

1/31/2022

DLD PROJECT PROPOSAL

Minimization of Boolean Expression

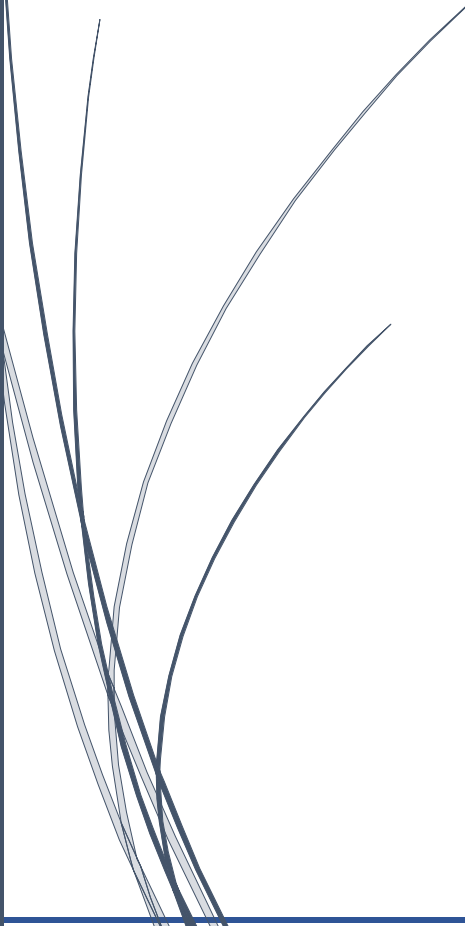
Group Members:

CE-42-A

Amina Qadeer

359607

Submitted To: Lt Col Hammad Tanveer

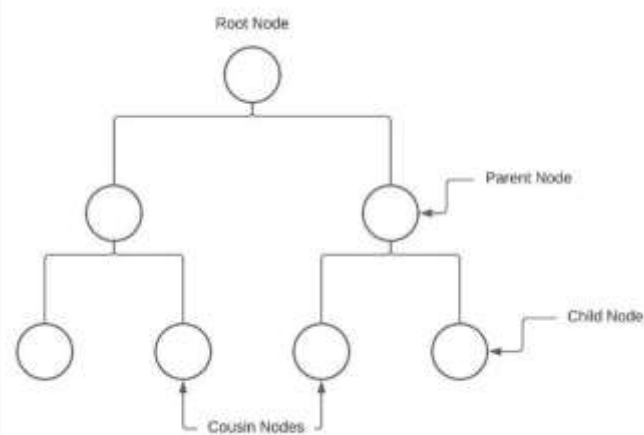


AUTOMATION OF BOOLEAN EXPRESSION MINIMIZATION

Abstract—Binary Tree is a graphical representation that looks like a tree with roots and branches; it played a key role in Boolean expression's minimization which led to a reduced expression after elimination of many redundant branches. Our methodology uses the relation between sibling nodes of a binary tree to minimize the expression down to its simplest form.

INTRODUCTION:

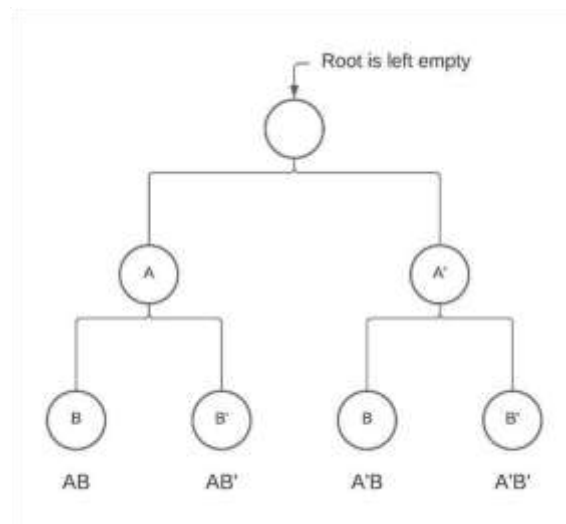
Our work includes the use of binary tree mapping as the basis of the minterm minimization algorithm. This method comprises of a relationship between outputs (0 or 1) and inputs (x_1, x_2, x_3, \dots) which could be characterized with Boolean function and by utilizing different Boolean algebra techniques. It is possible to reduce and simplify the Boolean formula of the system to a reduced form that meets the same relationship between inputs and outputs.



Some terminologies: -

Sibling Nodes: Nodes originating from the same parent node are called siblings.

Cousin Nodes: Nodes whose parent nodes are sibling nodes are called cousins.



Leaf Nodes of a complete binary tree provide us with all possible minterms for the given number of variables. This is similar to the concept of decoders in digital circuits.

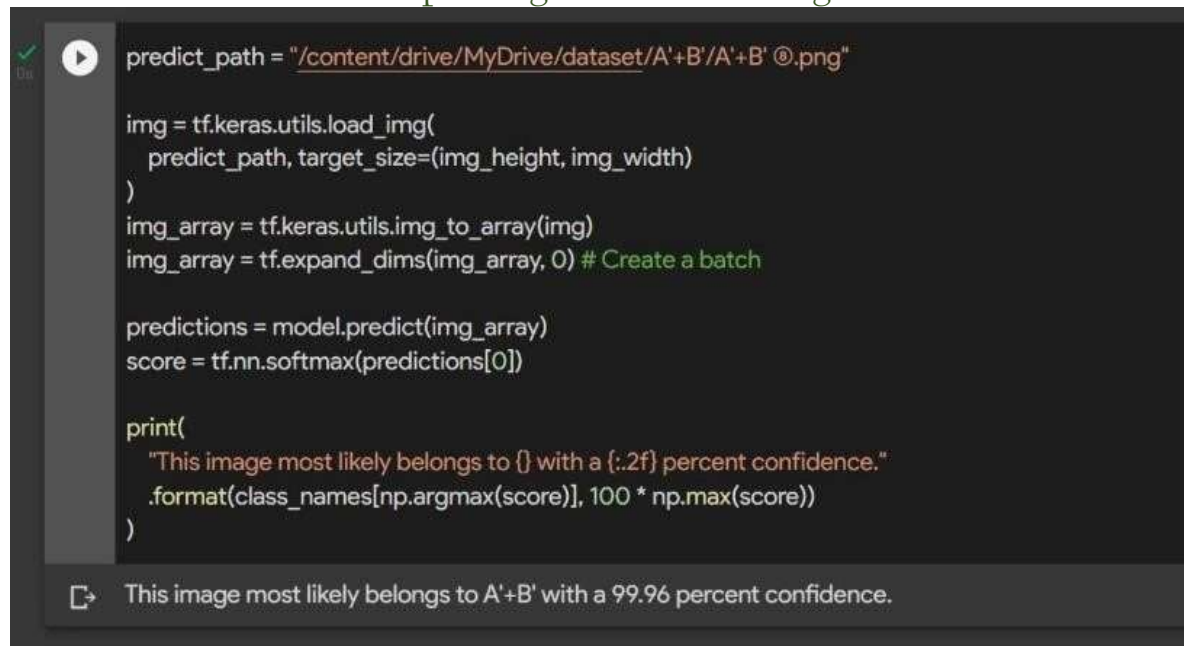
RELATED WORK:

- Tabular method of minterm minimization.

Drawbacks:

1. Although this algorithm is scalable, time consumption increases exponentially for an increasing number of variables.
2. Produces excess terms that should have been eliminated.

- Automation of K-Maps using Machine Learning.



```
predict_path = "/content/drive/MyDrive/dataset/A'+B'/A'+B' @.png"

img = tf.keras.utils.load_img(
    predict_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

➞ This image most likely belongs to A'+B' with a 99.96 percent confidence.

Drawbacks:

1. It isn't scalable because it needs a dataset for training. For every extra variable, it needs an exponentially larger dataset.
2. Karnaugh Maps are only feasible for up to 6 variables.

<https://colab.research.google.com/drive/lsxHNGtApoO9WwKoMVLG90-e4riL647kx>

METHODOLOGY:

Steps to use Binary tree algorithm:

1. Write all minterm from the truth table given.
2. Draw a binary tree of the following order: A (Root), B (level 1), C (level 2) so on, depending upon the number of variables.
3. Assign all minterm truth values from the output column of the truth table to the leaf level of the tree accordingly.
4. Now depending upon the following patterns eliminate the 0 terms and look for “No effect” causing minterm.
 - Between Siblings:
 - 1 1 (At the leaf level eliminates the respective minterm, reducing one level with minterm 1) 0 0 (Eliminate)
 - 0 1 (Eliminate)
 - 1 0 (Eliminate)
 - Between Cousins:
 - 1 0 1 0 (Reduction rule: Swap two levels to obtain “No effect” minterm.)
 - 0 1 0 1 (Generating a derived Tree from Main Tree, Hence Working upon a new order of Tree).

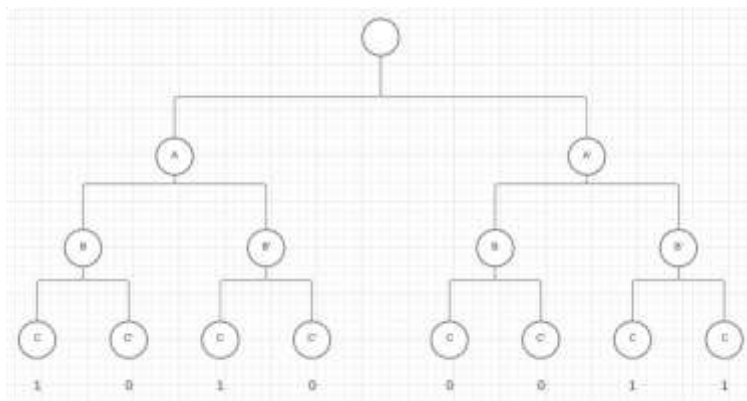
We can have orders containing levels like:

- ABC • CAB
- BCA

Note all variables get a chance to be at leaf level minterm at least once.

Once a tree has been reduced and has given us a minimized term, we cross off all the minterms that the minimized term covers. This prevents the algorithm from giving repetitive terms and stops the algorithm once all the minterms have been covered.

EXAMPLE 1:



A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

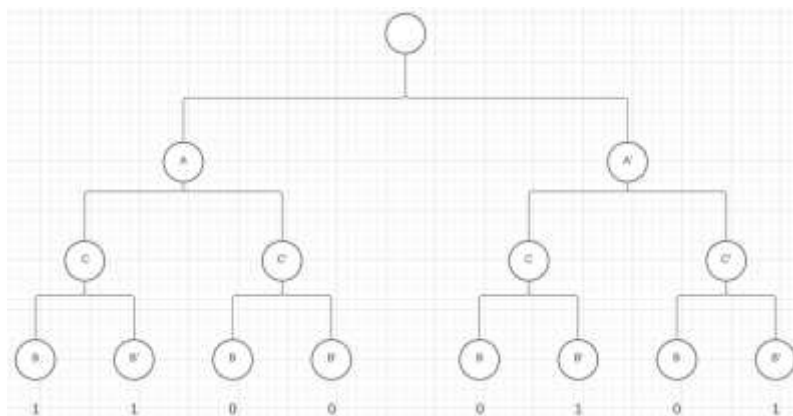
The tree above contains 3 variable inputs. We see two subtrees from A' . Here the patterns identified are 0 0 and 1 1.

Eliminating the leaf nodes gives us $B = 0$ and $B' = 1$. Hence, giving us the term $A'B'$, covering the minterms $A'B'C$ and $A'B'C'$. These minterms can be crossed off the list.

Pattern 1 0 1 0 from the left subtree of parent A. This gives a derived tree with B and C swapped.

The new tree gives us the 1 1 pattern at the subtree of A. After the elimination of leaves, the tree will give us another term AC , covering ABC and $AB'C$. These minterms can be crossed off the list.

~~$A'B'C'$~~
 ~~$A'B'C$~~
 $AB'C$
 ABC



~~$A'B'C'$~~
 ~~$A'B'C$~~
 ~~$AB'C$~~
 ~~ABC~~

All the terms have been crossed off; hence no more trees are required, and our minimization is complete. The output of our process is $A'B' + AC$.

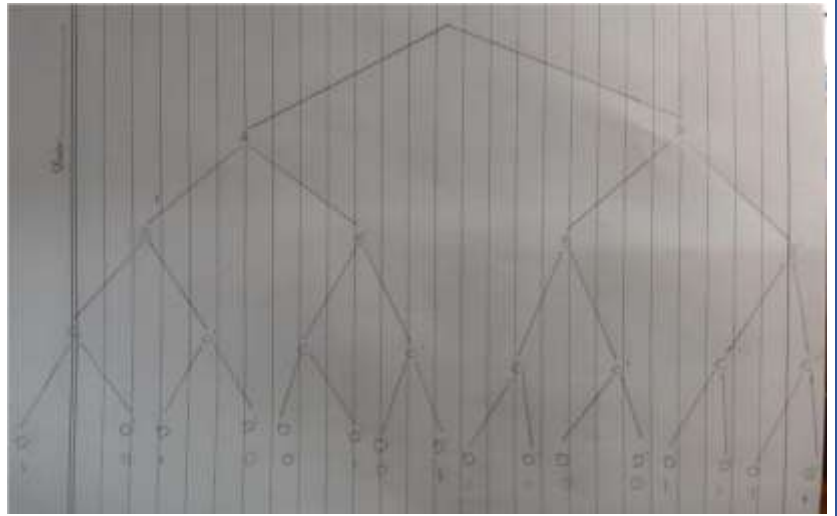
Using the same minterms with K-Maps gives us the same results as shown in the figure below.

	$\overline{B}\overline{C}$	$\overline{B}C$	BC	$B\overline{C}$
\overline{A}	1	1	0	0
	0	1	3	2
A	0	1	1	0
	4	5	7	6

EXAMPLE 2:

A	B	C	D	X
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

$A'B'C'D'$
 $A'B'C'D$
 $A'B'CD'$
 $A'B'CD$
 $A'BCD'$
 $A'BCD$
 $AB'C'D'$
 $AB'CD'$
 $ABC'D$
 $ABCD$



The first tree minimizes the minterms down to $A'B' + A'BC$, covering all $A'B'$ and $A'BC$ terms that can be crossed off.

	CD	CD	CD	CD
AB	1	1	1	1
AB	0	0	1	1
AB	0	1	1	0
AB	1	0	0	1

- **SCALEABILITY** - (workable till n variable)

This method of binary tree reduction satisfies for all n variables minterm trees can easily extend downward by the increase in variables, following a similar procedure. Hence it is scalable.

- **PERFORMANCE** - (Logical Minimization and identifying all possible paths without duplication)

The problem is in finding the best possible minimization when dealing with 4 variables or more. We lack to identify the best order of the tree that can lead to the best possible minimization with an accuracy of the desired result. (Reduced minterm).

- **COMPUTATIONAL COMPLEXITY** - (Mapping out all Literal possibilities and the designed algorithm will choose out required literal)

There is a working principle of optimization for reducing the number of tree diagrams. Otherwise, with the increase in the number of variables, there is an exponential increase in the number of tree diagrams.

Conclusion:

The K-Map algorithm in python worked well up to six variables. The issue encountered in scalability. The training part of the dataset once achieved, takes no time in execution. K-Map algorithm can work well up to 4 variables once the dataset has been provided to train the model. Time taken to train the model is the only factor in the time parameter because once the model has been trained it can give results instantly. However, the algorithm is not scalable as K-Maps can only work up to 6 variables.

The tabular method works well for up to 10 variables, but it isn't heuristic. And increasing the number of variables increases the operations exponentially. This makes it unviable.

The binary trees are scalable but take an exponentially increasing time for an increasing number of variables. However, optimizing the algorithm stops the algorithm before it makes all possible trees. The best possible minimization can only be achieved if the proper order of trees is known, which the algorithm lacks for now.

THE END