Optimal Synthesis of Boolean Functions by Threshold Functions

José Luis Subirats, Iván Gómez, José M. Jerez, and Leonardo Franco

Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga,
Campus de Teatinos S/N, 29071 Málaga, Spain
{jlsubirats, ivan, jja, lfranco}@lcc.uma.es
http://www.lcc.uma.es/~lfranco/

Abstract. We introduce a new method for obtaining optimal architectures that implement arbitrary Boolean functions using threshold functions. The standard threshold circuits using threshold gates and weights are replaced by nodes computing directly a threshold function of the inputs. The method developed can be considered exhaustive as if a solution exist the algorithm eventually will find it. At all stages different optimization strategies are introduced in order to make the algorithm as efficient as possible. The method is applied to the synthesis of circuits that implement a flip-flop circuit and a multi-configurable gate. The advantages and disadvantages of the method are analyzed.

1 Introduction

In this paper we introduce an algorithm for finding a set of threshold functions (also called linearly separable functions) that will compute a desired (target) arbitrary Boolean function. This problem is known as the synthesis problem of Boolean functions and has been much studied since the 60's ([1,2,3]). The interest in using threshold gates or linearly threshold functions instead of standard logical AND, NOT and OR gates relies on the fact that threshold elements are more powerful than standard gates and as a consequence, the size of the circuits that can be constructed to compute the desired functions can be smaller. There is an extra interest in the study of threshold circuits as they are very close related to neural networks models, and thus some of the properties and characteristics of the circuits also apply to neural networks architectures. Our main motivation for the development of this method is the study of neural networks architectures [4,5]. The standard practice for the architecture selection process within the field of artificial neural networks is the trial-and-error method that is very time consuming. Knowing and characterizing which architectures are best suited for a given class (or set) of functions can much help to the development and refinement of existing methods for constructing better neural architectures and also for gaining further understanding on the learning process. The problem of finding optimal architectures is relevant also for a more theoretical point of view within the area of circuit complexity, as different existing bounds can checked and/or improved ([6]).

The method introduced in this paper permits the synthesis of Boolean functions in terms of the set of threshold Boolean functions. The standard threshold circuits, use threshold gates connected by weights that in the present formulation are replaced by threshold functions. Obtaining the whole set of linear separable functions is a complicate and computationally intensive problem and the set functions of threshold functions have been only obtained up to $N = 10$ variables ([3]).

Research in threshold logic synthesis was done mostly in the 1960s ([1,2,3]). Nowadays, different implementations of circuits by threshold gates are available, and several theoretical results have been obtained [6,7,8,9] together with different applications ([10]). The main difference with previous approaches is that the present work is aimed to find optimal architectures.

2 Synthesis of Boolean Functions by Linearly Separable Functions

We introduce in this work a new method for finding a set of linearly separate functions that will compute a given desired Boolean function (the target function). The method use the linearly separable set of functions as basis functions (or primitives) and thus assumes that whether a list of the functions is available or that a method for testing whether a given function is linearly separable is provided (the first approach is used throughout this work). One important difference with previous works is that our approach works straightforward with the set of threshold functions as basis functions and not through their equivalent representation by threshold gates and weights.

2.1 The Algorithm

Without loss of generality, we will analyze the case of architectures comprising a single layer of hidden nodes. The algorithm can be straightforwardly extended to the case of several layers of nodes. The algorithm is aimed to find a set of linearly separable functions for the hidden nodes and for the output node that implement a desired target function. The choice of the output function is crucial for the procedure as once that an output function has been chosen, the algorithm will test all possible solutions and then the choice of the output function has a multiplicative computational effect. However, the search procedure of the hidden node functions is the most intensive and complicated step, and different optimization strategies can be implemented.

Selection procedure for the output function. From first principles, there are two “logic” choices for the output function: the first one would be to select an output function as the most similar threshold function to the target one. This procedure is not straightforward as a measure of similarity between the functions is needed, but a simple choice seems to select the closest linearly separable function in terms of the Hamming distance (number of different bits) between the outputs of the two functions. The second logic choice could be to select the output function according to the success rate of the functions (or of similar functions in lower dimensions) in previous cases. While the two mentioned approaches seems to merit consideration, there is also a computationally effective oriented choice. In this case functions that will generate a shorter node search procedure will be considered first. This was the approach used in this work. It will be more clear later in the work after the node search algorithm is introduced, but the general idea is that as the node search algorithm generate a tree of possible solutions and as the number of branches at a ramification point is proportional to the number of solutions, the more computational intensive cases will occur for output functions with a balanced number of 0's and 1's. Thus, functions with a non-balanced number of 1' and 0's will be considered earlier as candidates for the output function. This choice, even if motivated by computational efficiency, may be not the more efficient, specially if unbalanced functions results to be not very good for the synthesis problem. On the contrary, if almost all functions are more or less equivalent, or if unbalanced functions are well suited for this task of generating non-threshold functions, then the approach taken might be the most computationally effective as it was intended. Some preliminary experiments showed that the unbalanced functions seem to be good as output functions.

Selection procedure of the hidden nodes function. Once the output function has been set to a given linearly separable function, the problem translates into finding a set of linearly separable functions for the hidden nodes that would make the circuit to implement the target function. The algorithm analyze the different possibilities of mapping the inputs into a set of recoded inputs for the output function, in way such that the output function might be able to map them correctly onto the solution. To exemplify the procedure, we will analyze the simple case of finding the optimal circuit for the NXOR function. The NXOR function is a Boolean function of two input bits where the output of the function is 0 if the sum of the two input bits is 1 and the output is 1 if the sum of the input bits is even (inputs 0-0 and 1-1). The NXOR function is a classic example of a non linearly separable function and thus an architecture with two hidden nodes, at least, is needed for their implementation with threshold functions (it is known that 2 hidden nodes are enough for implementing this function). For exemplifying the procedure we use an architecture with two hidden nodes, both connected to the two inputs. In Fig. 1 such an architecture is shown, where I_0 and I_1 indicate the input nodes that will feed their values into the hidden nodes. The hidden nodes will compute two linearly separable functions named f_0 and f_1 while the output function is indicated in the figure by f_{out}.

For the example under consideration, we will analyze the situation in which the output function has been already selected. We consider as output function the function AND, that produce an output 1 only for input values 1-1. The

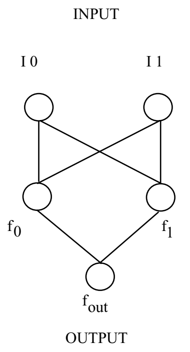

Fig. 1. A two hidden node architecture used for showing the procedure of finding a solution for the NXOR function. The inputs to the network are indicated by I_0 and I_1. The hidden nodes will compute as a function of the input bits two linearly separable functions indicated by f_0 and f_1 that will send their outputs to the output node function indicated by f_{out}.

functions will be designated according to the outputs that the function produce in response to the inputs, with values ordered according to their binary value. For the present case, in which input functions of two bits are considered the order of the inputs and outputs is as follows: first, the output in response to the input 0-0, second to the input 0-1, third to 1-0 and finally the output corresponding to the input 1-1. In this way, the AND function is indicated by 0001, while the NXOR function would be coded as 1001. The procedure for finding the threshold functions for the node functions f_0 and f_1 proceeds in a number of steps in which the possible outputs for f_0 and f_1 are considered.

Now we consider the first output bit of the target function, that is equal to 1 (in response to the first input 0-0). As the function AND was selected as output function (f_{out}), and this function produce a 1 in response only to an input 1-1, it is needed that both node functions, f_0 and f_1, produce an output 1 in response to the input pattern 0-0. That is, the node functions have to recode the original input of the network, the input 0-0, into the input 1-1 for the output node, otherwise the output function would not coincide with the target one. Now the procedure continues with the second input bit, 0-1, for which the target output is 0. The selected output function produce a 0 in response to three input cases and then there are three possible cases into which the input 0-1 can be recoded by the node functions. The function f_{out} gives a 0 in response to the inputs 0-0, 0-1 and 1-0 and then we obtain, putting together the results for the first and second output bits the following tree of possibilities, shown in Fig. 2. The three cases are indicated in the diagram as branches of the first case on the top, for which there was a single possibility.

The procedure continues considering the rest of the cases corresponding to the third and fourth input bits. The whole tree generated as the procedure is

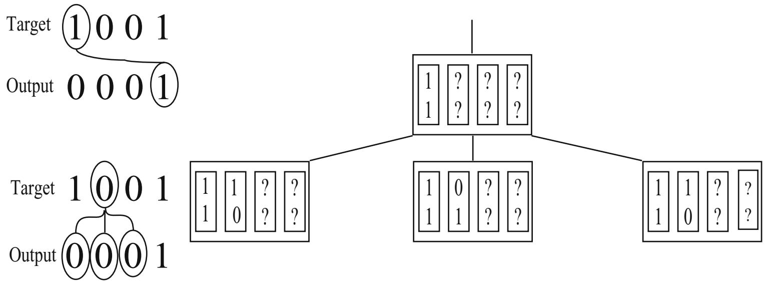

Fig. 2. The possible cases for the node functions f_0 and f_1 as the two first input bits are considered for the case of choosing the NXOR as target function and the AND as output function. Within the boxes, the 4 pairs of values correspond to the outputs of the two functions in response to the 4 inputs. The question mark indicates that the value is still undetermined.

applied is shown in Fig. 3. The nine boxes in the last two bottom rows contain the nine function candidates for the node functions, but the only possible ones are those boxes containing two threshold (linearly separable) functions (the two boxes at the rightmost end of the bottom row containing functions with only one output bit equal to 1, as all the other boxes contain the function 1001 that is non-linearly separable).

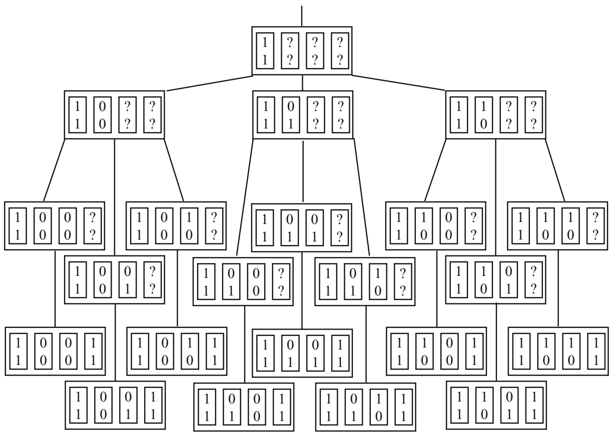

Fig. 3. The whole tree of possibilities for the choices of the node functions for the case of the NXOR target function. The nine boxes in the last two bottom rows are the nine candidates for the functions but the only possible ones are those in which both functions are linearly separable functions (See the text for details).

The node searching procedure ends if two linearly separable functions are found, but if this is not the case, a different output function, f_{out}, is chosen from the set of threshold functions and the whole procedure is repeated until all the threshold functions with a number of variables equal to the number of nodes are tested. If a solution is not found, the number of nodes is augmented by one and the whole procedure repeated, and so on.

2.2 Optimization Steps: Order of Branching and Pruning by Testing Linear Separability

Several optimization steps were introduced to speed up the computational process. We will refer here to only the most important ones. The first optimization step is about the order in which the input bits are considered for the node searching process. If the output function is unbalanced, it is convenient to analyze first the inputs of the output function that produce an output that occur less infrequently as the degeneracy of the output values is related to the number of branches that are created in the tree of possible choices for threshold functions. The second optimization concern the test of linear separability for the partial defined functions. It is possible to analyze by previously constructing a tree with all the linearly separable functions, whether a partial defined Boolean function will end up producing some threshold functions. If the partially defined function will not produce any threshold function, the branching process is ended at that point, and the tree is pruned. Alternatively, the checking for linear separability can be done by testing the unateness of the variables, as all threshold functions are unate (but not conversely, so the test is a partial one).

3 Application of the Algorithm to the Construction of Optimal Architectures for a Flip-Flop Circuit and a Configurable Gate Circuits

We applied the algorithm developed in the previous section to the construction of threshold circuits that implement in an optimal way (with a minimum number of nodes in a single hidden layer) a flip-flop circuit and a configurable gate. The flip-flop circuit (or bistable) is a standard digital circuit in electronics that incorporates the storage of memory in its functioning.

The resulting architecture obtained by the algorithm has three nodes in the unique hidden layer and is depicted in Fig. 4.

The three hidden functions receive input from the 4 inputs and then project into the two outputs units. The "fan-in max" (maximum number of connections that a node receives) is 4 and thus, the whole set of threshold functions of up to 4 variables was needed. The four inputs of the circuit are the Memory input bit (M), the Enable bit (EN), the Input bit (IN) and the Read/Write bit (R/W). The circuit has two output nodes, the first one (f_M) feedbacks into the memory bit permitting the storage of memory and the second one is the true output of the circuit, designated in the figure as (f_{out}) . When the enable bit is OFF (value 0), the circuit outputs the value of the Input bit (I). When the Enable (EN) bit

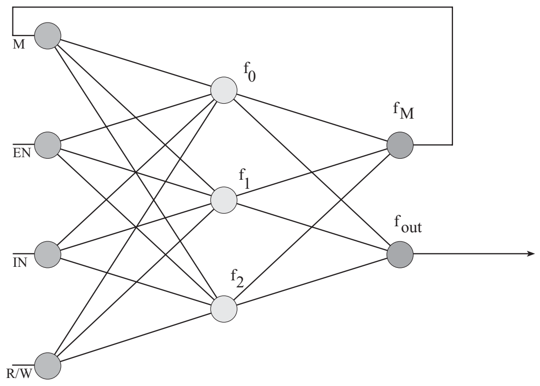

Fig. 4. The architecture designed to implement a flip-flop circuit. The architecture has a single hidden layer containing three nodes and two outputs nodes. One of the output nodes, indicated as f_M, feedbacks the input value to the memory neuron, and the other output, f_{out}, can be considered as the real output of the network.

is ON the circuit works depending on the value of the R/W bit, reading the value stored in memory and sending it to the output or by writing the value of the input into the memory. The truth table for the flip-flop circuit implemented is shown in Fig. 5.

In Table 1 the output values of the obtained functions that implement the circuit are shown. The case shown is only one of the 120 cases obtained.

Table 1. The threshold functions obtained for one of the solutions found for the flip-flop circuit. The node functions are functions of N=4 input bits and then 16 output values exists, while the output functions have a fan-in of 3 and then 8 binary values are needed to code them.

Function	Output bits
f_0	0000000111111111
f_1	0011001100111011
f_2	1011000011111010
f_M	00000111
f_{out}	00010001

Also a circuit to implement a multi-configurable gate was constructed. The circuit can compute as desired a AND, OR, NXOR or NAND as a function of the two inputs (IN_0 and IN_1). The function to be computed is selected by two extra input bits (C_0 and C_1), and the truth table of this multi-configurable gate is shown in Fig. 6.

The architecture implementing the multi-configurable gate is shown in Fig. 7 where it is also shown, close to the nodes, the functions obtained from the

M	EN	IN	R\W	Out-M	OUTPUT
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	0	#
0	1	1	0	0	0
0	1	1	1	1	#
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	0	#
1	1	1	0	1	1
1	1	1	1	1	#

Fig. 5. Truth table of a flip-flop (bistable) circuit. The circuit, when the Enable (EN) input is activated, can store a value in memory (Write procedure) or transmit the stored value (Read procedure) depending of the value of the R/W input bit. If the Enable bit is off, the network simply output the value of the single input (IN).

C0	C1	IN0	IN1	OUTPUT
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Fig. 6. Truth table of a multi-configurable circuit with N=2 inputs. The two selection bits,(C_0 and C_1), indicate which function should be computed.

procedure coded by their outputs. It is also shown in the figure, the operation of the circuit for two inputs, the first and the fourth, showing for the first input how this is mapped by the node functions to the first bit of the output node to produce a desired output of 0. For the fourth input, the output of the three hidden nodes recoded it as 1-0-1 producing a desired output of 1 by the output node.

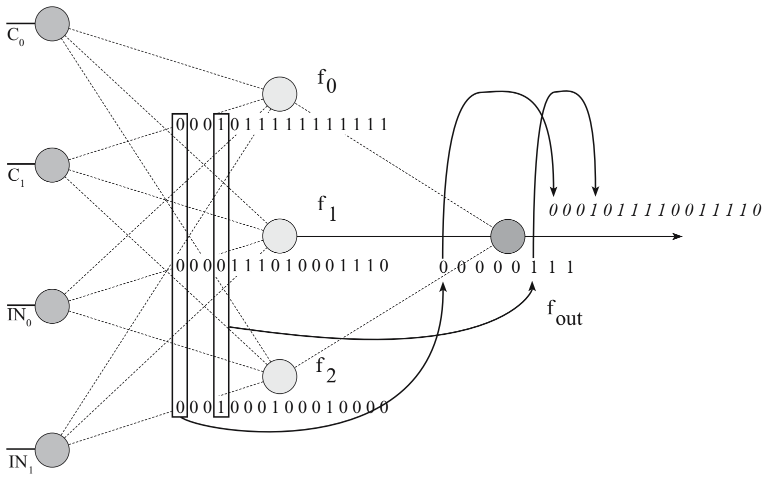

Fig. 7. The architecture of the circuit computing a multi-configurable gate that can work depending of the value of the indicator bits (C_0 and C_1) as a AND, OR, NXOR or NAND gate. Close to the nodes, the functions that make the circuit to work as desired are indicated by the value of their outputs in response to the input patterns. (See the text for more details)

4 Discussion

We have introduced a new method for finding the optimal circuit architecture that computes an arbitrary Boolean function in terms of linear separable primitives. Starting from different "smart" choices of the output function the algorithm tries to find linearly separable functions for the hidden nodes according to the values of the output and target function. Eventually if a solution has not been found, the algorithm searches for all the possible solutions and in that way does an exhaustive search. Different optimization steps were introduced to improve the computational efficiency. The method was successfully applied to the construction of threshold circuits for the flip-flop circuit and for a multi-configurable gate. For both implementations the whole synthesis procedure took less than half of a minute, but we are aware that they only involved nodes with a fan-in max of 4. The worst case complexity of the algorithm is of order $\mathcal{O}(2^{MN^2}\ 2^{M^2})$, where N is the number of input variables, M is the number of hidden nodes and the factors of the type 2^{N^2} arise because that is the order of the number of threshold functions on N variables (It is worth noting, as a reference value, that the total number of Boolean functions on N variables is 2^{2^N}). We are currently computing the average case complexity for the case of all functions on 4 variables that can be treated exhaustively. We believe that the algorithm can be of practical application for larger dimensions of up to 8 or 10 in its present way and it seems also possible to develop from the current algorithm non-optimal procedures for larger dimensions. We are currently analyzing all the optimal architectures for all the existing Boolean functions of 4 variables, measuring the speed of the algorithm against standard benchmarks and studying the properties of multi-layer and modular neural network architectures.

Acknowledgements

The authors acknowledge support from CICYT (Spain) through grant TIN2005-02984 (including FEDER funds). Leonardo Franco acknowledges support from the Spanish Ministry of Education and Science through a Ramón y Cajal fellowship.

References

1. Winder, R.O. (1962). *Threshold Logic*, Ph.D. dissertation, Department of Mathematics, Princeton University.
2. Dertouzos, M.L. (1965) *Threshold Logic: A Synthesis Approach.* Cambridge, MA: The M.I.T. Press.
3. Muroga, S. (1971).*Threshold Logic and its Applications*, Wiley, New York
4. Franco, L. (2006). Generalization ability of Boolean functions implemented in feedforward neural networks. *Neurocomputing.* In Press.
5. Franco, L. and Anthony, M. (2006). The influence of oppositely classified examples on the generalization complexity of Boolean functions. *IEEE Transactions on Neural Networks.* In Press.
6. Siu, K.Y., Roychowdhury, V.P., and Kailath, T. (1991). Depth-Size Tradeoffs for Neural Computation *IEEE Transactions on Computers*, *40*, 1402-1412.
7. Oliveira, A. L. and Sangiovanni-Vincentelli, A. (1991). LSAT: an algorithm for the synthesis of two level threshold gate networks. In: *Proceedings of the ACM/IEEE International Conference on Computer Aided Design, Santa Clara, CA, IEEE Computer Society Press*, 130-133.
8. Noth, W., Hinsberger, U., and Kolla, R. (1996). TROY: A Tree-Based Approach to Logic Synthesis and Technology Mapping, *In: Proceedings of the 6th Great Lakes Symposium on VLSI*, p. 188.
9. Zhang, R., Gupta, P., Zhong, L. and Jha, N. K. (2005). Threshold Network Synthesis and Optimization and Its Application to Nanotechnologies, *IEEE Transactions on computer-aided design of integrated circuits and systems*, *24*, 107-118.
10. Beiu, V., Quintana, J.M. and Avedillo, M.J. (2003). LSI implementations of threshold logic - A comprehensive survey, *IEEE Trans. Neural Networks*, *14*, 1217-1243.