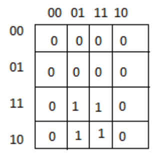
Branch Prediction by Learning Conjunctinos

***Abstract*—In this paper, we investigate a new branch predictor that operates by learning a k-term DNF per branch. This contrasts the behavior of most modern global branch predictors that have separate predictions for each history a branch may encounter. When the branch history register (BHR) length equals 4, our branch predictor significantly out performed g-share. However, as the BHR length increased, g-share significantly outperformed our predictor. We analyze these results and draw conclusions as to why it may not be practical to generate predictions on a per branch basis.**

I. Introduction

Branch prediction has been a bottleneck in exploiting instruction level parallelism in modern pipelined processors. Generally, branch predictors use information such as the history of previous branches (global behavior) or the history of each independent branch (local behavior). In this paper, we attempt to capture the local and global behavior of branches by generating a k-term DNF per branch, where a k-term DNF is a disjunction of k conjunctive clauses. To gain some context about this idea, let us examine some hypothetical data for a particular branch. The format of the data is $< b_i, y_i >$ where b_i is the branch history and y_i is 1 if the branch is taken and 0 otherwise.
$< (1, 0, 0, 1), 1 >$
$< (1, 0, 1, 1), 1 >$
$< (1, 1, 0, 1), 1 >$
The goal is to learn a conjunction consistent with the data In order to do this, we initialize our prediction to b_0, delete the third term after examining b_1, and delete the second term after examining b_2, leaving us with the conjunction $x_0 \wedge x_3$. From a graphical perspective, our prediction generates the following results:



	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	1	0
10	0	1	1	0

Note that the data point $(1, 1, 1, 1)$ will be predicted as taken even though we did not explicitly observe the data point. Let us now add some more hypothetical data:
$< (0, 0, 0, 1), 0 >$
$< (1, 0, 1, 0), 0 >$
Our prediction also consistent with b_3 and b_4 because $x_0 = 0$ for b_4 and $x_3 = 0$ for b_5. Let us now add some more interesting hypothetical data:
$< (0, 1, 0, 0), 0 >$
$< (1, 1, 1, 0), 0 >$
$< (0, 1, 1, 0), 1 >$
Note that for b_5, b_6, and b_7 our conjunction will mispredict because both x_0 and x_3 do not equal 1. If we update our predictor by deleting all candidate terms (x_0 and x_3) we would have erased our entire prediction. Thus, there does not exist a single conjunctive term consistent with the data. The solution to this issue of adapting to inconsistent data points lies in adding a second conjunctive term. By following the same logic as we did with the first three data points, we can generate a conjunction consistent with the last three data points $x_2 \wedge \neg x_3$. Combining our two results, we end up with a 2 term DNF $(x_0 \wedge x_3) \vee (x_2 \wedge \neg x_3)$ which is consistent with all the data points.

II. Design and Implementation

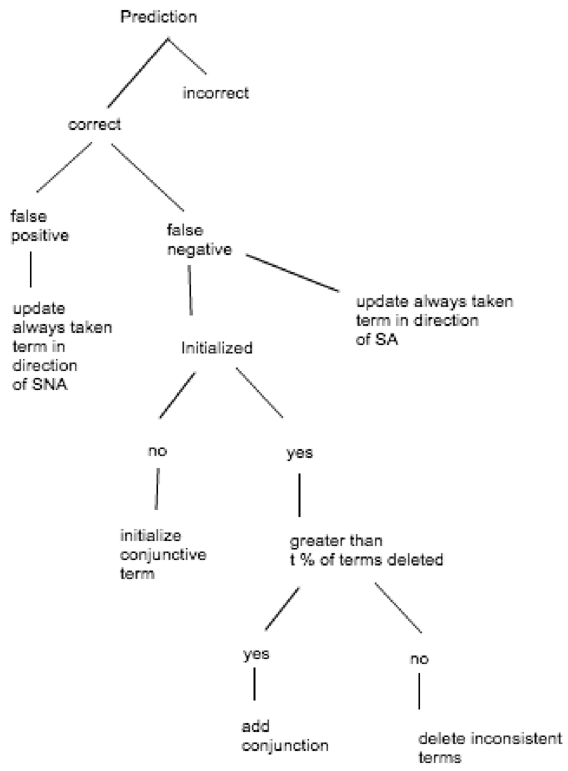
Let us generalize the above phenomenon of being able to represent data in the form of $< b_i, y_i >$ for a particular branch by a k-term DNF to a formalized branch prediction algorithm. We shall begin by looking at the updating prediction logic. A misprediction can be characterized into two classes: false negatives and false positives.

On a false negative, if this is the first time we mispredicted the branch, we set the first conjunction equal to the current branch history. If this is not the first time we mispredicted the branch and we have not deleted more than t percent of the terms where t is some threshold (in our predictor t was chosen to be 70), delete any inconsistent terms with the current branch history. Deleting a term means that if the ith term of the branch history does not equal the ith term of the conjunction, replace the term with a -1. So if the first conjunction is $< 0, 1, 1, 0 >$ and the branch history is $< 0, 0, 1, 1 >$, the first conjunction would be transformed to $< 0, -1, 1, -1 >$. Note that when we delete inconsistent terms from a conjunction, we never delete all terms from a conjunction because that would erase all work done. If we have deleted more than t percent of the terms from the current conjunction (implying that the current conjunction cannot represent anymore direction of branches based on branch histories), we add a new conjunctive term and initialize the conjunction to the current branch history. Note that from now on, when we delete inconsistent terms, we delete from the most recent conjunction because the previous conjunctions have theoretically been exhausted.

There remain two issues with this current algorithm. The first issue is determining when to stop adding conjunctions. In other words, in our k-term DNF, what should the value of k be? After evaluating the performance of the branch predictor on various values of k, it was found that for branch history lengths greater than 10, values of k greater than 2 did not lead to any significant gains in performance and if anything, hurt performance due to overfitting of data. The second issue arose after running experimental tests on our branch predictor and observing the proportion of false negative to false positives: FP-1: 6, FP-2: 14, FP-3: 100+, FP-4: 100+, INT-1: 4, INT-2: 2, INT-3: 9, INT-4: 6, INT-5: 100+.

This data indicates that the 2 term DNF generated per branch is unable to encompass all data points that are taken. Looking

at some hypothetical data, similar to the one presented in the introduction, to illustrate why this is the case:
$< (1,0,0,1),1 >$
$< (1,0,1,1),1 >$
$< (1,1,0,1),1 >$
$< (0,0,0,1),1 >$
$< (1,0,1,0),1 >$
After examining the third data point, we are left with the conjunction $x_0 \wedge x_3$. However, this conjunction mispredicts the 4^{th} and 5^{th} data points and no terms can be deleted from the conjunction to generate the correct prediction for the 4^{th} and 5^{th} data points. Also, assume our k-term DNF already has k terms, so we cannot add another conjunction to fit these new data points. To adapt to this situation, a term that predicts always taken is separately added to the prediction. The term can be in 4 states: strongly not activated (SNA), weakly not activated (WNA), weakly activated (WA), strongly activated (SA) and the prediction uses the always taken term when it is in the SA state. On a false negative, the always taken term goes to the next state in the direction of SA and on a false positive the term goes to the next state in the direction of SNA. To summarize the update prediction logic [1]:


Prediction
correct
incorrect
false positive
false negative
update always taken term in direction of SNA
update always taken term in direction of SA
Initialized
no
yes
initialize conjunctive term
greater than t % of terms deleted
yes
no
add conjunction
delete inconsistent terms

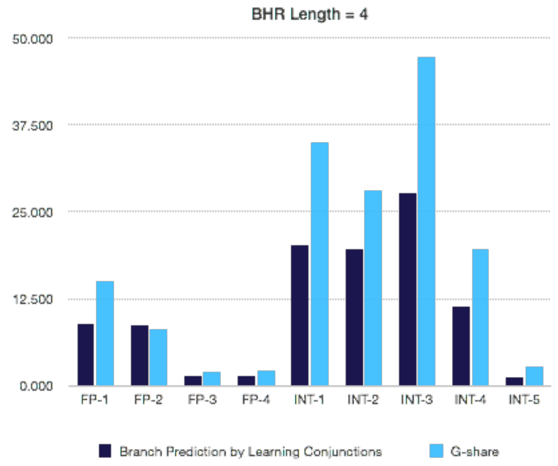
In order to make a prediction, if the always taken term is not activated, we compare the 2 term DNF to the branch history. We predict taken if, for either conjunction, each term that is relevant in the current conjunction equals the corresponding term in the branch history. If the always taken term is strongly activated, we predict taken. More formally, if we denote conjunction one as c_1, conjunction two as c_2, the branch history as b, and m as the number of bits in the branch history, the algorithm to make a prediction is:

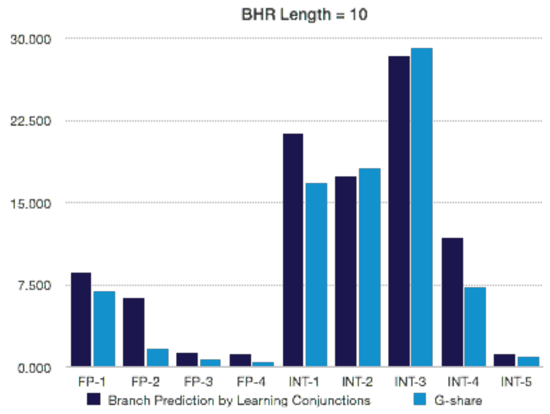
```
prediction = false
**if** always taken term is strongly activated **then**
    return true
**end if**
**for** $j \leftarrow 1, 2$ **do**
    **for** $i \leftarrow 0, m-1$ **do**
        **if** $c_j[i]$ is relevant **then**
            **if** $c_j[i] \neq b[i]$ **then**
                break
            **end if**
        **end if**
        **if** $i = m-1$ **then**
            prediction = true
        **end if**
    **end for**
**end for**
return prediction
```

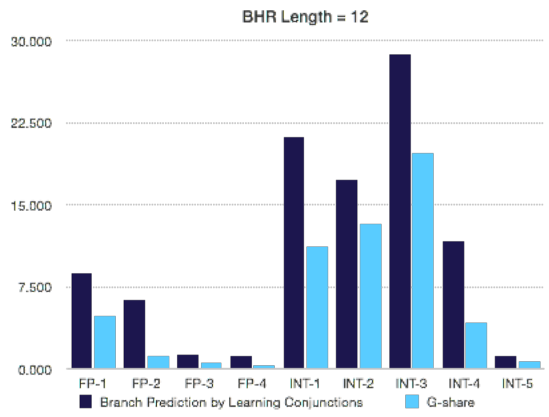
[1]correct and incorrect switched

RESULTS

The following bar graphs display branch misprediction percentages for certain traces as a function of BHR length. The traces were provided by the championship branch predictor framework.



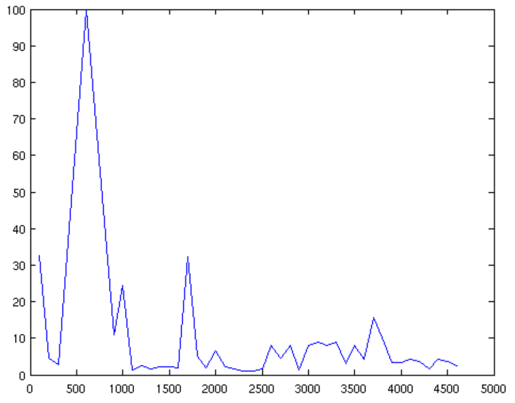


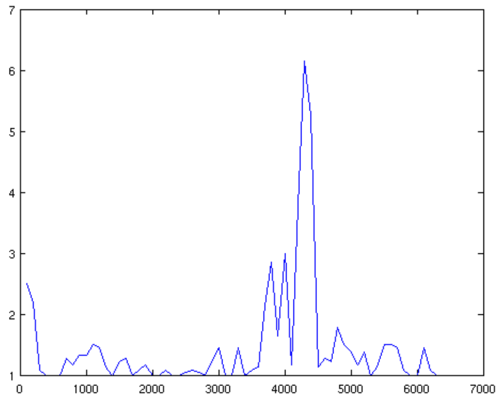


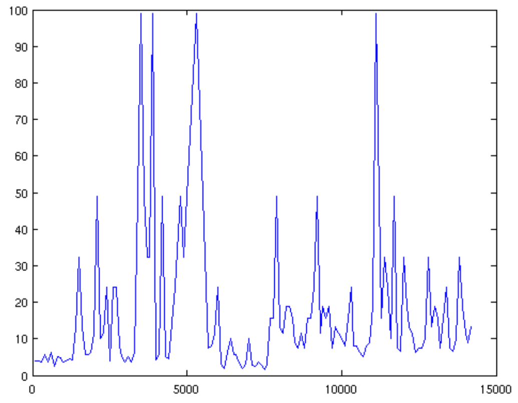
Discussion

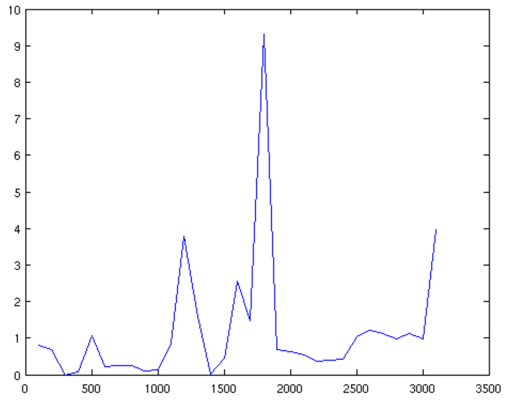
As we can see from the results, while the branch predictor trained by learning conjunctions does beat g-share when the BHR length = 4, gshare significantly outperforms as the BHR length increases. It is well known that g-share's performance dramatically improves as the BHR length increases. The predictor trained by learning conjunctions does not see this same increase in performance because the predictor is trying to create a prediction per branch, meaning it is trying to encompass the outcome of all branch histories in two conjunctions and an always taken term. Thus, as the branch history length increases, there are exponentially more possible branch histories, making it difficult for two conjunctions to encompass all the data.

The main flaw with this branch predictor is its ability to adapt to data points inconsistent with previous data. As can be seen from the following graphs for four different branches that plots the number of times the branch was not taken divided by the number of times the branch was taken every 100 iterations, the behavior of a branch is constantly changing.









Thus, the key to any branch predictor lies in its ability to adapt to new trends in data. Our branch predictor's main adaptive feature lies in the 4 state always taken term, but as the results indicate, this is not as effective as indexing into a pattern history table indexed by a combination of the program counter and branch history. Future work on this predictor may explore different and more effective implementations in adapting to new trends in data.

Acknowledgment

I would like to thank Prof. Craig Zilles for his time and his invaluable feedback and insights.