg(fitness2)	Avg(fitness3)
2	4	7	60	60	60	60
3	8	15	200	190	195	194
4	16	31	700	740	731	748
5	32	63	2000	2150	2138	2150

percentage of correct solutions found. Previously, we reported that in 90% of the runs we found the solution (for the case of fitness based on Hamming distance). For the three fitness functions based on entropy we found the solution in 99% of the runs.

10. Final Remarks and Conclusions

We have tested in this paper a fitness function using only conditional entropy for circuit synthesis, with no success at all. We believe this is a clear indication of a fitness function that does not take into account entropy properties. Therefore, the evolutionary algorithm cannot converge because there is more than one attractor in the search space. Figure 5 reveals an amorphous search MI landscape with a quite weak wall at 45^o. The left hand side of the wall seems more regular than the right hand side. Although it is hard to derive any conclusions from this figure, it is clear that no attractor dominates the area and it could explain the failure of the fitness function based on MI only.

The landscape of Normalized Mutual Information seems less chaotic and more regular. The great advantage of a fitness function designed over NMI is the appearance of the wall at 45^o. It is clear that the wall must appear when the random vectors are equal; as the intersection of the vectors increases so it does the MU. What we have shown in this paper is that, in spite of all the credit given to MI as the "real information" shared between two random processes, the NMI landscape is more suitable for searching than the MI landscape. In the landscape of the fitness function of Figure 7, we can see the wall due to equal vectors is preserved, so we believe it is part of the landscape of three fitness functions using Equation 10.

In general, the three fitness functions proposed in this paper worked quite well. All of them found the optimum in most cases, thus they are comparable to other fitness functions based on Hamming distance (Hernández Aguirre et al., 2000).

A final remark goes to the convergence time and quality of results that are comparable to those previously reported in the specialized literature. From Tables II and III we can give some advantage to normalized mutual information over simple mutual information because it is less biased. Results from Table V could imply that mutual information is able to capture "that" relationship between the data that the sole Hamming distance can not convey to the population.

11. Future Work

There are two immediate lines for further research in which we are already working. These are the following:

1. **Use of entropy-based measures to estimate the complexity of a tree:** In the experiments of this article, we reported the generation at which the first 100% functional solution was found. We also counted the number of muxes in that first functional solution (not reported here), and also the size of the best tree produced along all the evolutionary process. Comparing the size of trees for fitness functions based and not based on entropy, we found shorter trees when using entropy (NMI to be precise). A detailed analysis is in process in order to explain this behavior. It is quite important because shorter trees usually denote shorter convergence time to the solution. Thus, our hypothesis is that entropy could improve the measure of the complexity of a tree and that it could help to relate tree size with target goal. So far, the experiments show that trees tend to grow but evolution keeps their entropy low and with about the same value. Certainly, many different trees can share the same entropy value, but if this is low then it could mean some trees are replicating a branch with low probability of success. Entropy could also be used to keep diversity high. This is of major importance during the exploration phase of the evolutionary algorithm and, therefore, deserves to be explored in more depth.

2. **Circuits with more than one output:** The generation of common or shared branches is the main issue in this type of circuits. Our initial approach, has consisted on measuring the mutual information between the output vectors, and to relate the circuit outputs to the argument vectors of the Boolean function. Our preliminary results are encouraging but more work is still necessary in this direction

Acknowledgements

The first author acknowledges partial support from CONACyT project No. I-39324-A. The second author acknowledges support from CONACyT through project no. 32999-A.

References

Cheushev, V.; C. Moraga, S. Yanushkevich, V. Shmerko, and J. Kolodziejczyk. Information theory method for flexible network synthesis. In *Proceedings of the IEEE 31st. International Symposium on Multiple-Valued Logic*, pages 201–206. IEEE Press, 2001.

Coello Coello, Carlos A.; Alan D. Christiansen, and Arturo Hernández Aguirre. Use of evolutionary techniques to automate the design of combinational logic circuits. *International Journal of Smart Engineering System Design*, 2:299–314, 2000.

Coello Coello, Carlos A.; Alan D. Christiansen, and Arturo Hernández Aguirre. Towards automated evolutionary design of combinational circuits. *Computers and Electrical Engineering*, 27:1–28, 2001.

Coello Coello, Carlos A.; Alan D. Christiansen, and Arturo Hernández Aguirre. Using genetic algorithms to design combinational logic circuits. In Cihan H. Dagli, Metin Akay, C. L. Philip Chen, Benito R. Farnández, and Joydeep Ghosh, editors, *Intelligent Engineering Systems Through Artificial Neural Networks. Volume 6. Fuzzy Logic and Evolutionary Programming*, pages 391–396. ASME Press, St. Louis, Missouri, USA, nov 1996.

Coello Coello, Carlos A.; 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.

Cover, T.M. and J.A. Thomas. *Elements of Information Theory.* John Wiley & Sons, New York, 1991.

Hartmann, C.R.P.; P.K. Varshney, K.G. Mehrotra, and C.L. Gerberich. Application of information theory to the construction of efficient decision trees. *IEEE Transactions on Information Theory*, 28(5):565–577, 1982.

Hernández Aguirre, Arturo; Bill P. Buckles, and Carlos A. Coello Coello. Ga-based learning of $kdnf_n^s$ boolean formulas. In *Evolvable Systems: From Biology to Hardware*, pages 279–290, Tokyo, Japan, 3-5 October 2001. Springer Verlag.

Hernández Aguirre, Arturo; Bill P. Buckles, and Carlos A. Coello Coello. Evolutionary synthesis of logic functions using multiplexers. In C. Dagli, A.L. Buczak, and et al., editors, *Proceedings of the 10th Conference Smart Engineering System Design*, pages 311–315, New York, 2000. ASME Press.

Hernández Aguirre, Arturo; Bill P. Buckles, and Antonio Martínez Alcántara. The pac population size of a genetic algorithm. In *Twelfth International Conference on Tools with Artificial Intelligence*, pages 199–202, Vancouver British Columbia, Canada, 13-15 November 2000a. IEEE Computer Society.

Hernández Aguirre, Arturo; Bill P. Buckles, and Carlos A. Coello Coello. Gate-level Synthesis of Boolean Functions using Binary Multiplexers and Genetic Programming. In *Conference on Evolutionary Computation 2000*, pages 675–682. IEEE Computer Society, 2000b.

Hernández Aguirre, Arturo; Carlos A. Coello Coello, and Bill P. Buckles. A genetic programming approach to logic function synthesis by means of multiplexers. In Adrian Stoica, Didier Keymeulen, and Jason Lohn, editors, *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*, pages 46–53, Los Alamitos, California, 1999. IEEE Computer Society.

Iba, Hitoshi; 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 259–276. Springer-Verlag, Berlin, 1997.

Kabakcioglu, A.M.; P.K. Varshney, and C.R.P. Hartmann. Application of information theory to switching function minimization. *IEE Proceedings, Part E*, 137:387–393, 1990.

Koza, John R. *Genetic Programming. On the Programming of Computers by means of Natural Selection.* MIT Press, Massachusetts, USA, 1992.

Lloris, A.; J.F. Gomez-Lopera, and R. Roman-Roldan. Using decision trees for the minimization of multiple-valued functions. *International Journal of Electronics*, 75(6):1035–1041, 1993.

Louis, Sushil J. *Genetic Algorithms as a Computational Tool for Design.* PhD thesis, Indiana University, Indiana, USA, 1993.

Louis, Sushil J. 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.

Luba, T.; C. Moraga, S. Yanushkevich, V. Shmerko, and J. Kolodziejczyk Application of design style in evolutionary multi-level network synthesis. In *Proceedings of the 26th EUROMICRO Conference Informatics:Inventing the Future*, pages 156–163. IEEE Press, 2000.

Luba, T.; C. Moraga, S. Yanushkevich, M. Opoka and V. Shmerko. Evolutionary multi-level network synthesis in given design style. In *Proceedings of the 30th IEEE International Symposium on Multiple valued Logic*, pages 253–258. IEEE Press, 2000a.

Maes, Frederik; André Collignon, Dirk Vandermeulen, Guy Marchal, and Paul Suetens. Multimodality image registration by maximization of mutual information. *IEEE Transactions on Medical Imaging*, 16(2):187–198, April 1997.

Miller, J.F.; 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.

Miller, Julian F.; 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.

Miller, Julian F.; Dominic Job, and Vesselin K. Vassilev. Principles in the Evolutionary Design of Digital Circuits—Part II. *Genetic Programming and Evolvable Machines*, 1(3):259–288, July 2000a.

Quinlan, J.R. Learning efficient classification procedures and their application to chess games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 463–482. Springer, Berlin, Heidelberg, 1983.

Shannon, Claude E.. A Mathematical Theory of Information. *Bell System Technical Journal*, 27:379–423, July 1948.

Studholme, C.; D.L.G. Hill, and D.J. Hawkes. An overlap invariant entropy measure of 3D medical image alignment. *Pattern Recognition*, 32:71–86, 1999.

Thompson, Adrian; Paul Layzell, and Riccardo Salem Zebulum. Explorations in Design Space: Unconventional Design Through Artificial Evolution. *IEEE Transactions on Evolutionary Computation*, 3(3):167–196, September 1999.

Weaver, W. and C. E. Shannon. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Illinois, 1949.