Branch Prediction with Bayesian Networks

Jeremy Singer, Gavin Brown, and Ian Watson

University of Manchester, UK

Abstract. This paper studies the architectural problem of *branch prediction*. We analyse the popular technique of *two-level adaptive prediction*, relating it to the state-of-the-art Machine Learning technique of *Bayesian Networks* (BNs). We show that a two-level predictor is an approximation to the BN formalism. This link allows us to explore the wider family of BN predictors. We investigate how to adapt BN techniques to operate within realistic hardware constraints, using the same primitive components that are present in existing branch predictors. We systematically study how performance is affected by these simplification. We aim to use these ideas to reduce the storage overhead of BN predictors without losing significant prediction accuracy. The key motivating factor is that storage required in two-level predictors grows exponentially with branch history length, whereas BNs provide a framework to reduce this overhead.

1 Introduction

There is an increasing trend to apply machine learning (ML) techniques to diverse prediction problems in computer systems. Recent examples include architectural simulation [1] and operating system message passing [2]. A challenging case appears to be the online learning problem of *branch prediction* since it requires (1) high accuracy, (2) low latency and (3) low implementation complexity.

1. High accuracy is essential, since predictors deployed in existing commodity processors achieve accuracy rates of over 97% on integer benchmarks [3]. However even slight improvements on this score are welcome, since the branch mispredict penalty is increasing all the time, with longer pipelines and speculative execution schemes relying on almost-perfect branch prediction.
2. Low latency generally means that the predictor must be able to supply a result in less than than a single processor cycle [4]. With the seemingly relentless increase of clock speeds, this latency requirement becomes ever more demanding.
3. Low complexity is necessary to ensure that the predictor designs can be fabricated in realistic transistor budgets, using low storage overheads and simple inputs that are trivially available in existing processor layouts.

1.1 Motivation

This paper relates a popular ML technique, *Bayesian Networks* (BNs), to existing branch prediction technology, *two-level adaptive branch predictors* [5], particularly the *GAg* and *gshare* schemes. We investigate how to adapt BNs to operate within realistic hardware constraints, by using the same primitive components that are present in existing branch predictors. We systematically study how performance is affected by these simplifications. We aim to use these ideas to reduce the storage overhead of two-level predictors without losing significant prediction accuracy. The key motivating factor is that storage required in two-level predictors grows exponentially with history length, whereas BNs provide a framework to reduce this overhead. The general community consensus is that longer history allows more accuracy, only the exponential growth rate of two-level adaptive predictors prevents longer histories from being used.

1.2 Contributions

This paper has four main contributions.

1. It simplifies the design of BNs so that they are suitable for hardware implementation.
2. It studies how the parameters of the BN model affect its prediction accuracy and storage capacity, the age-old tradeoff in a new context.
3. It discusses the theoretical relation between the GAg predictor and the *fully connected Bayesian Network*.
4. It explores whether BNs are a suitable alternative to GAg or gshare for branch prediction tasks.

2 Background

Any project that combines research from two distinct areas has the task of communicating with two distinct audiences, often with very different research objectives. In this work we attempt to use consistent terminology throughout, so after clarification, some ML concepts may be given architectural labels and vice versa. Section 2.1 describes the practicalities of two-level prediction from an architectural perspective. Section 2.2 outlines the theory of BNs from a ML perspective. Section 2.3 relates these two approaches to prediction, highlighting potential research issues.

2.1 Two-Level Adaptive Branch Prediction

Conventional branch predictors in the systems architecture community are generally implemented as lookup tables. This paper focuses on predictors composed of a global table indexed by a global history register. Yeh and Patt describe these as *GAg* schemes [5].

Each table entry is a 2-bit bimodal counter [6]. When the branch is taken, the appropriate counter is incremented until it saturates at 11_2. When the branch is not taken, the appropriate counter is decremented until it saturates at 00_2. The more significant bit is used to determine whether the predicted outcome is taken or not taken. The bimodal counter provides hysteresis, or the ability to remember general previous behaviour despite transient variations, for instance after the last iteration of a loop.

The index into this table of counters is derived from the outcomes of the most recently executed branches. Hence such schemes are known as *finite context method* predictors since they use a finite context of recent global branch history outcomes to construct the table index. This finite context is recorded in the *global history register* which operates like a shift register, shifting in the most recent branch outcome as the least significant bit after each branch instruction execution. In the simplest case (GAg), this global history is used as the table index directly, which means that the table must have 2^n entries where n is the length of the global history register. In a more complicated case, the global history may be XOR'd with low-order bits from the branch instruction address. This scheme is known as *gshare* [7]. Its XOR-based hashing function spreads the branch predictions more evenly throughout the table. This spreading alleviates the aliasing problem, which can occur when different branch instructions with different history patterns (although with a common suffix) map onto the same table entry.

2.2 Bayesian Networks

Bayesian Networks are a type of *probabilistic model*—a mathematical formalism within the field of Machine Learning, capable of representing and manipulating arbitrary probability distributions over arbitrary random variables. These are now commonly accepted as a state-of-the-art learning technique, finding applications in numerous domains from medical informatics to traffic control.

Bayesian Networks (BNs) are represented as directed acyclic graphs, where each node represents a different random variable. A directed edge from node X to node Y indicates that X has a direct influence on Y. This influence is quantified by the conditional probability $P(Y|X)$, stored at node Y. Nodes in a network can be of two types: *evidence (or attribute)* nodes, and *query (or class)* nodes.

For our purposes, BNs have multiple evidence variables X_1, X_2, ..., X_n and a single class variable C. In terms of branch prediction, the evidence is the value of previous branch outcomes. X_n is the outcome of the most recently committed branch and X_1 is the outcome of the oldest branch on record. The class node C represents the predicted next branch outcome. We adopt the standard encoding for branch outcomes, that 0 denotes 'not taken' and 1 denotes 'taken'. Thus all variables in the network are *boolean*.

The task of any branch predictor is to predict the next outcome, which can be rephrased as predicting the chance of each possible outcome (taken/not taken), given the evidence of previous outcomes. In the language of probability theory,

this is the *posterior probability*, $P(C|X_1, \ldots, X_n)$. One approach to this is to attempt to devise a function that will directly estimate this value. This is the approach taken by the majority of predictors to date.

Alternatively, Bayes' theorem can be used to rearrange the problem,

$$P(C|X_1, X_2, \ldots, X_n) \;=\; \frac{P(C)\; P(X_1, X_2, \ldots, X_n|C)}{P(X_1, X_2, \ldots, X_n)} \tag{1}$$

In practice, for a fixed branch history vector $(X_1, X_2, \ldots, X_N)$ the right-hand denominator is fixed. So we disregard this constant scaling factor. Now the numerator can be rearranged according to the rules of conditional probabilities, giving us a decision rule

```
if P(C = 1)P(X_1, ..., X_n|C = 1) > P(C = 0)P(X_1, ..., X_n|C = 0)
then predict taken
else predict not taken
```

The power of this approach is that we can make assumptions on the dependencies between the branches contained in the branch history buffer.

The simplest type of BN is called *Naive Bayes* (NB). This assumes that all the attributes are independent of each other. Figure 1 shows a NB network.

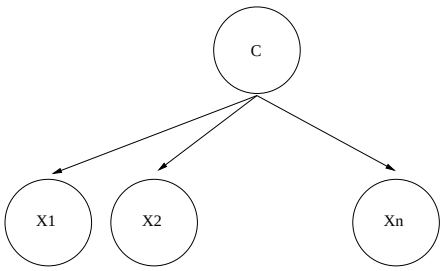

Fig. 1. Naive Bayes classifier

The independence property removes the dependence between each X_i and X_j, simplifying to:

$$P(C|X_1, X_2, \ldots, X_N)\; \alpha\; P(C) \prod_i P(X_i|C) \tag{2}$$

Note that we employ upper case letters (X) to denote random variables, and lower case to denote specific values (x).

For a given history vector $x_1, x_2, \ldots, x_n$, we calculate $P(C = c|X_1 = x_1, X_2 = x_2, \ldots, X_N = x_n)$ using Equation 2 above, for each possible value of c (in our case either 0 or 1). We select as our prediction the more likely value, i.e. the one with the higher conditional probability. (Note that the scaling constant of proportionality is the same for all values of C.)

Although the NB classifier makes simplifying assumptions, it performs robustly for many prediction tasks.

Now we consider the space requirements of the NB classifier in terms of *number of probability values* that must be stored. Note that probabilities can be represented in different ways in hardware, as Section 3 explores. So, since there are two possible values for C, we require one probability value $P(C = 1)$. We can calculate $P(C = 0)$ as $1 - P(C = 1)$. Then, for each attribute X_i, each corresponding to a bit of global history, we require two conditional probabilities. $P(X = 1|C = 0)$ and $P(X = 1|C = 1)$. Again, we can derive $P(X = 0|C = c)$ as $1-P(X = 1|C = c)$. This means, in total, we must remember $2n+1$ probabilities, where n is the length of the global history register.

Friedman et al [8] show that the performance of a BN improves when *augmenting edges* are added between attributes. Recall that the NB classifier assumes all attributes are independent of one another. An *augmented naive Bayes classifier* relaxes this assumption by allowing edges between attributes.

A *tree-augmented naive Bayes classifier* (TAN) is a BN in which the class variable C has no parents and each attribute X_i has as parents the class variable and at most one other variable. Figure 2 shows a TAN network, in which each attribute X_i depends on the preceding attribute X_{i-1}. In terms of branch prediction, this means that a historical branch outcome depends on the prior branch outcome.

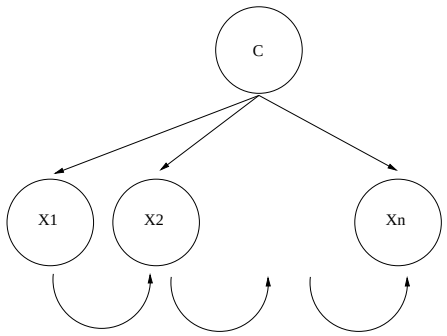

Fig. 2. Tree-augmented Naive Bayes classifier

The probability equation now looks like:

$$P(C|X_1, X_2, \ldots, X_n) \; \alpha \; P(C) \; P(X_1|C) \; P(X_2|X_1, C) \; \ldots \; P(X_n|X_{n-1}, C) \quad (3)$$

Whereas NB stores $2n + 1$ probability values for a history length n, TAN requires $4n - 1$ probability values, due to the additional dependences in the conditional probabilities. However the asymptotic space complexity is still $O(n)$. It is possible to add further augmenting edges to the BN, until eventually it becomes fully connected. In a *fully connected Bayesian network*, if there are n

attribute nodes and one classifier node, then each node is a member of n edge relations, either incoming or outgoing.

2.3 Relating Bayesian Networks and GAg

The GAg scheme effectively stores $P(C|X_1, X_2, \ldots, X_n)$ directly, using 2-bit bimodal counters as discretized estimates of conditional probability. This paper shows how we can use Bayes' rule to approximate this conditional probability, storing fewer probability values along the way. However, poor accuracy of predictors leads us to conclude that the NB network and its improvements do not capture sufficient dependence information for good prediction accuracy. This could be due to compound errors from approximations to probability multiplication.

3 Modifying Bayesian Networks for Hardware Implementation

The default NB model requires some adaptation to make it suitable for deployment in hardware. This section shows how a simplification of the NB model can be implemented using standard hardware components from existing branch prediction schemes, with a small storage overhead. Thus the simplified NB predictor satisfies two requirements from Section 1: *low latency* and *low implementation complexity*.

3.1 Representing the Probabilities

Equation 2 shows that a NB classifier requires the following probability values to be stored, or at least estimated: $P(C = 1)$ and each $P(X_i = 1|C = c)$ where i is bounded by the length of the global history register, and $c \in \{0, 1\}$. Recall that X_i is the ith bit of the global history register, storing the ith most recent branch outcome.

The accurate calculation of probability values requires frequency counts of events. For instance, to calculate $P(X_i = 1|C = 1)$ requires two event counters:

1. number of times that $X_i = 1$ and $C = 1$
2. number of times that $C = 1$

The second event counter will be reused for many probability calculations. However asymptotically the number of event counts scales as $O(n)$ with the history length n, in the same way as the number of probabilities.

Unfortunately, such event counts require an unbounded amount of storage. It is not possible to record probabilities as floating-point numbers since then it is not possible to know the relative significance of each new event, so the probabilities cannot be updated properly.

One workaround solution is to have a fixed window of recent events (like the global history register). It would be possible to store bounded counts in

relation to this window size. This gives us a bound on storage capacity but there is another difficulty. The event counts have to be converted into probabilities using floating point arithmetic, which is almost certainly too complicated to include in the prediction unit. It may be possible to work in terms of logarithms, then the calculations become integer arithmetic but this is still too complex.

Discrete Approximations to the Probabilities A simpler scheme uses *2-bit bimodal counters*, inspired by the conventional table-based prediction schemes such as GAg and gshare. This effectively discretizes probability estimates. So $P(X_i = x|C = c)$ can be a value from the set $\{00_2, 01_2, 10_2, 11_2\}$. We use Smith's standard update scheme with increment/decrement and counter saturation at 00_2 and 11_2 [6]. For each bit of history, there are two counters $p_{i,0}$ and $p_{i,1}$ one for the case when c=0, the other for c=1. Counter $p_{i,0}$ will be updated when the branch outcome is 0. It will be incremented if $x_i = 1$, and decremented if $x_i = 0$. On the other hand, counter $p_{i,1}$ will be updated when the branch outcome is 1. Again, it will be incremented if $x_i = 1$, and decremented if $x_i = 0$. There is also a counter p_c, which is incremented every time the branch outcome is taken and decremented every time it is not taken.

So counter $p_{i,c}$ corresponds to the original probability value $P(X_i = 1|C = c)$. Just as we can compute $P(X_i = 0|C = c)$ using $1 - P(X_i = 1|C = c)$, we can compute the corresponding 2-bit bimodal counter as $11_2 - p_{i,c}$. For the rest of this section, we define function q as follows:

$$q_{i,c} = \begin{cases} p_{i,c} & \text{when } x_i = 1 \\ 11_2 - p_{i,c} & \text{when } x_i = 0 \end{cases}$$

This scheme provides a discretized estimate of posterior probabilities.

$q_{i,c}$	$P(X_i = x_i\|C = c)$
00_2	0
01_2	1/3
10_2	2/3
11_2	1

It might be possible to use these discretized probabilities to perform the actual probability calculation from Equation 2, using either lookup tables or simple binary algebra. However, the high frequency of 00_2 values ensures that most calculations generally result in 00_2 answers. The more satisfactory alternative to predict the outcome is to determine the more likely value (0 or 1) in the most significant bit of each bimodal counter indicated by Equation 2. So, for a history vector $x_1, x_2, \ldots, x_n$, we determine which of the following sets of counters has more top bits set: either (a) $\{p_c\} \cup \bigcup_i q_{i,1}$, or (b) $\{(11_2 - p_c)\} \cup \bigcup_i q_{i,0}$. If (a) has more top bits set than (b) then the predicted outcome is 1, otherwise the predicted outcome is 0.

The storage overhead of this 2-bit bimodal NB predictor is: two 2-bit counters for each bit in the global history register, plus one counter for C. This is a small

storage overhead indeed. Moreover it scales linearly with history length, whereas the gshare storage scales exponentially.

Figure 3 shows how the simplified NB predictor is conceptually implemented in hardware. This schematic diagram shows that the set of 2-bit counters is arranged as a 3-d array, based on (c, i, x_i) where c is branch outcome, i is history bit index and x_i is the actual value of the ith history bit. In fact, since the limits of each dimensional index are fixed in advance, the 3-d array can be flattened to a linear vector. Also note that in the x_i dimension, we only need to store the 2-bit value v for when $x_i = 1$, since we can calculate the value for when $x_i = 0$ as $11_2 - v$. The same applies for the unconditional probabilities $P(C = 1)$ and $P(C = 0)$. Thus it is clear to see that for n bits of global history, the simplified NB predictor stores $2n + 1$ counter values.

The figure shows how to calculate the likelihood that the next branch outcome will be 0. We use counters from the $c = 0$ row and select between the $x_i = 0$ and $x_i = 1$ counters for each i based on the corresponding values in the global history register. We check the top bits for each selected counter and sum to see how many of these top bits are set. Then we do the same for the $c = 1$ row. We use a comparator (not shown in figure) to determine which outcome is more likely, and use this outcome as our prediction.

Predictor state update works in a similar way. We use the actual branch outcome to determine whether to update the $c = 0$ or $c = 1$ row. We update the $x_i = 1$ counters based on the values in the global history register—increment the counter if $x_i = 1$ or decrement if $x_i = 0$.

4 Evaluation Framework

We use the recently released second championship branch prediction framework (CBP2) [9] to evaluate our branch prediction implementations. Each implementation is coded in high-level C++, although in such a way that could easily be encoded in hardware. Thus arrays will map into indexed table lookups, integers will become bit strings, shifts and masks are used for bit selection and concatenation, and so on.

The default predictor is a simple gshare implementation. We compare all our predictors in this paper with this default gshare predictor. The framework includes a selection of real-world execution traces containing branch information. For each branch, the predictor is supplied with the branch instruction address and the branch target address. From these inputs and its internal state, the predictor must predict the branch outcome. Some short time later, the result of this branch is fed back to the predictor to enable it to update its state. The framework keeps track of the prediction accuracy and reports this at the end of the trace. Traces cover programs from benchmark suites including SPEC INT 2000 and SPEC JVM 98. The final result for a trace is reported in units of *MPKI*, which is mispredicts per 1000 instructions. The final result for all the programs is taken as the arithmetic mean of all individual traces. We modify the framework to report results in terms of percentage of mispredicted branches,

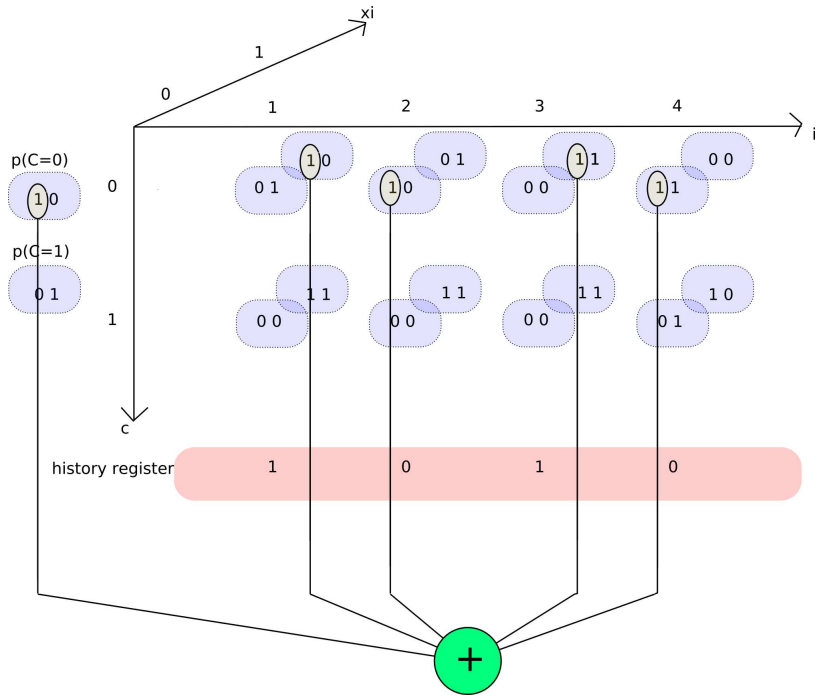

Fig. 3. Schematic diagram of simplified Naive Bayes predictor, calculating the likelihood that the next branch outcome will be 0 given the history 1010. The complete calculation requires checking the likelihood that the next branch will be 1, and then selecting the larger probability as the predicted outcome.

which is the conventional metric for branch prediction accuracy. CBP2 is an industrial quality simulation framework for branch prediction. Its predecessor, CBP1, has been comprehensively analysed by Loh [10].

The CBP2 default gshare predictor achieves a score of 6.3 MPKI (just under 5% misprediction rate). In CBP1, the winning predictor achieved a score of 2.5 MPKI. Any new predictor will need to have comparable performance and hardware complexity if it is to be accepted as a realistic alternative to current implementations.

5 Predictor Comparison: unbounded versus bounded NB

This section investigates how the NB simplification affects prediction accuracy. We implement two predictors, u-NB and n-NB. The u-NB model is an *unbounded* NB predictor, as described in Section 3.1. It keeps unbounded integer counts of all necessary events. It uses double-precision floating-point arithmetic to calculate all probabilities. It computes Equation 2 exactly to determine the most likely outcome. The n-NB model is the bimodal simplification as outlined in Section 3.1. It maintains probabilities as n-bit bimodal counters. These are updated with saturating increments or decrements. The prediction is made by examining top bits and choosing the outcome that has more top bits set.

Figure 4 shows the accuracy scores for u-NB, 2-NB and 4-NB on the CBP2 dataset, with different lengths of global history register. It is clear to see that accuracy improves as history length increases. This trend is apparent for both all three predictors. Although 2-NB tracks the performance of u-NB for short history lengths, the divergence increases with history length. The 4-NB predictor is more accurate than u-NB for all history lengths, and the performance improvement is sustained over longer history lengths than 2-NB. This is a most satisfactory result—our discrete approximation to the NB algorithm performs better than the original algorithm, for this dataset. This is because 4-bit counters provide enough *hysteresis* to avoid being upset by transient behaviour, but they are sensitive enough to forget the distant past history, in a way that u-NB cannot. Section 7.2 discusses this tradeoff further.

6 Predictor Comparison: NB versus gshare

This section compares the performance of our n-NB predictor with the standard gshare predictor. A fair comparison must ensure that predictors use equal storage capacities. We assume that the rest of the prediction logic to be roughly equivalent, so when we set the storage capacities to be equal then the predictors have equivalent implementation complexity.

For the 4-NB predictor from the previous section, with a history register length of h_b, there will be $2h_b + 1$ 4-bit bimodal counters, making a total of $8h_b + 4$ bits. In contrast, a gshare predictor for history length h_g will have a table of 2^{h_g} entries, each of which is a 2-bit bimodal counter, making a total of

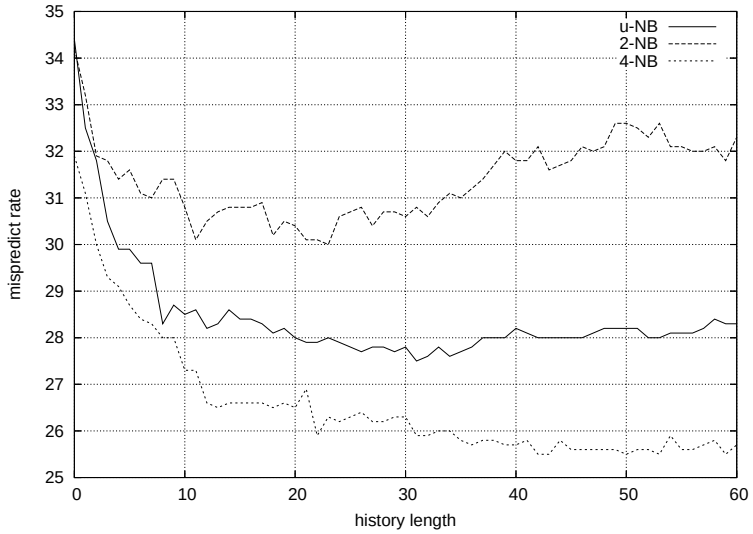

Fig. 4. Comparison of Naive Bayesian Predictors

2^{h_g+1} bits. A rearrangement of these equations shows that to have equal storage, the gshare predictor must have history length $h_g = \lceil 1 + log_2(2h_b + 1) \rceil$.

We investigate how these equations effect the predictors by comparing 4-NB predictors of varying history lengths with equivalently sized gshare predictors (with correspondingly shorter history lengths according to the above equation). Figure 5 shows the difference in performance between equally sized 4-NB and gshare predictors. Note that gshare is the clear winner and the performance difference grows with history length.

Thus it is clear that the simple adaptation of NB is inferior to gshare, not a suitable candidate for deployment in real processors. The next section investigates how to close this gap between n-NB and gshare, by considering different improvements to the n-NB prediction scheme. We hope to retain the branch predictor characteristics (from Section 1) of low implementation complexity and low latency, while achieving *high accuracy*.

7 Exploring the Bayesian Predictor Family

There are various parameters in the n-NB predictor model that may be tuned in order to improve the accuracy of predictions. This section considers tuning the global history register length (Section 7.1), the bimodal counter length (Section 7.2), the set associativity (Section 7.3) and the connectedness of the BN (Section 7.4).

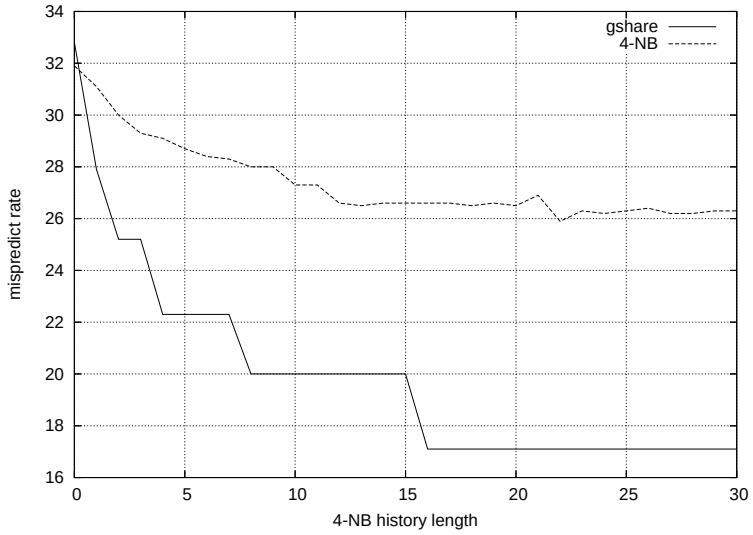

Fig. 5. Comparing performance of equally sized NB and gshare predictors

7.1 History Length

The standard method to increase prediction accuracy is to make the global history register longer. This allows for more context in a prediction, reducing the effects of the aliasing problem. Figure 6 shows how NB predictors, with different bimodal counter lengths perform as the global history register length increases. It is clear to see that the misprediction rate decreases as the history register lengthens, although the rate of decrease reduces at higher history lengths.

7.2 Reactivity

The problem with the u-NB predictor from Section 3.1 is that it is insensitive to sudden probability distribution changes or program phase shifts. In contrast, saturating bimodal counters are able to react to such changes. The length of the bimodal counter determines its hysteresis, or how long it takes to react to changes.

We varied the length of the bimodal counters to find the optimum length. For the benchmarks given, the optimum performance was 4-NB, as shown in Figure 6. Longer counters cause degraded predictions since since they take too long to react to changes. Shorter counters cause degraded predictions since they forget reliable old history in favour of extremely transient behaviour. Note that some n-NB implementations outperform u-NB since u-NB is never able to 'forget' old history, so it becomes progressively more difficult to react to phase changes.

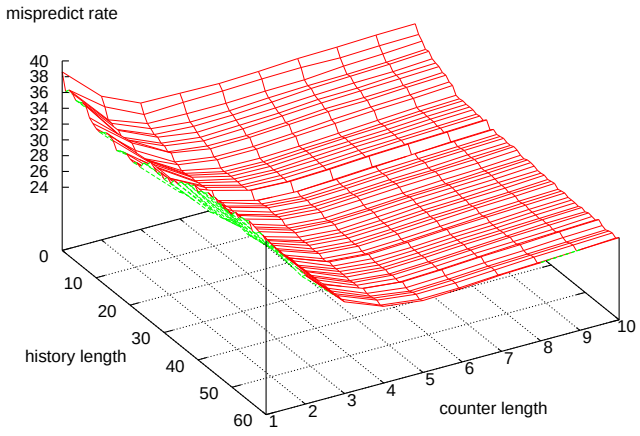

Fig. 6. Naive Bayes predictor accuracy varies with history length and counter length

7.3 Set Associativity

A common reason for low prediction accuracy is *aliasing*. The gshare predictor reduces aliasing by incorporating some low order bits from the branch instruction address into table lookup index. Another approach to reduce aliasing is *set associativity*. In this case, there are several NB predictors operating in parallel (NB/sa). The appropriate NB predictor to use for branch instruction b is based on the low-order bits from the branch instruction address. This approach is commonly used in processor caches. We investigate how it works for NB predictors. The main drawback is the growth in storage requirements for the predictor. The storage space grows exponentially with the number of instruction address bits used.

Figure 7 shows how set associativity affects the performance of the 2-NB and 4-NB predictors, each with a global history register length of 20. The x axis shows the number of bits of branch instruction address used, so the number of NB predictors will be 2^x. The graph shows how NB/sa prediction accuracy increases with set associativity. This is due to the decreasing amount of aliasing. However once the set-associativity exceeds a certain amount then the accuracy begins to degrade, presumably because there are so many parallel n-NB predictors, each for so few branch instructions that there is little or no global correlation between branch instructions.

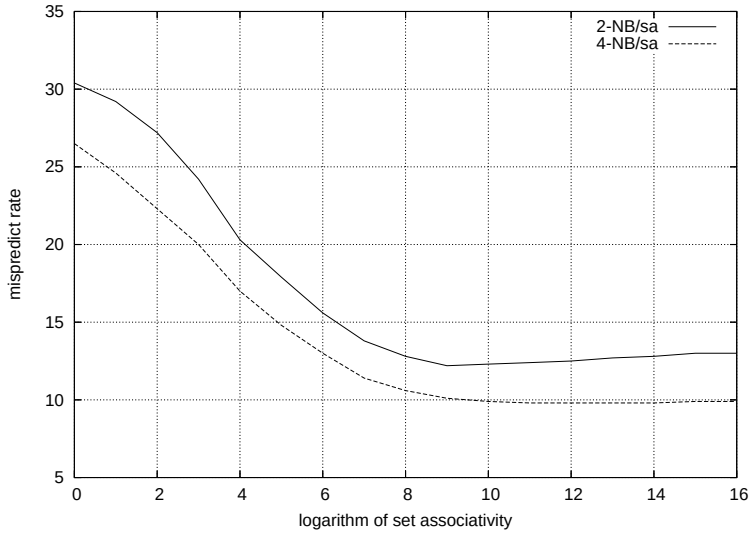

Fig. 7. Predictor performance improves with set-associativity

7.4 Network Connectivity

Section 2.2 described how the performance of the NB predictor can be improved by adding augmenting dependence edges between the attribute nodes, resulting in an augmented Naive Bayes predictor (aNB). This section investigates how these extra edges affect prediction accuracy. The conditional probability terms simply gain extra dependent variables. So for NB each term has the form $P(X_i|C)$ which translates into two counters, for when $(X_i = 1, C = 0)$ and when $(X_i = 1, C = 1)$. For TAN, each term has the form $P(X_i|X_{i-1}, C)$ which translates into four counters, for the four possible combinations of values for (X_{i-1}, C). In the general case, if each term has m dependent variables, then it requires 2^m counters.

The update scheme only examines $n+1$ counters each time, one for P(C) and one for each of the n history bits. It uses the values of the dependent variables to determine which counter to select for each term. Similarly, a likelihood check for a particular outcome only examines the top bits of $n + 1$ counters.

Figure 8 shows how increased network connectivity affects the performance of the 2-aNB and 4-aNB predictors, each with a global history register length of 20. The x axis represents the *maximum* number of augmenting edges (AEs) per node. (Edges always point forwards, so X_1 can only depend on C, X_2 on C and X_1 whereas X_n can depend on C and all of $X_1, X_2, \ldots, X_{n-1}$.)

When the number of AEs is 0, the predictor is the special case of NB. When the number of AEs is 1, the predictor is the special case of TAN.

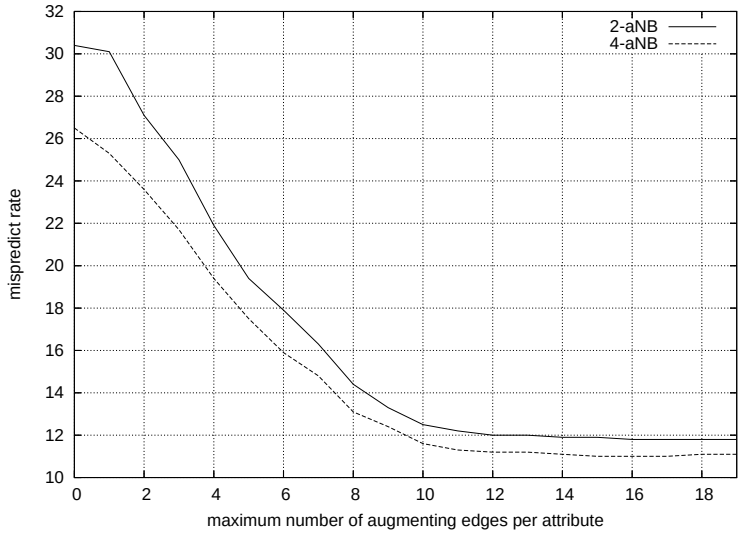

Fig. 8. Performance improves with increasing numbers of augmenting edges

8 Predictor Comparison: Improved NB versus gshare

Whereas Section 6 compared the performance of the n-NB predictor with equivalently sized gshare predictor, this section compares the improved version of the NB predictor with gshare. We combine all the NB performance enhancements from Section 7. Note that some enhancements seem to provide greater improvements than others. A more detailed empirical investigation would measure the ratio of accuracy improvement to storage overhead for the different enhancement schemes. Again, it is not clear how the various enhancements interact with one another. For simplicity, here we assume independence. A more detailed study would investigate this empirically!

We conducted numerous experiments, although we did not exhaustively explore the parameter space. Disappointingly none of our parameter settings enabled the bayesian predictor to achieve better accuracy than gshare, although many settings were close. For instance, given a aNB/sa predictor that uses 4-bit counters, 31 bit history, maximum of 15 augmenting edges and 256-way set associativity, the average mispredict rate is 4.7%. In contrast, an equivalently sized gshare predictor would have a history length of 19 bits, achieving accuracy score 4.0%.

9 Related Work

Several papers characterize branch prediction as a ML problem. For instance, Calder et al [11] use *decision trees* to predict branch outcomes at compile time,

based on static program features. Fern and Given [12] formulate dynamic branch prediction as an online learning problem. They use *ensemble learning* techniques that are suitable for 'resource-limited environments.' However they do not provide hardware implementation details and they focus on a small set of branches that are difficult to predict.

Vintan [13] pioneers the idea of using *perceptrons* for branch prediction. Jimenez and Lin [14, 15] go on to show how perceptrons can be implemented in realistic hardware, and achieve better results than existing non-ML predictors. However they require complex techniques to disguise the latency of their perceptron predictor, involving cascaded predictors and pipelining.

Yeh and Patt [5] study a parameterized family of two-level adaptive predictors. They fully explore this small parameter space, which has 9 members. Their principled approach is a good guide for our work. Emer and Gloy [16] devise a language to describe conventional branch predictor models, and then use *genetic programming* to evolve new predictor models. However they admit that the auto-generated predictors are 'logically complex and probably not directly implementable.'

Online feature selection [17] is another interesting ML problem. In our case, we could chose which augmenting edges to insert dynamically, and perhaps adapt for programs with different branching characteristics.

10 Concluding Remarks

This paper has shown that Bayesian Networks provide a useful formalism for describing a family of branch predictors. The common GAg and gshare schemes can be explained in relation to BNs in terms of conditional probabilities. We have sketched a potential hardware implementation of this ML technique, and performed some initial evaluation.

One interesting observation is that a full floating-point implementation of NB is outperformed by our discrete approximation, using 4-bit bimodal counters.

Although we have not yet fully explored the space, we have found BN predictors that approach the gshare accuracy (to within 1%). We believe that this framework provides promise for future branch prediction technology, particularly in terms of storage overhead reduction. Our current research is focusing on further exploration of the space, and application of ML methods to automatically *learn* the best network connectivity, while the predictor is in use.

It should be noted that in adopting the BN formalism, we are addressing a subtle aspect of branch predictors that has not been previously considered. The Machine Learning literature can be broadly divided into two camps—*discriminative* and *generative* learning. The form of learning we have used is generative, since we model the joint distribution $P(X, C)$. Neural branch predictors [14] are discriminative, since they model the posterior distribution $P(C|X)$. The exact advantages of each in any given situation are an area of active research and therefore constitute a novel and promising technology for the architectural community to consider.

References

1. Hamerly, G., Perelman, E., Lau, J., Calder, B., Sherwood, T.: Using machine learning to guide architecture simulation. Journal of Machine Learning Research **7** (Feb 2006) 343–378
2. Barham, P., Isaacs, R., Mortier, R., Harris, T.: Learning communication patterns in Singularity. In: Proceedings of the First Workshop on Tackling Computer Systems Problems with Machine Learning Techniques. (2006)
3. Hennessy, J.L., Patterson, D.A.: Computer Architecture A Quantitative Approach. 3rd edn. Morgan Kaufmann (2003)
4. Jiménez, D.A., Keckler, S.W., Lin, C.: The impact of delay on the design of branch predictors. In: MICRO 33: Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture. (2000) 67–76
5. Yeh, T.Y., Patt, Y.N.: A comparison of dynamic branch predictors that use two levels of branch history. In: Proceedings of the Annual Symposium on Computer Architecture. (1993) 257–266
6. Smith, J.E.: A study of branch prediction strategies. In: Proceedings of the Annual Symposium on Computer Architecture. (1981) 135–148
7. McFarling, S.: Combining branch predictors. Technical Report TN-36, Digital Equipment Corporation (Jun 1993)
8. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Machine Learning **29**(2-3) (1997) 131–163
9. Jiménez, D.A.: Second championship branch prediction (2006) `http://camino.rutgers.edu/cbp2/`.
10. Loh, G.H.: Simulation differences between academia and industry: A branch prediction case study. In: International Symposium on Performance Analysis of Software and Systems. (2005) 21–31
11. Calder, B., Grunwald, D., Jones, M., Lindsay, D., Martin, J., Mozer, M., Zorn, B.: Evidence-based static branch prediction using machine learning. ACM Transactions on Programming Languages and Systems **19**(1) (1997) 188–222
12. Fern, A., Givan, R.: Online ensemble learning: An empirical study. Machine Learning **53**(1-2) (2003) 71–109
13. Vintan, L., Iridon, M.: Towards a high performance neural branch predictor. In: International Joint Conference on Neural Networks. Volume 2. (1999) 868–873
14. Jiménez, D.A., Lin, C.: Neural methods for dynamic branch prediction. ACM Transactions on Computer Systems **20**(4) (2002) 369–397
15. Jiménez, D.A.: Improved latency and accuracy for neural branch prediction. ACM Transactions on Computer Systems **23**(2) (2005) 197–218
16. Emer, J., Gloy, N.: A language for describing predictors and its application to automatic synthesis. In: Proceedings of the 24th annual international symposium on Computer architecture. (1997) 304–314
17. Fern, A., Givan, R., Falsafi, B., Vijaykumar, T.: Dynamic feature selection for hardware prediction. Journal of Systems Architecture **52**(4) (2006) 213–224