Logic Regression about Circuit Learning

Sung Che-Kuan
Dept. Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan
b05902045@ntu.edu.tw

Wu Yu-Tsung
Dept. Electrical Engineering
National Taiwan University
Taipei, Taiwan
r09943111@ntu.edu.tw

***Abstract*—An unknown system with large input space, it is difficult to identify all inputs and outputs relation. Since the set of input pattern grows exponentially as input number increases. Sampling is a better method instead of exploring all possible input pattern. Sampling results usually give us some properties of the unknown system.**
We will talk about algorithm which champion of *2019 ICCAD Contest* use. And give some discussion about this winning approach. Based on the wining algorithm, we proposed some improved method help increase accuracy or decrease execution time.

I. Introduction

An unknown system with large input space, it is difficult to identify all inputs and outputs relation. Since the set of input pattern grows exponentially as input number increases. Sampling is a better method instead of exploring all possible input pattern. Sampling results usually give us some properties of the unknown system.

Boolean model may be a good model to represent the input-output relations of the unknown system. There are lots of application use Boolean model, for instance, data analysis [1], reverse engineering [2], model verification [3], testing [4]. Furthermore, Boolean circuit is often semantic expression. Not only human can quickly understand the relation of unknown system. It is also helpful for tool doing optimize based on the result of transformation from unknown system to Boolean circuit.

In ICCAD 2019 CAD Contest Problem A [5]: Logic Regression on High Dimensional Boolean Space, the problem asks to find the input-output relation based on an unlimited-try black-box, in where the relation is completely specified Boolean function $f : \mathbf{B}^{|I|} \rightarrow \mathbf{B}^{|O|}$ The target of the problem aims to produce the Boolean Circuit whose accuracy over 99.99% with each Black-Box. The Boolean Circuit produced for this problem is composed of 2-input primitive gate. Besides high accuracy, the number of 2-input primitive gate need to be minimized.

From problem description, the input size ranges from 25 to hundreds. The upper bound of output size.

This work is based on the framework of [6]. In section II, we briefly give a fast review of this related work. In section III, we discuss about the advantages and drawbacks of the method in that work. In section IV and section V, we will introduce some new methods we proposed. Experimental result is shown in section VI. The last, we make a conclusion in section VII.

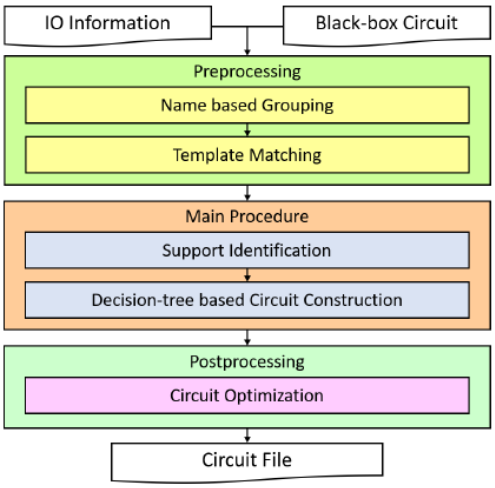

Fig. 1: the frame work of []

II. Related Work

The work [] can be summarized into three part. First is preprocessing, second is the main algorithm, the last part is optimization about Boolean circuit.

A. Preprocessing

Preprocessing contains Grouping and Matching based on the information of input and output names. Consider the name of inputs and outputs, the common string of different PIs or POs often shows that they have some certain relation in the industrial design. Assume relation exists and construct corresponding group for each sub-string. For each group, use a vector V and represent a number N.

After the grouping, matching starts from any two vector, finding some relation between two N, shown in table(). The target is finding a internal variable O_s that represent the certain relation of N_{v1} and $N_{v2}$is satisfying or not. If such a internal variable O_s exists and it can be observed on some

outputs by assigning other inputs. Assigning the O_s instead of assigning $v1$ and $v2$ has more efficiently since the search space decreases.

B. Support Identification

The main algorithm consists of two part, one is Support Identification, the second is Decision-tree based Circuit Construction. Instead of searching the whole possible combination of input, only focus on the support set of an output gives us less effort and time on both sampling and optimization. Identify the exact support of an output is hard and need lots of entry. Hence, good sampling is needy that we can roughly see the relation. Consider an primary output O, randomly generating a set of input patterns S, for each pattern we flip a certain bit to generate the set S' of pair which is used to test this PI is the support of O or not. If there are different result O appears in any pair of S', this PI is added to the support of O. If the $|S|$ is big enough that the rest part of true support can be ignored. Since the error can be endure .

C. Decision-tree based Circuit Construction

Since there exists optimization at the end, we divide the circuit by each output and its corresponding support. For each output, there are a binary decision tree used to find the onset. The procedure is similar to Support Identification. At each node, the decision criterion is one variable, the left child and right child are representing the '0' and '1' of that variable. From root we extract free variable at each node from support set, we fix the value of extracted variable. And the parent node set of fixed variable is inherited to each child. At each node, do sampling the same as the method in support identification but there are fixed variable which need to be assigned to the fixed value.

D. Optimization

After simulation, pick the variable which has the most number of pair which have different output value. We call the variable is most frequent variable. Use the most frequent variable to divide the support, until the simulation return constant value. After all the leaves are found or the time limit is reached, we collect the leaves with one constant value to form a POS (product-of-sum) form function.
The last part is optimization, they use *abc* tool to help optimizing.

III. Discussion

As naming information seems not so generally that can't be always applied to each Black-Box problem. We want to focus on the input and output relations only provided by the Black-Box since it's more reliability instead of the names.

The decision tree is a kind of binary tree that we wan to find some topological order of variables so that the search space can be reduced when we search from that kind of order. But sometimes the binary classification has its restriction when it's applied to this problem. It is easy to find the significant variables and the cube contains such variables. We call such cubes as significant cubes which often contain less variables. Variable in significant cubes tends to be the frequent variable since the significant cube is easily to be largely approached by random assignment. Based on large number of patterns meet the significant cube, the variations of corresponding variables can be observed quickly.

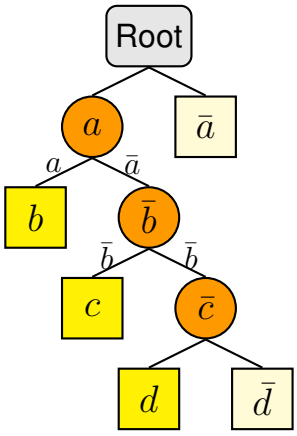

Fig. 2: Decision tree example: $F = ab + ac + ad$

Multiple entries to the simulation, IO-generator, take lots of time to read input and output. If the simulation need nested relation that binary search may cause deep and nested simulation result. Consider one example $F = ab + ac + ad$ shown in figure 2. If we use the binary search we need five levels to fulfill the search. If we don't limit the number of children of each node or loose the limit can be more than two. We may fulfill the search at third level (figure 3).

Consider another example $F = ab + efgh$ shown in figure 4. We start from a and find the cube ab at second level. Find $aefgh$ at the sixth level below the a node. Find $\bar{a}efgh$ below the $\bar{a}$ node. It shows that the smaller cube needs more simulation times than the bigger one. If the smaller cube doesn't contain the most frequent variable of Root node, or the first variable which belong to that cube appears in deep layer, there can be much effort spending on search the cubes which can be merged to a bigger cube.

IV. Improved Method

We will introduce our new methods including some improvement about decision-tree based method.

A. Two-Variable Decision Tree

Instead of one variable criteria at each node, use the top two frequent variables as decision criteria. For example, if the a, b is the top two frequent variables of current node I. There are four children extended by node I, the fixed variables

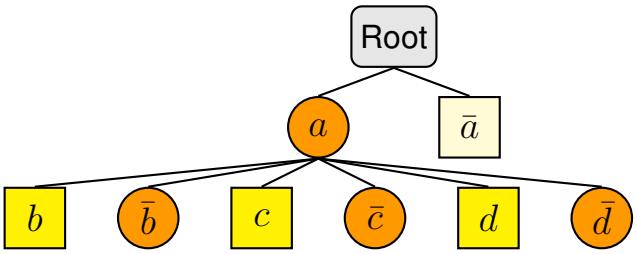

Fig. 3: Multiple children Decision tree example: $F = ab + ac + ad$

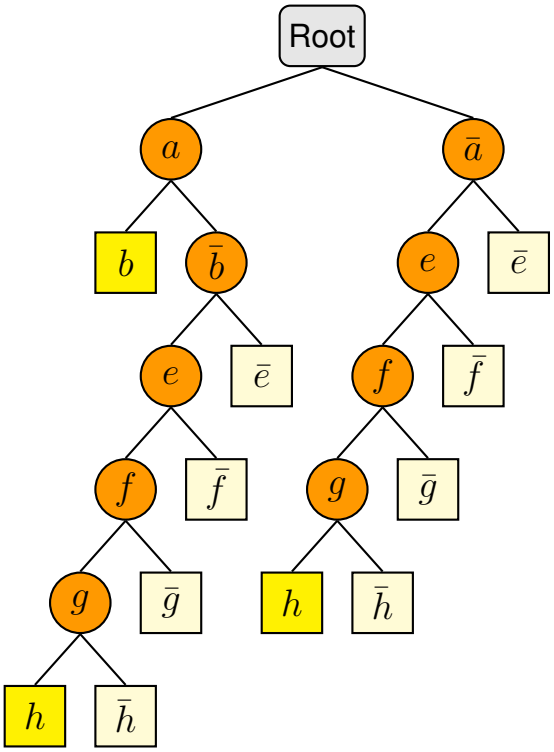

Fig. 4: Multiple children Decision tree example: $F = ab + efgh$

appended to each child are respectively $ab, \bar{a}b, a\bar{b}, \bar{a}\bar{b}$. And we can extend the method use the most frequent variable and the least frequent variable.

B. Min tree add Max tree

Construct two decision tree, one decision criteria uses the most frequent variable, the other use the least frequent variable. Notice that the least frequent variable is still has variation.

C. New variation support

For proceeding the time for search, at each node we first inherit the support set from its parent excluded the fixed variable at this node. When we do simulation that test which variable in support set is the most frequent variable. We drop the variables which are zero variation in the simulation result.

V. Single pattern flipping

We find that the information get from each simulation we did before may be so discrete that we only get frequency about variables. We want to get more useful information in each round of simulation.

First we also use the support identification method to find the support. When detect the support for output O, for each single pattern for flipping inputs we construct a group contains the variable which has different results on O.

A. Division

Since each Boolean function can be expressed as SOP form. We can express a function $F = \sum_{i=1}^{n} c_i$ where each c_i is a product (cube) of variables. When a pattern P_j satisfies F, it also satisfies some cubes of F. We call this set of satisfied cubes as C_j. Now when a variable k is flipped on P_j forms pattern P_{jk} which results in F turning into 0 value, the result of flipping on k shows that k is contained in all cube in C_j. The total variables which meet the flipping in output form a set S_j. The S_j can be seen as a cube. But it's not ensure that it belongs to F. For example $F = abc + c'd, P_1 = abcd$. Then we find $S_1 = ab$. We want to use these cubes to further find the smaller cube.

Based on the S_j, we move into deep search space. We want to find the cubes contained by S_j satisfy F. Fix S_j and continue do the support identification. Since F can be expressed as $G * H + R$. Let S_j be the G and we want to find H. Because the cube S_j is fixed, H then is independent of the variables in S_j. If the support set of H is small enough that we can do the brute force. Otherwise, we recursively do single pattern flipping by finding the new S under S_j.

After finding H for S_j, we move on to next S. So that we express F as $\sum_i G_i * H_i + R$, the smaller R is better. Since the summation can act like F. Now the problem is how to select good S.

B. Grouping

After doing several times $SinglePatternFlipping$, we need to distinguish which S is good to further explore. First we sort the set of S which called S_{set} by the variables number of each S and pick the high variables part of S_{set} to do grouping. The concept of grouping is that there may be some S_1, S_2, S_3 cubes are similar to each other. For instance, $abcd$, $abcf$, $abch$ are distinct to each other on two variable, missing one and existing extra one. The common cube is abc. We collect such cubes to form a $Group\ GP$ and use a cube to represent the GP.

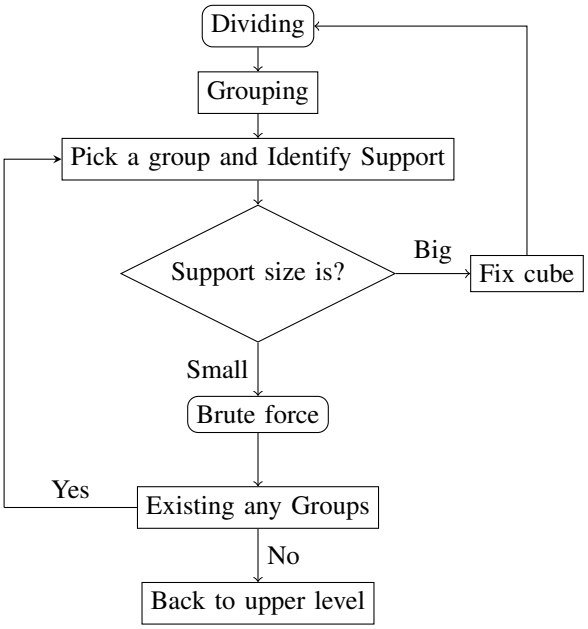

Fig. 5: Single Pattern Flipping Detailed flowchart:

For each primary input PI_1, count the most frequently appear value. Use such value to represent PI_1 in this GP. Notice that there are three values might appear in one pattern, 0, 1, $-$. 0 means the inverse of PI_1, 1 means PI_1, $-$ means both 0 and 1 are accepted. If three or two values both have the highest frequently appearance. The table.1 shows the decision criteria.

TABLE I: Grouping criteria

Grouping variable	
The most frequent value	represent value
0,-	0
1,-	1
0,1	-
0,1,-	-

C. Overlapping

When doing the recursion step for each GP we might find some cube in R if we perform weak division. Such cube is independent of GP. It doesn't reduce the accuracy but it increases the time and search effort. For example, $F = abce + c'd + e'f + gh$, the cube here we fix is ab, but with pattern $abc'd'e'f'gh$, we found both g and h cause variation to F on this pattern, where origin F is 1 but F turns to 0 when g or h is flipped.

VI. Experimental Result

We use C++ as the software language to implement our algorithm. We focus on the NEQ and ECO two types of IO-generators. Since it is mentioned [6] that this two types problems have the same difficulty with and without data pre-processing. We randomly generate the one hundred thousand input patterns. And see the outputs are correct or not. Once an output is wrong, the corresponding input is classified as the wrong pattern. Total wrong pattern number is W. Accuracy $= 1 - \frac{W}{10000}$. The table. II shows our result about time and accuracy verse each case. The speed is quite fast, but the accuracy need some improvement. We think the support set is important part to improve the accuracy. We do less in single pattern flipping part. One is we don't generate the unbalanced patterns which have more '0' bits or '1' bits. Two is we use origin support not all primary inputs.

TABLE II: Experimental result

Case	Type	Accuracy	time	circuit size
1	ECO	1.0	35	170
4	ECO	0.991240	39	338
5	NEQ	0.9813	54	138
7	NEQ	1.0	5	40
9	ECO	—-	—-	—-
10	NEQ	1.0	5	31
11	NEQ	—-	—-	—-
13	ECO	1.0	5	27
14	NEQ	—-	—-	—-
17	ECO	0.995760	794	—-
18	NEQ	—-	—-	—-
19	ECO	—-	—-	—-

TABLE III: Result [6]

Case	Type	Accuracy	time	circuit size
1	ECO	1.0	35	165
4	ECO	1.0	229	173
5	NEQ	0.99833	2578	1436
7	NEQ	1.0	5	40
9	ECO	—-	—-	—-
10	NEQ	1.0	6	23
11	NEQ	0.99640	2657	1928
13	ECO	1.0	5	27
14	NEQ	0.28194	2689	11207
17	ECO	0.99989	1983	2598
18	NEQ	0.59757	—-	3391
19	ECO	0.99956	1764	2991

VII. Conclusion

Our algorithm provide a fast model to effectively use the simulation results. Since decision-tree is not scalable, the run-time may grow up exponentially. The work [6] use data-preprocessing to reduce the input space. We use single-pattern flipping want to do grouping among input Variable. In the future, we want to apply more grouping strategy to help decrease the run-time.

References

[1] I. Ruczinski, C. Kooperberg, and M. LeBlanc, "Logic regression," *Journal of Computational and graphical Statistics*, vol. 12, no. 3, pp. 475–511, 2003.

[2] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascón, W. Y. Tan, A. Tiwari, N. Shankar, S. A. Seshia, and S. Malik, "Reverse engineering digital circuits using structural and functional analyses," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 63–80, 2013.

[3] S. Robinson, "Simulation model verification and validation: increasing the users' confidence," in *Proceedings of the 29th conference on Winter simulation*, 1997, pp. 53–59.

[4] B. Beizer, *Black-box testing: techniques for functional testing of software and systems*. John Wiley & Sons, Inc., 1995.

[5] C.-Y. Huang, C.-A. R. Wu, T.-Y. Lee, C.-J. J. Hsu, and K.-Y. Khoo, "2019 cad contest: Logic regression on high dimensional boolean space," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–6.

[6] P.-W. Chen, Y.-C. Huang, C.-L. Lee, and J.-H. R. Jiang, "Circuit learning for logic regression on high dimensional boolean space," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.