A Continuous Restricted Boltzmann Machine and Logistic Regression Framework for Circuit Classification

Leandro Maia Silva
Department of Informatics COLTEC
Universidade Federal de Minas Gerais
Belo Horizonte, MG 31270–901
elmaia@coltec.ufmg.br

Fabricio Vivas Andrade
Computer Science Department
Centro Federal de Educacao Tecnologica
Belo Horizonte, MG 30421–169
vivas@decom.cefetmg.br

Luiz Filipe Menezes Vieira
Computer Science Department
Universidade Federal de Minas Gerais
Belo Horizonte, MG 31270–901
lfvieira@dcc.ufmg.br

***Abstract*—Circuit identification and classification is an important field of research in Electronic Design Automation (EDA). This paper provides a novel framework for circuit classification based on a Continuous Restricted Boltzmann Machine and Logistic Regression. An undirected graph representation of a circuit CNF instance is created and employed to perform CNF-signatures' search, thereof we classify it. A library with CNF-signatures of thousands of logic gates and functional blocks was pre-generated by our framework. These signatures are searched in the original CNF instance graph via traditional subgraph isomorphism algorithm and the results are applied as inputs for the Boltzmann Machine. Finally, a Logistic Regression classifier can determine to which class of the circuit each instance belongs. Our implementation is capable to correctly identify several circuit classes such as adders, multipliers and dividers with accuracy over 92%.**

***Index Terms*—circuit classification, restricted Boltzmann machines, subgraph isomorphism, neural networks**

I. Introduction

Seeing that structural recognizing and recovering of logical and arithmetic circuits became a prolific research area [1]–[5], given the current impossibility to recover all structural information or completely recognize circuits, there are different levels of recovery that answer to different questions. Certain applications benefit from knowing the connections and the gates of the circuit; whereas others view these data as not relevant, weighing the complex functional block as the desired information.

Considering scenarios in which applications are based on circuits, generic SAT solvers have been extended in order to be employed explicitly on derived from circuits CNF (Conjunctive Normal Form) input [6]. In order to propel SAT solvers' performance, circuit structural recovering became broadly applied.

Among the main techniques that utilize the circuit structure are 1) parallel simulation of a small number of random inputs, 2) detection of correlation between the signals using hash and 3) a guide for SAT solvers to refute the equivalence of related signals and thus help you generate more concise and efficient conflict clauses. When using the information of the circuit structure, the authors assume that the information present in it was lost during the conversion to a CNF-SAT [6] instance.

Some CNF formulas can contain a large number of clauses derived from circuits, although they did not originate from circuits. An example of this type of formula is property verification, in which the circuit part describes the hardware and the non-circuit part represents more general properties, such as "at most m of n signals can be low". In addition, several circuits can appear in the form of mathematical properties, without the knowledge of the designer who performed the coding.

Despite the fact that many SAT instances are not derived from circuits, other factors such as the involvement of unoriented ports (XOR, for example) make it difficult to recover the circuit present in the formula. Even in these cases, Roy et al. [7] show that parts of CNF formulas compatible with the circuit structure can be found and this recognition of circuit parts facilitates SAT solvers, regardless of whether the reconstructed circuits are unique or not.

Due to their efficiency in solving SAT problems, SAT solvers have become a standard tool in many applications. Even though this class of problems is NP, the solvers have proved to be efficient in solving practical problems, with acceptable time periods.

A. Motivation

Most SAT solvers expect their input as CNF, more precisely in CNF-DIMACS [8]. Whenever dealing with electronic circuits, it is possible to code the circuit in CNF under linear time using the Tseitin [9] transformation, which is the most used coding. Although simple and efficient, this transformation flattens the circuit, destroying its topology, causing the logic gates and their connections to be lost and an equivalent circuit composed of a large AND with ORs and inverters connected to it is used instead. The lost information of the circuit structure can be useful for SAT solvers that benefit from this information.

The structural recovering task, from the CNF-encoded circuit, may seem unnecessary inasmuch as the SAT solver could make use of the original circuit information from which the

CNF was produced. In an ideal scenario, this is the most efficient process; however, the original circuit may not always be available. In regards to the extraction of information being useful due to the absence of the original circuit, two distinct scenarios occur: 1) the CNF is not generated from a circuit, such as planning and scheduling or routing in FPGAs; 2) the CNF is generated from a circuit but it is not available, as is the case with most of the benchmarks used in SAT solver competitions.

Nonetheless, any type of extra information related to the circuit structure within the formula, regardless of whether the source of the circuit is an electronic circuit, increases the solvers' performance that had access to said information.

B. Our contribution

Our work presents a circuit recognition framework based on the identification of functional blocks using subgraph isomorphism and a Continuous Restricted Boltzmann Machine and Logistic Regression Classifier for processing this information. Among the main contributions are 1) the use of isomorphism in the subgraphs for detecting complex blocks present in circuits, not only simple logic gates, 2) the possibility of determining the circuit size based on the number of occurrences of each block, and 3) the creation of a flexible tool that allows the user to define which function blocks they want to detect without source code changes. To the best of our knowledge, this is the first attempt to detect complex functional blocks using graph isomorphism combined with neural networks.

C. Organization

This paper is organized as follows: Section II presents a brief summary of the related work in structural recovering of circuits and the consequent performance gains where this information was used; in addition to our contribution associated with the aforementioned work. The following section provides the reader with a theoretical foundation that should allow a better understanding of our framework. In section IV we demonstrate our decisions and implementation details for the framework and in section V we supply the experiments carried and their results. The final section ends the document with our conclusions and possible directions for future work.

II. Related work

Under the previous section, we noticed that, in general, information loss is inevitable when coding a circuit in CNF. Hence, we could state that, due to this consideration, there was not much effort in extracting the circuit structure from CNF.

The earliest works in structural recovering accessible were extraction of equivalences between literary [1] and simple AND and OR gates [10]. Roy et al. were the first to explicitly extract the logic gates from CNF [7]. They coined the concept of CNF-Signature, which, succinctly, is the CNF encoding of a logic gate. This signature is transformed into a graph and subgraph isomorphism algorithms were used to find matches of those signatures in the circuit's graph. Clearly, the focus of their work was in finding basic logic gates (AND, OR, NAND, NOR, XOR, XNOR and inverter), at the cost of imposing strong restrictions on the occurrences of XOR/XNOR gates during the extraction process.

Later work by Zhaohui Fu and Sharad Malik [11] is based on a library of logic gates that describes target patterns to be extracted. Using this library ensures a more flexible approach, but less efficient than specifics pattern matching. Another contribution of this work is that, according to the authors, they not only extract logic gates but also guarantee to extract the largest acyclic circuit possible through the use of SAT solvers.

More recently, Harald Seltner [12] defended his master's thesis over a strong theoretical foundation in Zhaohui Fu's work. As his main result, the cnf2aig tool was developed; the tool reconstructs the circuit from the CNF and generates, as output, and-inverter graphs (AIG) [13]. The author guarantees that the reconstructed circuit is as close to its original as possible concerning the gates that the tool's algorithms can detect.

Structural information is very useful in other types of problems that are not related to the SAT. An example of this is the work of Chakraborty [5] whose identification of certain multiplier structures and the addition of special assertions to the input formula allowed the solution of several formulas faster; considering they would usually timeout, despite Satisfiability Modulo Theories (SMT) solvers being known as inefficient in formulas with bit vectors in multipliers.

Previously presented works, use exact algorithms to obtain information about the circuit structure. The main drawback of the exact algorithm's approach is that performance is degraded as the circuit gets bigger and new functional blocks are coded. To overcome this difficulty and still yield the results quickly, the state-of-the-art works in structural recovery use some level of the neural network in the process. Dai and Brayton [14] use the Convolutional Neural Network (CNN) in a new form of circuit representation to perform classification, as well as function detection and location. Similarly, Fayyazi et al. [15] use CNNs and another form of circuit representation to perform structural recovering. Both use gate-level-netlist as input to carry out their experiments.

Silva et al. [16] have circuits at a lower level of abstraction, since they are encoded in CNF. They propose to transform the circuit encoded in CNF into an equivalent image so that CNN can perform and learn the patterns of the circuits in order to classify it. The performance limitation of this approach is that images recognition by CNNs is very associated with the Euclidean proximity of the modules, which implies that they have to be locally near to be well recognized.

Despite their excellent performance in several areas, the major challenge for using CNN to recover circuits' structures is how to properly represent the circuit in order to aid CNN's learning process.

Our work shows an improvement in relation to the abovementioned works because, in addition to detecting the logic gates, it also allows the detection of complex functional blocks such as adders and multipliers. The key advantage of the detection of these large blocks is that is possible to infer the

$$z = not(x) \equiv \Big(\neg x \vee \neg z\Big) \wedge \Big(x \vee z\Big)$$
$$z = and(x_1, ..., x_n) \equiv \Big[\bigwedge_{i=1}^{n} \, (x_i \vee \neg z)\Big] \wedge \Big(\bigvee_{i=1}^{n} \, \neg x_i \vee z\Big)$$
$$z = or(x_1, ..., x_n) \equiv \Big[\bigwedge_{i=1}^{n} \, (\neg x_i \vee z)\Big] \wedge \Big(\bigvee_{i=1}^{n} \, x_i \vee \neg z\Big)$$
$$z = nand(x_1, ..., x_n) \equiv \Big[\bigwedge_{i=1}^{n} \, (x_i \vee z)\Big] \wedge \Big(\bigvee_{i=1}^{n} \, \neg x_i \vee \neg z\Big)$$
$$z = nor(x_1, ..., x_n) \equiv \Big[\bigwedge_{i=1}^{n} \, (\neg x_i \vee \neg z)\Big] \wedge \Big(\bigvee_{i=1}^{n} \, x_i \vee z\Big)$$

Fig. 1: Tseitin transformation for some logic gates.

direction of numerous signals and logic gates since they are easily detected as the inputs and outputs of the blocks. None was done by the previous works.

III. A Boltzmann Machine and Logistic Regression Framework for Circuit Classification

Our framework is composed of three main components: a subgraph isomorphism search algorithm to find CNF-signatures, Continuous Restricted Boltzmann Machine neural network, and a Logistic Regression Classifier. Prior to giving details about these components, it is imperative to explain how combinational circuits are encoded in CNF.

A. Encoding Circuits in CNF

Combinational circuits encoding in CNF-DIMACS is very direct and made in linear time [9]. As any combinational circuit can be described as a sum-of-products (Disjunctive Normal Form - DNF) or a product-of-sums (Conjunctive Normal Form - CNF), it is always possible to write it using the three basic Boolean operators: OR ($\vee$), AND ($\wedge$) and NOT ($\neg$). A Boolean variable can only assume the values TRUE or FALSE. In a CNF formula, a clause is a disjunction of a number of variables, where they can be negated or not. An expression in CNF is, therefore, a conjunction of one or more clauses. As seen in section I, the most common input format for SAT solvers is exactly a formula expressed in CNF.

According to the Tseitin transformation, the most employed method when performing logical gate encoding, as previously stated, each of the gate's inputs, as well as its outputs, is represented by a variable. A logic gate with three inputs and one output will be encoded as an expression containing four variables, where the number and the formation of the clauses will depend on the function being encoded.

Fig.1 presents the encoding for the logic gates NOT, AND, OR, NAND and NOR in CNF formulas. XOR and XNOR gates can also be easily coded, but their expressions involve 2^n clauses, where n is the number of inputs of the gate.

In summary, the encoding of a circuit to CNF consists of expressing the characteristic functions of each logic gate in the form of CNF. Each logic gate has a unique characteristic function, but it can be expressed in many CNF formulas.

Consequently, we can take into consideration that a CNF-Signature for a logic gate is a CNF formula that represents the characteristic function of the alluded gate.

While a logic gate can have several CNF-Signatures, a given CNF-Signature identifies no more than one logic gate.

B. Graph Isomorphism and CNF Signatures

Finding out all the occurrences of a given CNF-Signature in a circuit instance, we can reduce our problem to the problem of recognizing isomorphism in subgraphs [7]. The subgraph isomorphism problem is known as NP-complete for graphs in general, yet there are several practical algorithms with good performance [17], [18].

Every CNF formula can be represented by a directed bipartite graph without any loss of information.

Taking a bipartite graph BG = (V_c, V_v, E_g), where V_c corresponds to the set of vertices that represent the clauses, V_v to the set of vertices of the variables and E_g to the set of ordered pairs (V_a, V_b) indicating the edge that leaves the vertex V_a and enters the vertex V_b, in order that $(V_a, V_b) \in$ $((V_c, V_v)$ or $(V_v, V_c))$.

An edge is created between two vertices whenever a variable is associated with a clause, its direction indicates whether that variable appears in the negated form. As an illustration, we have mapped the CNF formula of the 2-AND logic gate in a directed bipartite graph.

$$c = and(a, b) : (\neg a \vee \neg b \vee c) \wedge (a \vee \neg c) \wedge (b \vee \neg c)$$

Assuming the clauses exactly in the order in which they appear, numbered from one to three. Whenever a variable appears in the clause, it is called a literal, whether in its negated form or not. A literal is numbered according to the number of variables, so in a formula with three variables, suppose a, b and c, they would be represented as 1, 2 and 3. Thus, the formula for the 2-AND gate presented could be written with the three clauses:

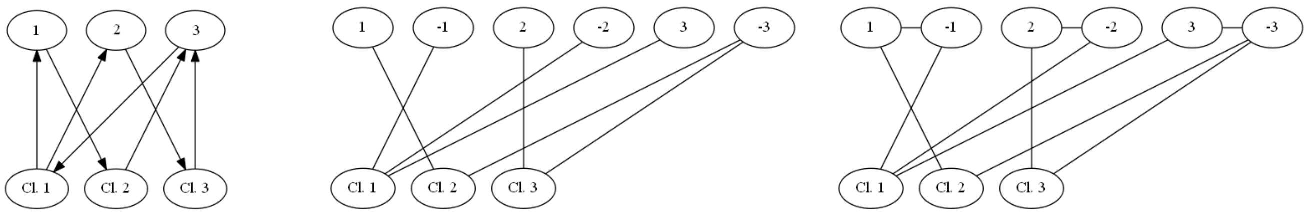
```
-1 -2 3
1 -3
2 -3
```

This formula can be represented by the graph in Fig.2a.

Whenever the literal appears in the negated form, its direction is $(V_a, V_b) \in (V_c, V_v)$, and whenever it appears in the non-negated form, its direction becomes $(V_a, V_b) \in (V_v, V_c)$.

The original formula can be restored regardless of the order of the variables and without losses in relation to its characteristic function.

The execution of efficient subgraph isomorphism algorithms requires transforming the directed bipartite graph into an undirected graph, which preserves the existing information. Wherefore, be UDG = $(V_c, V_{v+}, V_{v-}, E_g)$, where V_c as in BG, V_{v+} corresponds to the vertices of non-negated literals and V_{v-} to the vertices of negated literals. Thus, each directed edge V_g of BG becomes an undirected edge V_g of UDG as follows:



(a) Directed Graph (b) Undirected Graph (c) Final Undirected Graph

Fig. 2: Graphs of 2-AND Gate

- If the edge links a clause to a variable it means that the variable appeared in the negated form, then the new edge links the clause vertex to the corresponding variable vertex in V_{e-}.
- Otherwise, the variable appeared in the non-negated form, therefore the variable vertex is linked to a clause vertex; subsequently, the new edge links the clause vertex to the corresponding variable vertex in V_{e+}.

Applying the described procedure, we can transform the graph in Fig.2a into the graph in Fig.2b without any information loss.

The formula continues being recovered from the undirected graph, as initially stated. The down side to this operation, however, is that it becomes impossible to correctly execute the isomorphism algorithm, because the graph presented is disconnected; one part formed by the vertices (1, Cl. 2, -3, Cl. 3) and the other by the vertices (Cl. 1, - 1, -2, 3). Even though we can recover the clauses, it is impossible to establish a relation between the two graphs, and, as a result, it is impossible to recognize a CNF-Signature through a graph. This relationship can be established, nonetheless, by creating an edge between the vertices that correspond to the same variable, making the two vertices related to the same variable, as shown in the Fig.2c.

By doing so, the graph correctly represents its original formula, whereas allowing the execution of the subgraph isomorphism algorithms. The final graph has a number of vertices equal to (number_of_clauses + 2 $\times$ number_of_variables) and the number of edges is equal to (number_of_variables + number_of_literals).

C. Graph representation of CNFs-signatures

Although a logic gate has multiple CNF-Signatures, all the graphs formed from them are isomorphic among themselves, which is to say that the identification of a subgraph, corresponding to a signature in a graph, also allows the identification of a characteristc function. Examples of the basic logic gates are shown in Fig.3 with their respective CNF-Signatures and graphs.

To that end, the ability to identify characteristic functions is important for it allows the identification of logic gates through subraph isomorphism algorithms. Once the subgraph is identified, it represents solely a characteristic function.

In circuits, a connection between logic gates is made using wires, connecting a logic gate's output to another logic gate's

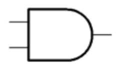
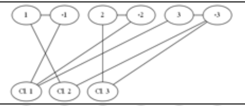
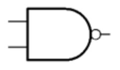
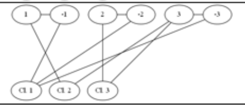
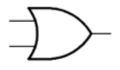
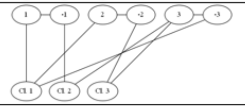
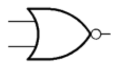
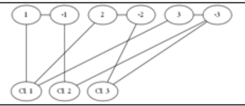
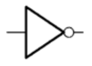
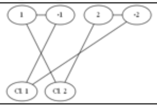
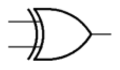
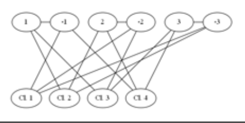
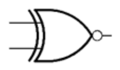
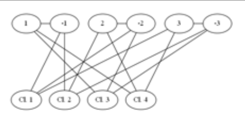
Gate	Operation	CNF-Signature	Graph
	c = a.b	$(\neg a \vee \neg b \vee c) \wedge$ $(a \vee \neg c) \wedge$ $(b \vee \neg c)$	
	c = $\overline{a.b}$	$(\neg a \vee \neg b \vee \neg c) \wedge$ $(a \vee c) \wedge$ $(b \vee c)$	
	c = a + b	$(a \vee b \vee \neg c) \wedge$ $(\neg a \vee c) \wedge$ $(\neg b \vee c)$	
	c = $\overline{a + b}$	$(a \vee b \vee c) \wedge$ $(\neg a \vee \neg c) \wedge$ $(\neg b \vee \neg c)$	
	c = $\overline{a}$	$(\neg a \vee \neg b) \wedge$ $(a \vee b)$	
	c = a $\oplus$ b	$(\neg a \vee \neg b \vee \neg c) \wedge$ $(a \vee b \vee \neg c) \wedge$ $(a \vee \neg b \vee c) \wedge$ $(\neg a \vee b \vee c)$	
	c = $\overline{a \oplus b}$	$(\neg a \vee \neg b \vee c) \wedge$ $(\neg a \vee b \vee \neg c) \wedge$ $(a \vee \neg b \vee \neg c) \wedge$ $(a \vee b \vee c)$	

Fig. 3: Basic logic gates and their respective operations, CNF-Signatures and graphs

input. In the CNF, each logic gate is encoded in a set of clauses, and a clause belongs to a single logic gate. For this reason, the connection between two logic gates can only be made by a variable that represents at the same time a gate's output and another gate's input.

Consider the following circuit, composed by two logic gates: d = AND(a,b); c = NOT(d).

Taking Fig.3 as a reference, we can transform the two logic gates into the following set of clauses:

$d = and(a, b) : (\neg a \vee \neg b \vee d)$, $(a \vee \neg d)$ and $(b \vee \neg d)$

$c = not(d) : (\neg c \vee \neg d)$ and $(c \vee d)$

Since two ports are connected by wires and can be represented by variables, two subgraphs, thus, will be connected by a vertex representing a variable, never a clause. Consequently, it is necessary to distinguish between vertices of clauses and vertices of variables. Moreover, as two vertices represent the

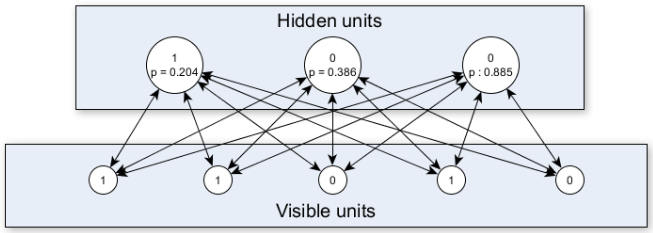


Fig. 4: Representation of an RBM with five visible units and three hidden units

same variable, it is also necessary to distinguish between their non-negated and negated forms. For this, the isomorphism algorithm is applied to a labeled graph, with three different labels: clause, positive literal, negative literal.

This process allows for building signatures of any size, representing any functional block, the basis of this work.

D. Continuous Restricted Boltzmann Machine

A Boltzmann Machine (BM) is a generative stochastic neural network and it was one of the first neural networks capable of learning internal representations, in addition to being able to represent and solve difficult combinatorial problems.

A BM specialization is known as Restricted Boltzmann Machines (RBM) [19] that corresponds to a neural network in the form of a bipartite graph, that is, two layers only. The first layer contains the visible units and corresponds to the network input; the second layer contains the hidden units and also holds the restriction denying intra-layer bonding. The purpose of this connectivity constraint is to facilitate inference and learning.

A representation of an RBM is illustrated in Fig.4. The visible nodes represent the binary input and the hidden nodes represent the value and the associated probability distribution.

Restricted Boltzmann Machines are trained to maximize the product of probabilities to a determined training set. Whenever the input represents binary characteristics, RBMs are powerful tools.

A Continuous Restricted Boltzmann Machine (CRBM) is a variation of RBM that accepts continuous values as input instead of only binaries, which allows it handling many different data; for instance, word-count vectors, images, and heartbeat medical data, and in our case, circuits.

E. Logistic Regression Classifier

Logistic regression is a statistical model that uses a logistic function to model a binary dependent variable. When applied in Neural Networks, it is usually applied in classification tasks.

Therefore, we used it as the last component of our framework to finish the classification work initiated by the CRBM.

IV. Framework Implementation

In this section, we will provide the decisions and implementation details of our framework. As stated previously, our framework is composed of three main components: a subgraph isomorphism search algorithm to find CNF signatures, a Continuous Restricted Boltzmann Machine, and a Logistic Regression Classifier.

A. CNF-Signature Detection

In order to detect CNF-Signatures present in circuits, we implemented a tool that makes use of the Boost library [20]. The mentioned library brings an implementation of VFLib [17] that has several efficient graph isomorphism algorithms.

As presented in Section III, our tool receives a circuit described in CNF-DIMACS [8] as input and builds an undirected labeled graph that corresponds to the circuit received. Similarly, each file with the CNF-Signature that should be checked is also loaded in the tool, which also builds its graph. Once the graph and all the desired signatures are loaded, the tool performs a search on the circuit's graph looking for occurrences of the signature using the subgraph isomorphism algorithms. The vertices are marked, at each occurrence found, so that they can be disregarded in case they might be found again as the signature is being detected. After all the occurrences of a signature have been found, the vertices are cleared and a new signature will be evaluated.

At the end of this process, the number of occurrences of each CNF-Signature found in the evaluated circuit is provided as an output. The information collected from the circuit signatures is used as an input to the neural network, the next step in the circuit recognition process.

B. GBRBM and Logistic Regression Classifier

For the construction of our Neural Network, we set up a pipeline composed of a Gaussian-Bernoulli Restricted Boltzmann Machine (GBRBM) and a Logistic Regression Classifier.

The visible layer of the GBRBM expects the data to be in a range [0, 1], for this reason, the values of each CNF-Signature are scaled so that they are always within the expected range. Each CNF-Signature corresponds to a characteristic of the circuit, which can be 0 when the signature is not present in the evaluated circuit. The values between]0,1] correspond to the number of times that signature was detected in the evaluated circuit in relation to the occurrence of that signature in all other circuits.

The hidden layer of GBRBM corresponds to each of the types of existing circuits, which in our work are 20 (see section V). A second GBRBM similar to the first one is created, differing only in the number of units in the hidden layer, in this second being the five classes (see section V) of circuits.

GBRBM is trained to learn the features of the circuit and provide these features to a Logistic Regression Classifier, whose job is to classify the circuit and determine to which type/class this circuit belongs.

The neural network was implemented using Python 3.6, TensorFlow [21] and scikit-learn [22]. TensorFlow was chosen due to the increase in its use and simplified configuration to use the GPU; scikit-learn is a high-level API with a lot of functions for data analysis and many resources to build neural networks. A NVIDIA TITAN V was chosen in order to run our tests and all experiments were executed on the GPU under 11GB of memory usage.

TABLE I: Summary of circuits and the respective number of different CNF-Signatures

Circuit	Class	N. Circuits	N. CNF-Sig.
Ripple Carry Adder	Adder	1,022	5
Ripple Carry Adder/Subtractor		1,022	5
Carry-Lookahead Adder		198	402
Carry-Skip Adder		1,022	13
Carry-Select Adder		1,022	22
Carry-Save Adder		1,022	6
Carry-Lookahead Block Adder		1,022	17
Dadda Tree Multiplier	Multiplier	126	514
Array Multiplier		254	6
Carry-Save Multiplier		254	7
Non-Restoring Cellular Array Divider	Divider	254	7
Restoring Celular Array Divider		254	10
Carry-Lookahead Array Divider		162	492
Decoder	Basic	18	20
Encoder		18	20
Multiplexer		18	20
Demultiplexer		18	20
Parity Circuit		510	3
Magnitude Comparator		510	515
Arithmetic Logic Unit	ALU	126	77
Total		8,852	1,517*

V. Experiments and Results

To perform our experiments, we used benchmark circuits generated by the Bencgen tool [23]. According to the tool's authors, it is possible to generate more than one million circuits among 24 different projects. This tool was chosen because it provides a significant number of circuits, nonetheless the circuits can also be provided in different formats (Bench [24], Verilog [25] and CNF [8]). The current experiment selected circuits generated in two formats: CNF for the detection of CNF-Signatures and the consequent Boltzmann Machine input, and Verilog for the construction of our CNF-Signatures library, since in Verilog the circuits are generated by modules instances.

The first part of our experiments consisted of extracting the Verilog modules present in the generated circuits. For each different module identified, a file was created with the corresponding CNF-Signature, as described in Section III-C. Table I shows the number of different modules identified per type of circuit. It is possible to verify that some modules are composed of few different CNF-Signatures, in general, basic logic gates or small modules such as half-adders or full-adders. However, some types of circuits have considerable amounts of different CNF-Signatures, such as the Magnitude Comparator, whose analysis allowed us to verify that they are basic logic gates with different amounts of inputs. The results of the first part of the experiment allowed us to create a library with 1,517 different CNF-Signatures. The sum of all CNF-Signatures for each type of circuit has a higher value than that, since most modules share several CNF-Signatures.

The next step was detecting the quantity of each CNF-Signature present in each circuit. To perform this task, the CNF-Signatures recognition tool described in Section IV-A

TABLE II: Sample of CNF-Signatures' number detected in Non-Restoring Cellular Array Divider from 3 to 12 bits

Circuit	nBits	Class	NOT_1	AND_2	BUFF_1	XOR_2	OR_2	CFullAdder	CBlock
31	3	Divider	2	19	2	27	9	9	9
31	4	Divider	2	33	3	48	16	16	16
31	5	Divider	2	51	4	75	25	25	25
31	6	Divider	2	73	5	108	36	36	36
31	7	Divider	2	99	6	147	49	49	49
31	8	Divider	2	129	7	192	64	64	64
31	9	Divider	2	163	8	243	81	81	81
31	10	Divider	2	201	9	300	100	100	100
31	11	Divider	2	243	10	363	121	121	121
31	12	Divider	2	289	11	432	144	144	144

TABLE III: Summary of circuits' classes employed in our experiments and their average accuracy

Circuit	Class	N. Circ.	Acc./Circ.	Acc./Grp.
Ripple Carry Adder	Adder	1,022	0.72	0.95
Ripple Carry Adder/Subtractor		1,022	0.69	
Carry-Lookahead Adder		198	0.83	
Carry-Skip Adder		1,022	0.96	
Carry-Select Adder		1,022	0.95	
Carry-Save Adder		1,022	0.80	
Carry-Lookahead Block Adder		1,022	0.92	
Dadda Tree Multiplier	Multiplier	126	0.78	0.92
Array Multiplier		254	0.82	
Carry-Save Multiplier		254	0.70	
Non-Restoring Cellular Array Divider	Divider	254	0.80	0.95
Restoring Celular Array Divider		254	0.72	
Carry-Lookahead Array Divider		162	0.92	
Decoder	Basic	18	0.78	0.88
Encoder		18	0.79	
Multiplexer		18	0.55	
Demultiplexer		18	0.62	
Parity Circuit		510	0.82	
Magnitude Comparator		510	0.74	
Arithmetic Logic Unit	ALU	126	0.75	0.93
Total		8,852	Avg.A 0.783	Avg.A 0.926
			Avg.W 0.821	Avg.W 0.939

was performed with all 1,517 signatures on all 8,852 circuits; producing a matrix of dimensions 8,852 x 1,517. Each cell in the resulting matrix corresponds to the number of instances in which each CNF-Signature was identified on that circuit. Table II shows a sample of a matrix with only ten circuits and seven CNF-Signatures. The first three columns were inserted here only for readability purposes.

Finally, we provide as input to our Neural Network the result of circuit processing. Each CNF-Signature corresponds to a feature that the circuit might present, and the normalization of the value identifies how many of each signature the circuit presents. The presence or absence of each CNF signature allows identifying how much the circuit corresponds to a type of circuit. In our tests, all 8,852 circuits, of 20 different types, generated by the Bencgen tool were used. Circuits were divided in the proportion of 60%/40% for training/testing. Table III shows the results of the experiments on the identification of each type and the class to which it belongs.

Regarding the results provided, we can affirm that the neural network presented is highly precise when the identification corresponds to the class to which the circuit belongs, with an arithmetic average of 92.6 % and a weighted average of 93.9 % relative to its classification accuracy. This precision is due to the fact that the classes of circuits are quite distinct from each other, mainly regarding the types of functional blocks and consequent CNF-Signatures.

Analyzing each type of circuit individually, we identified two types of circuits that presented more difficulties for the neural network, whose performance was bellow our expectation: Multiplexer and Demultiplexer. After deconstructing the steps the neural network took, as well as the input data of these types of circuits, we verified that both look indistinguishable between themselves considering the amount of each CNF-Signatures which each of these circuits present; in other words, the neural network will identify a Multiplexer and a Demultiplexer, with the same number of bits, as equals. Distinction between both circuits is related to the connection of its components, an aspect that is not emphasized in this work.

Continuing the individual analysis of each type of circuit, we also verify that some circuits present higher rates of precision. The Carry-Skip Adder, Carry-Select Adder, Carry-Lookahead Block Adder and Carry-Lookahead Array Divider circuits had rates of 96%, 95%, 92% and 92%, respectively. The evaluation of the intermediate steps revealed this is due to the presence of CNF-Signatures that are only available on these circuits. The Carry-Lookahead Array Divider circuit, for instance, is the only circuit type that has CLABlock's CNF-Signature, as well as the Carry-Lookahead Block Adder that has CLAGenerator's CNF-Signature. In case the identification process was executed through an exact method, there would not be any doubts regarding these results, but the presence of other variables the neural network takes into account, which end up helping in the identification of the less singular classes, we have the ocasional decrease in the accuracy of the above mentioned classes.

In general, the accuracy of the neural network in the classification of circuits is quite high, with an arithmetic average of 78.3% and a weighted average of 82.1%. The results achieved showed that the pre-processing of the CNF file for the extraction of CNF-Signatures allowed an RBM to be used where, in principle, it could not, since the CNF files do not have any structure or information that allow categorization by RBMs.

Associated with the information obtained by the neural network, in most cases, it is possible to determine the size of the detected circuit. This happens because each circuit has an equation of number of occurrences of functional blocks associated with the number of bits in it. Illustrating how the circuit size is determined, let us consider the Non-Restoring Cellular Array Divider circuit; the graph in in Fig.5 demonstrates the occurrence curves of each CNF-Signature by the number of bits.

Once the type of circuit is detected, it is possible to determine its size through its functions. In the case of the Non-Restoring Cellular Array Divider, $OR2(nBits) = nBits^2$, as well as $XOR2(nBits) = 3 \times nBits^2$. The problem with using formulas directly is that the number of combinations and the number of formulas make the exact search exhaustive, in a few cases impossible without adding a degree of uncertainty, as different encodings can generate different values for each CNF-Signature; which makes combinatorial analysis difficult, a task indicated to RBM.

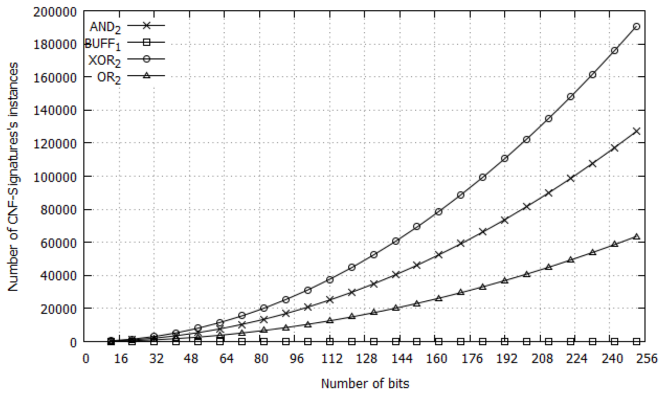


Fig. 5: Occurrences of CNF-Signature per bits in a Non-Restoring Cellular Array Divider

The results showed that our contribution to the feature extraction process for further classification consists of a new structural recovery method, which allows the determination of the type of circuit and its size.

VI. Conclusion

In this paper, we aimed at presenting a novel framework for circuit classification on a Continuous Restricted Boltzmann Machine and Logistic Regression. Our framework is composed of three main components. The first is an algorithm for subgraph isomorphism search of CNF-signatures of logic gates and functional blocks in a graph of a circuit specified in CNF; the result of this operation provides input data to a Continuous Restricted Boltzmann Machine, our second main component, which outputs valuable information to a Logistic Regression Classifier, the last main component of our framework. The latter, in succession, is capable of accurately classifying (over 92%) several circuit classes, in addition to providing its size for arithmetic circuits.

Our results display the benefits for circuit classification of combining the extracted information from a CNF-encoded circuit applied in conjunction with neural networks; in spite of that, the same information could also be employed to generate new circuits using complex functional blocks, which points to a potential new research subject. In conclusion, we intend to continue working to improve our circuit classification framework, as well as propose, in future research, the automation of the recognition processes of circuit size.

Acknowledgment

The authors would like to thank the research agencies CAPES, CNPq, and FAPEMIG for their financial support and NVIDIA for equipment donation.

References

[1] C. M. Li, “Equivalency reasoning to solve a class of hard sat problems,” *Information Processing Letters*, vol. 76, no. 1, pp. 75–81, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020019000001265

[2] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, “Solving difficult instances of boolean satisfiability in the presence of symmetry,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 9, pp. 1117–1137, Sept 2003.

[3] E. Grégoire, R. Ostrowski, B. Mazure, and L. Saïs, “Automatic extraction of functional dependencies,” in *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, ser. SAT’04. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 122–132. [Online]. Available: http://dx.doi.org/10.1007/11527695_10

[4] M. Ciesielski, W. Brown, D. Liu, and A. Rossi, “Function extraction from arithmetic bit-level circuits,” in *2014 IEEE Computer Society Annual Symposium on VLSI*, July 2014, pp. 356–361.

[5] S. Chakraborty, A. Gupta, and R. Jain, “Matching multiplications in bit-vector formulas,” *CoRR*, vol. abs/1611.10146, 2016. [Online]. Available: http://arxiv.org/abs/1611.10146

[6] F. Lu, L. C. Wang, K.-T. Cheng, and R. C. Y. Huang, “A circuit sat solver with signal correlation guided learning,” in *2003 Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 892–897.

[7] J. A. Roy, I. L. Markov, and V. Bertacco, “Restoring circuit structure from sat instances,” 2004.

[8] D. J. Johnson and M. A. Trick, Eds., *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*. Boston, MA, USA: American Mathematical Society, 1996.

[9] G. Tseitin, “On the complexity of proofs in propositional logics,” in *Seminars in Mathematics*, vol. 8, 1970, pp. 466–483.

[10] R. Ostrowski, É. Grégoire, B. Mazure, and L. Saïs, *Recovering and Exploiting Structural Knowledge from CNF Formulas*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 185–199. [Online]. Available: https://doi.org/10.1007/3-540-46135-3_13

[11] Z. Fu and S. Malik, “Extracting logic circuit structure from conjunctive normal form descriptions,” in *20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID’07)*, Jan 2007, pp. 37–42.

[12] H. Seltner, “Extracting hardware circuits from cnf formulas,” Master’s thesis, Johannes Kepler Universität Linz - Faculty of Engineering and Natural Sciences, Altenberger Strabe 69, 7 2014.

[13] A. Biere, “The aiger and-inverter graph (aig) format,” 2007. [Online]. Available: http://fmv.jku.at/aiger/

[14] Y. Dai and R. K. Braytont, “Circuit recognition with deep learning,” in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, May 2017, pp. 162–162.

[15] A. Fayyazi, S. Shababi, P. Nuzzo, S. Nazarian, and M. Pedram, “Deep learning-based circuit recognition using sparse mapping and level-dependent decaying sum circuit representations,” in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019, pp. 638–641.

[16] L. M. Silva, F. V. Andrade, A. O. Fernandes, and L. F. M. Vieira, “Arithmetic circuit classification using convolutional neural networks,” in *2018 International Joint Conference on Neural Networks (IJCNN)*, July 2018, pp. 1–7.

[17] V. Falco, “Vflib: A multipurpose library built on juce,” https://github.com/vinniefalco/VFLib, 2008, an implementation of VFlib available at github.

[18] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, “Performance evaluation of the vf graph matching algorithm,” in *Proceedings 10th International Conference on Image Analysis and Processing*, Sep. 1999, pp. 1172–1177.

[19] R. Salakhutdinov, A. Mnih, and G. Hinton, “Restricted boltzmann machines for collaborative filtering,” in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML-07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 791–798. [Online]. Available: https://doi.org/10.1145/1273496.1273596

[20] Boost, “Boost C++ Libraries,” http://www.boost.org/, 2015.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[23] F. V. Andrade, L. M. Silva, and A. O. Fernandes, “Bencgen: a digital circuit generation tool for benchmarks,” in *SBCCI ’08: Proceedings of the 21st annual symposium on Integrated circuits and system design*. New York, NY, USA: ACM, 2008, pp. 164–169.

[24] F.Brglez and H. Fujiwara, “A neutral net list of 10 combinational benchmark circuits and a target translator in fortran,” in *International Symposiumon Circuits and Systems*, 1985, pp. 695–698.

[25] “Standard for verilog hardware description language,” pp. 1–590, April 2006.