What Can Boolean Networks Learn ?

Arlindo L. Oliveira
Alberto Sangiovanni-Vincentelli
Dept. of EECS, UC Berkeley, Berkeley CA 94720

June 26, 1992

Abstract

We study the generalization abilities of networks that are composed of boolean nodes, i.e., nodes that implement only basic boolean functions: *and, or* and *not*.

The majority of the network learning algorithms proposed so far generate networks where each node implements a threshold function and are inappropriate for the generation of boolean networks from training set data. We propose an algorithm that, given a training set, generates a boolean network of small complexity that is compatible with the training set. The algorithm, inspired in techniques used in the logic synthesis community for the design of VLSI circuits, generates both the connectivity pattern and the architecture of the network. Furthermore, the resulting network can be implemented in silicon in a straightforward way.

Experimental results obtained in a set of problems from the machine learning literature show that the generalization performed by boolean networks synthesized with this algorithm compares favorably with the generalization obtained by alternative learning algorithms. Some of these results and examples of the layout of networks obtained using the algorithm are presented and discussed.

1 Introduction

We address the problem of supervised single concept learning in an attribute based description language. In this setting, a concept $\mathcal{C}$ is a subset of the input space, $\mathcal{U}$. From a set of examples extracted from S according to some distribution and labeled positive or negative according to whether they belong or not to $\mathcal{C}$, the learning algorithm outputs an hypothesis, H, that is an approximation as good as possible to the target concept, $\mathcal{C}$. This hypothesis is then used to classify other examples that may or may not have been presented in the training phase.

The focus of our research is to study whether or not boolean networks can be used to perform the classification task described above. For this particular purpose, we define boolean networks as networks where each node computes a primitive boolean function (*and, or, not*. Evaluating the power of boolean networks in classification tasks is important for a variety of reasons. In the first place, it is important, from a theoretical point of view, to study the situations in which networks composed of threshold gates (the ones most commonly considered in the neural networks literature) are more powerful than boolean networks. If boolean networks prove to be adequate for some learning tasks this may have great practical impact because, unlike threshold gate networks, the hardware implementation of boolean networks using digital circuit design techniques is easy and inexpensive.

We consider the case where the attributes are binary valued or can be encoded in some binary valued code and the domain is noise free. In section 3 we propose an algorithm that generates boolean networks compatible with the data present in the training set. From the many boolean networks that are compatible with this data, we search for one that has small complexity, as measured by the number of literals in a multi-level implementation[1].

This approach is equivalent to the use of an Occam's razor bias and can be justified in a more formal way by using the concept of capacity of a family of concepts. More specifically, assume the concept to be learned corresponds to a boolean function in some set of functions, S_N, and that $S_1, S_2 ... S_{N-1}$ can be defined obeying $S_1 \subseteq S_2 \subseteq ... S_{N-1} \subseteq S_N$. Theoretical results [3] have shown that if the learner outputs one function f_T that is compatible with the training set, then the generalization accuracy is a decreasing function of the VC-dimension of S_k where k is the smallest integer such that $f_T \in S_k$[2]. Since the definition implies that VC-dim$(S_j) \leq$ VC-dim(S_{j+1}) if $S_j \subseteq S_{j+1}$ the learner should always output a function compatible with the training set that is contained in S_k for k as small as possible. In our setting, it is natural to take S_k as the set of boolean functions implementable by a boolean

[1]The number of literals is a convenient measure of the complexity of a boolean network and is well correlated with the size of its physical implementation. It is simply the sum of the number of inputs for all the primitive logic gates in the network.

[2]The VC-dimension of a set of functions S_k is simply the maximum integer m such that functions in S_k can induce 2^m different dichotomies in any set of m examples taken from $\mathcal{U}$.

network with no more that k literals. A similar argument for the preference for networks with few literals could be made using the Minimal Description Length Principle (MDLP) of Rissanen [15].

Given that the objective is to generate a boolean network with as few literals as possible, it would seem that general purpose logic synthesis techniques could be used to synthesize a compact network given the data in the training set. Regrettably, this is not the case. Classical logic synthesis algorithms can be divided in two types: synthesis of two-level networks and synthesis of multi-level networks. Many algorithms for the synthesis of minimal two-level networks (i.e., and-or networks) have been proposed [4, 10] and they could, in principle, be used to generate minimal networks given the data in the training set. In this process, they are able to use the extra freedom allowed by the fact that not all input combinations are present in the training set to obtain compact implementations.[3] However, efficiency problems related with the particular form in which the data is available in learning problems makes it unfeasible to use them in all but the smallest of the problems. Multi-level synthesis algorithms [6, 2] are even less efficient and much worst at using the extra degrees of freedom allowed by missing input combinations.

2 Definitions

Let $\mathcal{B} = \{0, 1\}$. A completely specified Boolean function, f, is a mapping from $B^n \rightarrow \mathcal{B}$. An uncompletely specified boolean function is a partial mapping from $B^n \rightarrow \mathcal{B}$. In both cases, the mapping specifies which points of $\mathcal{B}^n$ are mapped to 1 (the on-set of f, f^{ON}) and which are mapped to 0 (the off-set of f,f^{OFF}). For uncompletely specified functions, points of $\mathcal{B}^n$ not specified in the mapping are said to belong to the don't-care set of f. A completely specified function g is a valid implementation of an incompletely specified function, f, iff $f^{OFF} \subseteq g^{OFF}$ and $f^{ON} \subseteq g^{ON}$.

Let $\{x_1, ...x_n\}$ be the input variables and $\{y_1, ...y_m\}$ be intermediate variables. A literal is either a variable or its negation. A cube is a conjunction of literals where no two literals corresponding to the same variable appear. A point of $\mathcal{B}^n$ is a minterm and it corresponds to a cube with n literals. A disjoint normal form (DNF) for f is a representation for f as a sum of cubes[4]. The support of a function is the set of variables that are explicit inputs to that function.

Let f be defined by $(f^{ON}, f^{OFF}, f^{DC})$, the on-set, off-set and don't care set respectively. In this setting, f^{ON} and f^{OFF} are specifically given as lists of minterms, the positive

[3] It is in this process that generalization to unseen parts of the input space actually takes place. By choosing value of the output in a point not present in the learning set the minimization algorithm is performing induction.

[4] We will generally denote disjunction by the $+$ sign and omit the conjunction sign when describing boolean expressions.

and negative examples. Let $e_i^+ \in f^{ON}$ be the ith minterm in f^{ON} and e_j^- the jth element in f^{OFF}. Finally let $f^C = f^{ON} \cup f^{OFF} = \overline{f^{DC}}$ and let e_k be the kth element of f^C. We assume there is no noise and therefore $f^{OFF} \cap f^{ON} = \emptyset$.

A Boolean network is represented by a directed acyclic graph (DAG). Associated with each node of the graph is a variable, y_i and a representation of a logic function, f_i such that $y_i = f_i$. There is a directed edge e_{ij} from y_i to y_j if f_j depends explicitly on y_i or $\overline{y_i}$. y_i is a fanout node of y_j if there exists a directed edge e_{ji} and a fanin node if there is a directed edge e_{ij}.

3 The synthesis algorithm

The algorithm presented here uses some ideas presented in [13] but uses a very different approach for hidden node selection. The approach proposed incrementally selects a set of hidden unit functions such that the implementation of a function compatible with f over the new variables defined by these functions is simpler. For that, we start by defining the conditions under which a set of functions can be used as a new support for the implementation of function f.

3.1 Selection of hidden node functions

Consider a boolean function $f(x_1, ..., x_n)$ and the directed acyclic graph that corresponds to some valid implementation of f. A cutset of this graph is a set of nodes that, if removed, leaves no path from the input nodes to the output node. A set of nodes $R = \{y_1, ..., y_k\}$ defined by the functions in $V = \{f_1, ..., f_k\}$ can be a cutset for the graph corresponding to **some** valid implementation of f iff f can be implemented as $f(x_1, ..., x_n) = g(h(x_1, ..., x_n))$ where $g : \mathcal{B}^k \rightarrow \mathcal{B}$ and $h : \mathcal{B}^n \rightarrow \mathcal{B}^k$ is the function that transforms $\{x_1, ..., x_n\}$ into $(y_1 = f_1(x_1, ..., x_n), ..., y_k = f_k(x_1, ..., x_n))$. A necessary and sufficient condition for R to be a cutset is that $H(f^{ON}) \cap H(f^{OFF}) = \emptyset$ where $H(X)$ is the image of X by h.

Finding h such that $H(f^{ON}) \cap H(f^{OFF}) = \emptyset$ can be done using different techniques. Since the minterms in the on and off set of f can be explicitly listed the following approach is used: consider a matrix $\mathcal{O}$ where each column corresponds to an element of f^{ON} and each row corresponds to an element of f^{OFF}. To every element of $\mathcal{O}$ in row r and column c, $\mathcal{O}_{rc}$ we associate an f^{ON} minterm e_c^+ and an f^{OFF} minterm e_r^-.

Consider now a boolean function f_i. This function will have the value 1 for some points in f^C and the value 0 for the remaining ones. Let $f_i^+ \subset f^C$ be the set of minterms in f^C that cause function f_i to have the value 1 and $f_i^- = f^C \setminus f_i^+$ the set of minterms in f^C that cause function f_i to have the value 0.

Given a f^{ON} minterm (e_c^+) and a f^{OFF} one (e_r^-), the value of node y_i can be used to distinguish between them iff one of the following two conditions holds:

a) $e_c^+ \in f_i^+ \wedge e_r^- \in f_i^-$

b) $e_c^+ \in f_i^- \wedge e_r^- \in f_i^+$.

In the first case, function f_i assumes value 1 for minterm e_c^+ and 0 for e_r^- while in the second the opposite is true.

We will say that function f_i covers element $\mathcal{O}_{rc}$ of $\mathcal{O}$ if f_i has different values for e_c^+ and e_r^-, i.e., if either a) or b) above holds. A given function will, in general, cover several elements of $\mathcal{O}$. In particular f_i will cover all elements in the columns that correspond to minterms in $f^{ON} \cap f_i^+$ and the rows that correspond to minterms in $f^{OFF} \cap f_i^-$. It will also cover all elements in the columns $f^{ON} \cap f_i^-$ and rows $f^{OFF} \cap f_i^+$. If the order of the rows and columns is properly rearranged, the set of elements in $\mathcal{O}$ that are covered looks like two rectangles with the corners touching. Figure 1 shows the elements of the $\mathcal{O}$ matrix covered by function f_i.

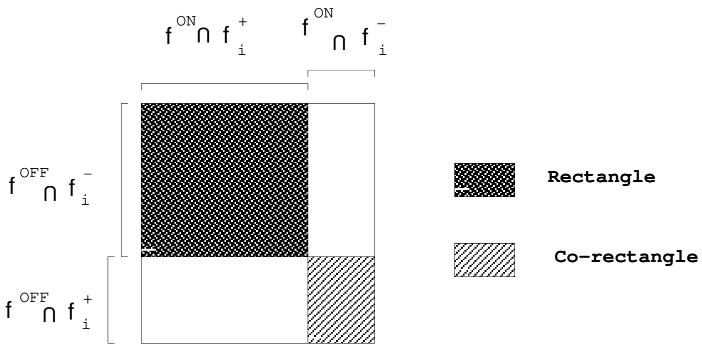

Figure 1: Elements of $\mathcal{O}$ covered by function f_i

A set of functions $V = \{f_1, f_2..., f_k\}$ is a cover for $\mathcal{O}$ if all elements in the $\mathcal{O}$ matrix are covered by at least one function in V. In this case, $h(x_1, ...x_n) = (f_1, ..., f_k)$ satisfies $H(f^{ON}) \cap H(f^{OFF}) = \emptyset$, the nodes $y_i = f_i$ are a cutset for a graph of some realization of f and the variables $y_i = f_i$ can be used as a support for implementing function f. The cover $V = \{l_1, l_2..., l_n\}$ where $l_i = x_i$ or $l_i = \overline{x_i}$ for $i = 1, ..., n$ is called a trivial cover.

The choice of an appropriate set of hidden unit functions can now be put as follows: **From W, the set of candidate functions, select a set V such that:**

1. **V covers $\mathcal{O}$.**
2. **The cost of implementing the functions in V plus the cost of implementing f as $f = g(y_1, ..., y_k)$ is minimal.**

Assuming the cost of each f_i is easy to compute, the problem with this criterion is that the evaluation of the cost of implementing f as $g(y_1, ..., y_k)$ is difficult to compute.

The heuristic approach we take is based on the assumption that functions realized over a smaller number of variables are easier to implement.

Therefore, we use the following heuristic: **select V such that $|V|$ is minimal**. This algorithm will lead, in some cases, to sub-optimal solutions, but we found that, for many functions of interest, it leads to compact implementations. This approach is reasonable if the functions $f_1, f_2..., f_m$ are easy to implement and of comparable complexity. In our approach we consider as candidate functions only functions that can be represented as conjunctions of existing literals. However, even with this restriction, there are still too many potential candidates. The next section describes how a restricted set of candidate cubes is generated.

3.2 Selection of candidate cubes

The selection of an appropriate set of candidate cubes, W, can be done in several different ways. However, no efficient algorithm will generate all interesting cubes without generating many others that are irrelevant for the problem at hand.

The algorithm we use for the selection of useful cubes is based on the concept of algebraic divisor of an expression. Let E_f be a DNF expression for f. A cube c is a divisor of E_f if E_f can be expressed as $E_f = E_g c + E_r$ when all the operations involved are performed algebraically and E_g is not null. A cube c is a maximal cube divisor of E_f if E_g cannot be reexpressed as $E_g = c' E_h$ where c' is a cube, i.e., if E_g is cube free[5].

The following example illustrates these concepts: let $E_f = abc + ab\overline{d} + \overline{b}c$. The maximal cube divisors are $ab, abc, ab\overline{d}, c, \overline{b}c$. An alternative but equivalent way to define maximal cube divisors is to consider the cube/literal matrix: a matrix where each column corresponds to a literal in the expression and each line corresponds to a cube. An entry in row r, column c is 1 if the cube that corresponds to row r has the literal that corresponds to column c. Figure 2 shows the cube/literal matrix for E_f.

In the cube/literal matrix, a rectangle is a set of columns and rows such that all intersections have a 1. The maximal cube divisors are given by the conjunction of the literals in the columns that correspond to maximal rectangles, i.e., rectangles that are not properly contained in any other rectangle. This concept is closely related with the concept of a co-kernel, introduced in [5],

Define $\overline{f} = (f^{OFF}, f^{ON}, f^{DC})$ and let $f = (f^{ON}, f^{OFF}, f^{DC})$ be the function we want to reexpress as $g(h(..))$.. The cubes that we select as candidates are:

1. Cubes that are maximal cube divisors of E_f where E_f is an expression of a function compatible with f obtained by two-level minimization.

[5]For this particular purpose we assume that E_g is cube free if $E_g = 1$.

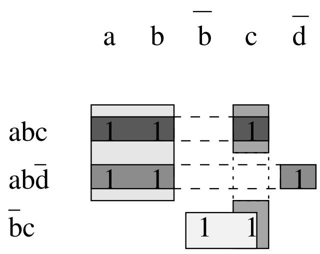

Figure 2: Cube/literal matrix for $E_f = abc + ab\overline{d} + \overline{b}c$.

2. Cubes that are maximal cube divisors of $E_{\overline{f}}$ where $E_{\overline{f}}$ is an expression for a function compatible with $\overline{f}$ obtained by two-level minimization.

The two-level minimization required for this step obtains a DNF expression for f with few cubes and is performed by the algorithm described in [12], although any two-level minimizer like ESPRESSO [4] could, in principle, be used. However, the algorithm described in [12] was developed specifically for two-level minimization in situations where the function can be described by two sets of minterms and is much more efficient than general purpose two-level minimizers.

3.3 Solving the set covering problem

Selecting a set $V \subseteq W$ of minimal cardinality that covers $\mathcal{O}$ is a difficult problem. More precisely, let M be a matrix with as many rows as elements of $\mathcal{O}$ ($|f^{ON}| \times |f^{OFF}|$ rows) and as many columns as candidate cubes. Element M_{ij} of M is 1 if the jth cube covers the ith element of $\mathcal{O}$. We want to select a set of columns of minimal cardinality such that every row has at least a 1 in one of the selected columns. This problem is NP-complete [9] and we need to solve an instance of high dimension. Therefore, an exact solution is hopelessly costly, in most cases. Several algorithms have been proposed for obtaining a good solution for this problem. For example, the covering algorithm used in ESPRESSO [4] is able to find an approximate solution for fairly large sized problems in reasonable time. However, one of the steps of this algorithm is to find a maximal independent set of rows and use this set to compute a lower bound in the size of V. For our case, where the number of rows can exceed 100000 this approach is unfeasible. For experimental purposes, we used a simple branch and bound algorithm approach that uses the fact that all columns have the same cost to bound the search tree in an efficient way.

This approach is successful in solving exactly the smaller problems but takes an inordinate amount of time for the larger ones. In some cases, it actually finds the absolute minimum solution in a reasonable time but takes a long time to prove that the solution is optimal. To solve this problem the search terminates when a pre-determined amount of time is elapsed and the algorithm returns the best solution found so far.

3.4 Outer loop

The algorithm starts with a network with two levels. The function of the nodes in the first level is obtained by listing the conjunctions that correspond to the positive examples. The output node is a disjunction of the variables computed by the first level nodes. By construction, this network will be correct for all the examples in the training set.

We then select set of candidate cubes defined over the input variables, W, and, from this set, a minimal non-trivial cover $V = \{f_1, f_2, ..., f_k\} \subseteq W$. After creating a node in the network for each $y_i \in V$, function f is then expressed as a function of $y_1..., y_k$ by a procedure similar to the one performed to build the first network.

This process is iterated until either all nodes are single cubes, no valid cover can be selected from the available candidates or further transformations lead to a deterioration of the size of the network.

4 Results

To test the performance of the algorithm we used the following set of 8 test concepts. *Carry4* is the value of the carry bit output when two 4 bit integers are added. *Xor8* is the exclusive or of 8 inputs. The next three concepts accept a compact multi-level representation (as do the previous two) and are listed in the appendix. Finally, the last three have been selected from the UCI machine learning database [11] and they satisfy the following criterion: two class classification with discrete valued input attributes. Discrete multi-valued attributes were encoded using a one-hot encoding.

The performance of the algorithm (MIFES) was compared with the performance of two alternative machine learning algorithms: a reimplementation of ID3 [14] and the cascade-correlation algorithm [8].

For each problem, a set of examples was created and 5 independent runs were performed. In each run a fixed fraction of the examples was selected to be in the training set and the resulting classification rule (boolean network, decision tree or threshold gate network) was tested in the complete set (including the examples present in the training set). The same training data and validation procedure was used for each of the methods. The size of the training sets used to learn each of the concepts was fixed either according to a simple empirical rule or set equal to values previously used in other experiments [16].

While it is true that previous bounds on the sample complexity of learning could, in principle, have been used to set these values, we found it hard to use them given the looseness of the bounds known to the authors for the number of different boolean functions implementable by a network of a given complexity, r.

For example, the expression derived in [1], $m > \frac{1}{\epsilon}(\ln(r) + \ln(\frac{1}{\delta}))$ and the best bounds presented in [7] for r lead to a minimum number of 1360 examples to PAC-learn the Xor8 concept with $\epsilon = \delta = 0.1$.

Table 1 lists the results obtained. The second, third and fourth column list the number of input variables, the average size of the training sets and the number of examples available. The last 3 columns show the average accuracy for each of the methods.

Concept	# inputs	Training	Testing	MIFES	ID3	CasCor
carry4	8	134	256	96.3	95.2	96.5
xor8	8	134	256	100.0	61.1	90.4
sm12	12	327	4000	100.0	88.9	84.7
sm18	18	605	4000	78.9	84.1	78.9
str18	18	605	4000	94.9	87.9	86.5
tictactoe	27	232	958	98.3	87.2	93.1
krkp	38	594	2398	97.9	98.2	95.9
mushroom	122	974	8124	99.8	99.8	99.1
Average				95.8	87.8	90.7

Table 1: Accuracy in the set of concepts selected.

These results show that the quality of the generalization obtained by the boolean network is comparable and, in many cases, better, than the one obtained by alternative approaches. These alternative approaches are not the best ones proposed to these problems, but they are general purpose robust learning algorithms and we believe the comparison shows that boolean networks trained with this or other algorithms are viable alternatives for problems of this type. In some cases (xor8, tictactoe) the results are the best known to the authors for algorithms that are not specially biased to perform well in these concepts.

The CPU time spent in the different examples varied wildly. Due to the limitations inherent to the algorithm used to solve the set covering problem, MIFES was run with a timeout value that varied with the size of the covering problem to be solved. ID3 was by far the fastest algorithm and MIFES the slowest. The Cascade-Correlation algorithm was in between but many times slower than ID3.

The synthesis algorithm is integrated in the Berkeley Logic Synthesis System (SIS) and it is straightforward to generate the layout of boolean networks obtained using the learning algorithm. Figure 3 shows the layout obtained for one run of the tic-tac-toe problem using a standard cell technology. In 1.5μ technology, this implementation of the boolean network takes an area of 0.31 square millimiters.

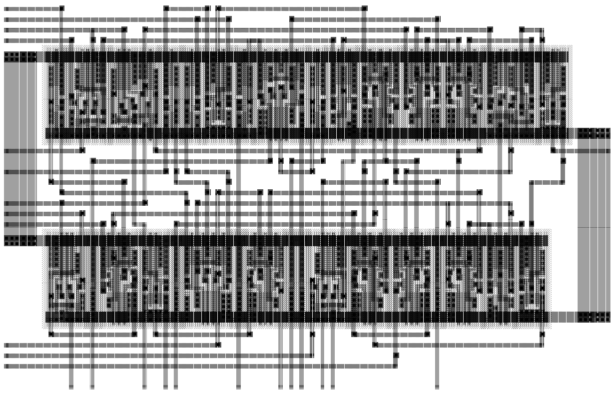

Figure 3: Layout for one network obtained for the tic-tac-toe problem.

5 Conclusions

The results presented show that, for some problems, boolean networks generated from training set data can be an interesting alternative to threshold gate networks. The approach is specially interesting when an hardware implementation is desired and high speed and small size are important characteristics. The type of problems that can be tackled by the current version of the algorithm is restricted to concepts defined by boolean or discretely valued attributes and low noise levels.

The algorithm proposed in this paper has some limitations that should be addressed in future work in this area:

The present version cannot handle continuously valued attributes in an efficient way. Methods based on the bisection of continuous ranges or in discrete coding are currently being studied.

The solution of the set covering problem by a branch and bound algorithm is very expensive and, in many cases, leads to inadequate results when the search is stopped after a specified time. Approaches that aim at obtaining an approximate solution with reasonable computational resources should be tried. Input attribute noise and class noise are handled in a very primitive way. Algorithms for the identification and removal of outliers can be used to surpass this limitation.

Appendix

Description of the functions used in the results section:

sm12 : $f(x_1 ... x_{12}) = (x_1x_2 + x_3x_4 + x_5x_6)(x_7x_8 + x_9x_{10} + x_{11}x_{12})$

sm18 : $f(x_1 ... x_{18}) = (x_1x_2 + x_3x_4 + x_5x_6)(x_7x_8 + x_9x_{10} + x_{11}x_{12})(x_{13}x_{14} + x_{15}x_{16} + x_{17}x_{18})$

str18 : $f(x_1 ... x_{18}) = (x_1x_2x_3 + x_4x_5x_6 + x_7x_8x_9)(x_{10}x_{11}x_{12} + x_{13}x_{14}x_{15} + x_{16}x_{17}x_{18})$

References

[1] D. Haussler A. Blumer, A. Ehrenfeucht and M. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[2] K. A. Bartlett, D. G. Bostick, G. D. Hachtel, R. M. Jacoby, M. R. Lightner, P. H. Moceyunas, C. R. Morrison, and D. Ravenscroft. BOLD: A multi-level logic optimization system. In *IEEE International Conference on Computer-Aided Design*, 1987.

[3] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Classifying learnable geometric concepts with the vapnik-chervonenkis dimension. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 273–282, Salem, 1986. ACM.

[4] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[5] R. K. Brayton and Curt McMullen. The decomposition and factorization of boolean expressions. In *Proceedings of International Symposium in Circuits and Systems*, pages 49–54. (Rome), 1982.

[6] R. K. Brayton, R. L. Rudell, A. L. Sangiovanni-Vincentelli, and A. R. R. Wang. MIS: A multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design*, CAD-6(6):1062–1081, November 1987.

[7] Paul E. Dunne. *The Complexity of Boolean Networks*. Academic Press, San Diego, 1988.

[8] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, San Mateo, 1990. Morgan Kaufmann.

[9] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[10] S. Hong, R. Cain, and D. Ostapko. Mini: A heuristic approach for logic minimization. *IBM Journal of Research and Development*, 18(5), 1974.

[11] P. M. Murphy and D. W. Aha. *Repository of Machine Learning Databases - Machine readable data repository*. University of California, Irvine.

[12] A. L. Oliveira and A. Sangiovanni-Vincentelli. Lsat - an algorithm for the synthesis of two level threshold gate networks. In *Proceedings of ICCAD-91*, pages 130–133. IEEE Computer Society Press, 1991.

[13] A. L. Oliveira and A. Sangiovanni-Vincentelli. Constructive induction using a non-greedy strategy for feature selection. In *Proceedings of the Ninth International Conference in Machine Learning*, San Mateo, 1992. Morgan Kaufmann.

[14] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[15] J. Rissanen. Stochastic complexity and mideling. *Annals of Statistics*, 14:1080–1100, 1986.

[16] J. Wogulis W. Iba and P. Langley. Trading off simplicity and coverage in incremental concept learning. In *Proceedings of the Fifth International Conference in Machine Learning*, pages 73–79, San Mateo, 1992. Morgan Kaufmann.