Branch Prediction Based on Universal Data Compression Algorithms

Eitan Federovsky, Meir Feder, and Shlomo Weiss
Electrical Engineering-Systems, Tel Aviv University
Tel Aviv 69978, ISRAEL

Abstract

Data compression and prediction are closely related. Thus prediction methods based on data compression algorithms have been suggested for the branch prediction problem. In this work we consider two universal compression algorithms: prediction by partial matching (PPM), and a recently developed method, context tree weighting (CTW). We describe the prediction algorithms induced by these methods. We also suggest adaptive algorithms - variations of the basic methods that attempt to fit limited memory constraints and to match the non-stationary nature of the branch sequence. Furthermore, we show how to incorporate address information and to combine other relevant data. Finally, we present simulation results for selected programs from the SPECint95, SYSmark/32, SYSmark/NT, and transactional processing benchmarks. Our results are most promising in programs with difficult to predict branch behavior.

1. Introduction

A substantial body of information theory research [6, 15, 22, and others] deals with the problem of universal coding, that is, compressing a sequence of symbols produced by an unknown source. A universal compression method builds a model of the source specifying the probability at which various symbols occur, so that frequently occurring symbols may be assigned short codes. This probabilistic model may be used to predict the next outcome in the sequence. Compression and prediction are closely related: a sequence that is compressed well is easy to predict, and on the other hand, a random sequence is both incompressible and unpredictable. In [7] it was shown that prediction and compression performance of binary sequences can be related

$$h^{-1}(H) \le M \le H/2, \qquad (1)$$

where $h(x) = -x\log x - (1-x)\log(1-x)$ is the binary entropy function, M is the misprediction ratio, and H is the entropy or compressibility. A more general relation between prediction and compression for general sequences is given in [8]. Hence if the compressibility of a sequence of branch outcomes is H, we can theoretically achieve misprediction of less than $0.5H$.

In this work we use the close relationship between compression and prediction methods, as indicated by (1). Our primary objective is to apply recent information theory results and methods of universal coding and universal prediction to branch prediction. We explore the application and performance of two algorithms, prediction by partial matching, and context-tree weighting, a recently introduced [18] universal data compression method.

1. 1 Related Work

Branch prediction has been a very active area of computer architecture research in recent years [10,14,20,21]. The more general problem, that of predicting the next outcome in an arbitrary sequence of observations, has been a topic of interest in information theory research for more than four decades. It was Shannon himself who saw this connection and used prediction to assess the entropy of the English language[16], and build a "mind-reading" machine that predicts human decisions. Kelly [9] and Cover [4] relate information and data compression to gambling. More recently, utilizing the theory of universal coding for individual sequences of Ziv and Lempel [22], Feder, Merhav and Gutman [7] presented a framework for universal prediction that defines the connection and allow the utilization of universal coding for prediction.

In computer architecture, data compression methods have been recently applied to prefetching [5,17] and to branch prediction [2,13]. In [2,13], Mudge, Chen, and Coffey introduced the concept of using data compression methods for branch prediction. Our work continues this approach. We investigate and apply prediction methods based on the following two algorithms:

1. Willems, Shtarkov, and Tjalkens introduced a new compression algorithm, context tree weighting (CTW), in [18]. We apply the context-tree weighting algorithm to branch prediction. We further develop the CTW algorithm and propose some variations of it that make it *adaptive.*
2. Partial prediction matching (PPM) is a well known data compression method [1,3,12] (in the rest of this paper we refer to this method as *static* PPM). There are several possible variations and extensions to the PPM, see Williams [19]. Some can be described as *adaptive* PPM. We look at both variations of PPM, static and adaptive. The latter, adaptive PPM, employs a context tree that grows dynamically, and is different than the static PPM algorithm used for branch prediction in [2].

1. 2 Paper Overview

We begin with a description of tree sources (section 2) and context trees (section 3). Sections 2 and 3 follow mostly the notation of [18], and provide the background for the branch prediction algorithms. In section 4 we describe the use of context trees to model a sequence of branches with the CTW and PPM algorithms, discuss adding address information to the context tree, and introduce the adaptive CTW and PPM algorithms. Section 5 contains performance results, and finally section 6 is a summary of the results and directions for future research.

2. Tree Sources

We look at a program in execution as a source that generates a binary sequence in which 1's correspond to taken branches and 0's correspond to not-taken branches. The simplest (and unrealistic) model is a memoryless source that generates a 1 with probability *p*. A *tree source* describes a source with multiple parameters p_i that depend on the previous sequence of bits.

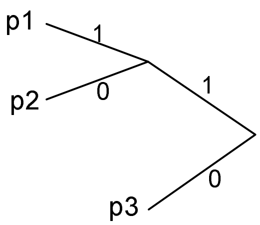

Figure 1 - Tree source

Assume that the outcome of previous branches is a binary sequence $x_1^T = x_1 x_2 \ldots x_t$. Beginning with the rightmost bit x_t, we trace our way from the root through the tree until we reach a leaf node, where we find the parameter for generating the next bit in the sequence. Hence the probability that the next bit will be 1 in the sequence produced by the tree source in Figure 1 is

$$P(X_{t+1} = 1 | x_1^T = 11) = p_1$$

$$P(X_{t+1} = 1 | x_1^T = 01) = p_2$$

$$P(X_{t+1} = 1 | x_1^T = 0) = p_3.$$

A tree source is specified by the *model*, which is the structure of the tree, and the *parameters* (probabilities p_1, p_2, p_3 in Figure 1) defined at the leaf nodes.

3. Context Trees

Often we have to deal with unknown sources, that is, both the model and the parameters of the source are not known. Lacking more precise information, we use a context tree for the source model. A *context tree* [18] is a binary tree of depth *d*. Each path from root to a node in the tree represents a binary sequence of length ≤ *d*. This sequence is the context *s* of the node. The context of the root is the null sequence λ. In Figure 2 below we draw the tree horizontally with the root at the right so that the context of a node in the tree can be visually associated with the corresponding binary sequence.

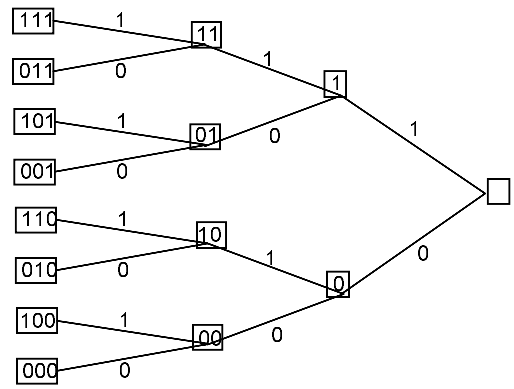

Figure 2 - Context tree of depth d = 3. The context of each node is shown in a rectangle.

The subsequence of x_1^T associated with a context *s* is the sequence of bits in x_1^T that occurred after the context *s*.

Example 1

Figure 3 shows a binary sequence in which the bits are numbered 1 to 9. We also show several bits that occurred in the "past" in order to provide the context for the first bits in the sequence. Context 010, for example, occurs twice in the sequence and is followed by bits 1 and 6. Hence the subsequence associated with context 010 is 11 (bits 1, 6), shown circled in the figure.

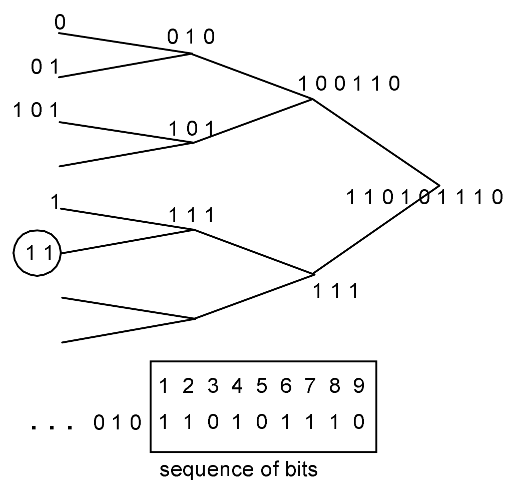

Figure 3 - The bits at each node are the subsequence associated with the context of that node.

In a tree of depth *d,* the *current context path* is the path that corresponds to the *d* rightmost bits of the sequence. In the example in Figure 3, the current context path is 110.

Having defined a model (a context tree) for the source, we also need a way to specify its parameters. Disregarding context trees for the moment, assume the sequence x_1^T consists of a zeros and b ones. Then the conditional probability, defined by the Krichevsky and Trofimov [11] distribution, that the next outcome in the sequence will be a one is:

$$P_e(X_{t+1} = 1 | x_1^T) = \frac{b + 1/2}{a + b + 1}. \tag{2}$$

The probability of the entire sequence x_1^{T+1} is the product of the conditional probabilities of the bits in the sequence:

$$P_e(x_1^{T+1}) = P_e(X_{t+1} = 1 | x_1^T) \cdot P_e(X_t = 1 | x_1^{T-1}) \cdot \ldots \\ P_e(X_1 = 1 | x_1^0), \tag{3}$$

which, assuming the sequence x_1^T has a zeros and b ones, can be rewritten as

$$P_e(x_1^{T+1}) = P_e(a,b) = \\ \frac{1/2 \cdot 3/2 \cdot \ldots \cdot (a + 1/2) \cdot 1/2 \cdot 3/2 \cdot \ldots \cdot (b + 1/2)}{1 \cdot 2 \cdot \ldots \cdot (a + b + 1)}. \tag{4}$$

Returning to the context tree, instead of counting zeros and ones of the entire sequence, we count zeros and ones at each node of the context tree separately.

Example 2

The subsequence associated with context $s = 010$ in Figure 3 is 11, shown circled in the figure. We have the count of zeros $a_s = 0$ and the count of ones $b_s = 2$, and from equation (4), $P_e(a_s, b_s) = 3/8$.

We can choose any complete sub-tree (that is, every node that is not a leaf node has both children) that includes the root of the context tree to allocate a probability to the sequence x_1^T as follows. The leaves s_i of this tree Tr are the relevant contexts, and each x_t is associated with some leaf. The entire probability will be

$$P_{Tr}(x_1^T) = \prod_{\{s_i \text{ leaves in } Tr\}} P_e^{s_i} = \prod_{s_i} P_e(a_{s_i}, b_{s_i})$$

The choice of the tree Tr defines a model with variable length history. The number of parameters in this model is $|s|$, the number of leaves in the tree.

4. Prediction Algorithms Based on Context Trees

4.1 Context Tree Weighting (CTW) Algorithm

Looking again at a binary sequence $x_1^T = x_1 x_2 \ldots x_t$, s_i is the context of the sequence at level $i \geq 1$ the tree, namely the context defined by the subsequence $x_{t-i+1} \ldots x_{t-1} x_t$. We define $\bar{s}_i$ as the context specified by the subsequence $\bar{x}_{t-i+1} \ldots x_{t-1} x_t$, where $\bar{x}_{t-i+1}$ is the bit complement of x_{t-i+1}. At level $i = 0$ (the root node), the context is the null context $s_0 = \lambda$. The *weighted* probability $P_w^{s_i}$ at an internal node node s_i, is defined by the following recursive equation:

$$P_w^{s_i} = \frac{1}{2} P_e^{s_i} + \frac{1}{2} P_w^{s_{i+1}} P_w^{\bar{s}_{i+1}}, \tag{5}$$

where $P_e^{s_i}$ is determined by the count of zeros a_{s_i} and ones b_{s_i} at node s_i using (4). For leaf nodes we simply have

$$P_w^{s_i} = P_e^{s_i}. \tag{6}$$

Example 3

Figure 4 illustrates weighted probabilities of the previous context tree example calculated using equations (5) and (6).

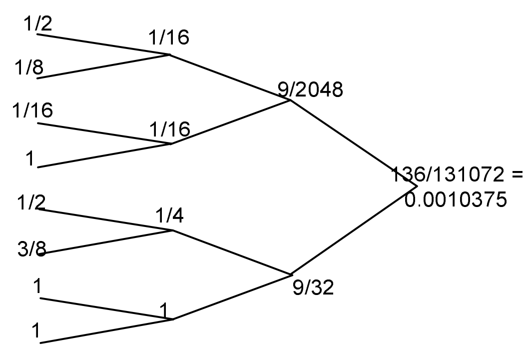

Figure 4 - Weighted probabilities at the nodes of the context tree example from Figure 3.

We have now chosen a model, a context tree of depth d, to represent a source of branch outcomes, and we have a way, equations (5) and (6), to compute the parameters of the model. Using equations (5) and (6), we compute the weighted probability P_w^{λ} at the root node (whose context is the null context λ).

It can be shown that $P_w^{\lambda}(x_1^T)$ is a weighted probability that weights the probabilities $P_{Tr}^{j}(x_1^T)$ induced by all possible complete subtrees. Specifically

$$P_w^{\lambda}(x_1^T) = \sum_j w(Tr^j) \cdot P_{Tr^j}(x_1^T),$$

where $w(Tr^j) = 2^{-(2|s|_j - 1)}$ and $|s|_j$ is the number of leaves in Tr^j.

We use the context tree to predict branches as follows. Let x_1^T be the outcome sequence of previous branches. Assuming the next branch outcome is 1, we append 1 to x_1^T to get the sequence of branches that corresponds to this assumption. We update *b*, the number of 1's, and recompute the weighted probabilities at the nodes of the context tree. Note that only nodes on a single path from root to a leaf node are affected. The weighted probability at the root node, after the update, is $P_w^{'\lambda}$ (P' is P updated). The outcome of the branch that follows the sequence x_1^T is 1 with a probability

$$P(X_{t+1} = 1 | x_1^T) = \frac{P_w^{'\lambda}}{P_w^{\lambda}} \tag{7}$$

We predict that the outcome of the branch will be 1 using the randomized predictor described in [7]. Specifically, if $P_w^{'\lambda} / P_w^{\lambda} > 0.5 + \varepsilon$, where $\varepsilon = 1/\sqrt{T}$, we predict 1, if $P_w^{'\lambda} / P_w^{\lambda} < 0.5 - \varepsilon$ we predict 0, and if $0.5 + \varepsilon \geq P_w^{'\lambda} / P_w^{\lambda} \geq 0.5 - \varepsilon$ we predict 1 with probability $\Phi(P)$ as shown in Figure 5.

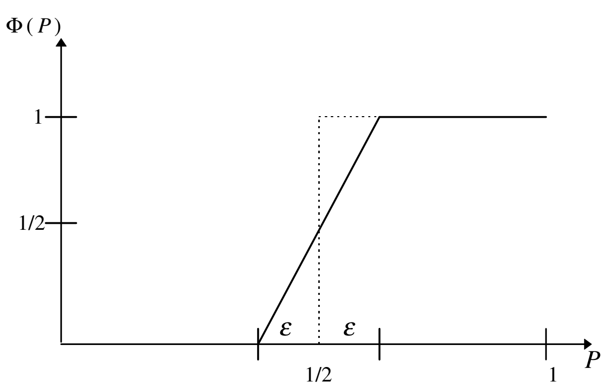

Figure 5 - Randomized predictor

If the branch outcome is indeed 1, we retain the updated context tree and proceed to predict the next branch. Otherwise, we restore the previous values of the context tree, and perform the update that corresponds to appending a 0 to the branch sequence.

Example 4

Appending a 1 to the right of the sequence of bits in Figure 3, and recalculating the weighted probabilities, we get $P_w^{'\lambda} = 0.0008006$. We have $P_w^{\lambda} = 0.0010375$ from Figure 4. Hence $P_w^{'\lambda} / P_w^{\lambda} = 0.77$, and we predict the next branch outcome will be 1.

The following values are maintained at each node *s* of the context tree:

1. A count of the zeros a_s that occurred in context *s*.
2. A count of the ones b_s that occurred in context *s*.
3. The probability estimate $P_e(a_s, b_s)$ given by equation (4).
4. The weighted probability P_w^s given by equations (5) and (6).

The last two values allow incremental computation at affected nodes, rather than recomputing the parameters of the entire tree for each update.

4. 2 Sequential Representation of CTW Based Prediction

Interestingly, the conditional probability used for prediction $P(X_{t+1} = 1 | x_1^T)$ turns out to be a weighted sum of the conditional probabilities of all the nodes on the current context path, as shown below.

Beginning with (5)

$$P_w^{s_i} = \frac{1}{2} P_e^{s_i} + \frac{1}{2} P_w^{s_{i+1}} P_w^{\bar{s}_{i+1}}$$

and dividing by $P_w^{s_i}$ we get

$$1 = \frac{1}{2} \frac{P_e^{s_i}}{P_w^{s_i}} + \frac{1}{2} \frac{P_w^{s_{i+1}} P_w^{\bar{s}_{i+1}}}{P_w^{s_i}}.$$

For internal nodes, define

$$P_n^{s_i} = \frac{1}{2} \frac{P_e^{s_i}}{P_w^{s_i}} = 1 - \frac{1}{2} \frac{P_w^{s_{i+1}} P_w^{\bar{s}_{i+1}}}{P_w^{s_i}}. \tag{8}$$

For leaf nodes, we define $P_n^{s_i} = P_e^{s_i} / P_w^{s_i}$, and from (6) it follows $P_n^{s_i} = 1$.

As before, to get the probability that the next outcome after sequence x_1^T will be 1, we append 1 to x_1^T. The updated weighted probability at node s_i is

$$P_w{}'^{s_i} = \frac{1}{2} P_e{}'^{s_i} + \frac{1}{2} P_w'^{s_{i+1}} P_w^{\bar{s}_{i+1}} .$$

Dividing by $P_w^{s_i}$

$$\frac{P_w{}'^{s_i}}{P_w^{s_i}} = \frac{1}{2} \frac{P_e{}'^{s_i}}{P_w^{s_i}} + \frac{1}{2} \frac{P_w'^{s_{i+1}} P_w^{\bar{s}_{i+1}}}{P_w^{s_i}} . \tag{9}$$

Define

$$P_0^{s_i} = \frac{P_e'^{s_i}}{P_e^{s_i}} . \tag{10}$$

Substituting the above definition and the definition of $P_n^{s_i}$ from (8) into equation (9), we get:

$$\begin{aligned} \frac{P_w{}'^{s_i}}{P_w^{s_i}} &= P_n^{s_i} \cdot P_0^{s_i} + \frac{1}{2} \frac{P_w'^{s_{i+1}} P_w^{\bar{s}_{i+1}}}{P_w^{s_i}} \\ &= P_n^{s_i} \cdot P_0^{s_i} + \frac{1}{2} \frac{P_w^{s_{i+1}} P_w^{\bar{s}_{i+1}}}{P_w^{s_i}} \frac{P_w'^{s_{i+1}}}{P_w^{s_{i+1}}} \end{aligned}$$

Using (8), we get

$$\frac{P_w{}'^{s_i}}{P_w^{s_i}} = P_n^{s_i} \cdot P_0^{s_i} + (1 - P_n^{s_i}) \frac{P_w'^{s_{i+1}}}{P_w^{s_{i+1}}} . \tag{11}$$

The probability that the next branch outcome will be 1 is:

$$P(X_{t+1} = 1 | x_1^T) = \frac{P_w'^{\lambda}}{P_w^{\lambda}} .$$

Using (11) we get:

$$P(X_{t+1} = 1 | x_1^T) =$$
$$P_n{}^{\lambda} \cdot P_0{}^{\lambda} + \left(1 - P_n{}^{\lambda}\right)\left(P_n{}^{s_1} \cdot P_0^{s_1} + \left(1 - P_n{}^{s_1}\right) \cdots \right)$$

or :

$$\begin{aligned} &P(X_{t+1} = 1 | x_1^T) = \\ &P_n{}^{\lambda} \cdot P_0{}^{\lambda} + \left(1 - P_n{}^{\lambda}\right) P_n{}^{s_1} \cdot P_0{}^{s_1} \\ &+ \left(1 - P_n{}^{\lambda}\right)\left(1 - P_n{}^{s_1}\right) P_n{}^{s_2} \cdot P_0{}^{s_2} + \cdots \\ &+ \left(1 - P_n{}^{\lambda}\right) \cdots \left(1 - P_n{}^{s_{N-1}}\right) P_0{}^{s_N} \end{aligned} \tag{12}$$

From (10), $P_0^{s_i}$ depends only on the count of zeros and ones at a node, not on other nodes. We can interpret equation (12) as follows. The contribution $P_0^{s_i}$ of a node s_i is weighted by the parameter $P_n^{s_i}$ at that node, and by $(1 - P_n^{s_{i-1}}) \ldots (1 - P_n^{\lambda})$ at higher (closer to root) levels of the tree. Hence the algorithm gives higher weight to higher levels of the tree, unless $P_n^{s_i}$ at these levels is low, in which case lower levels also make a "contribution."

4. 3 Prediction by Partial Matching (PPM) Using Context Trees

Prediction by partial matching is an alternative to the context tree weighting method described in the previous section. With PPM we also use a context tree, but maintain only two items at each node s_i:

1. A count of the zeros a_{s_i} that occurred in context s_i.
2. A count of the ones b_{s_i} that occurred in context s_i.

We do not use the probability estimate $P_e(a_{s_i}, b_{s_i})$ or the weighted probability $P_w^{s_i}$. We walk the relevant path (determined by the current sequence) in the context tree, from leaf node toward root, until we find a node at which the count of zeros is different than the count of ones. In other words, we seek the deepest context that has sufficient information to make a prediction. Using $P = \dfrac{b + 1/2}{a + b + 1}$ from (2), we predict the next branch outcome is 1 with probability $\Phi(P)$, when $\Phi(P)$ is shown in Figure 5.

4. 4 Adding Address Information

We have assumed so far that the only information used to predict the next branch outcome is the global sequence of branches. We can add address information by creating a context tree in which the context is determined by concatenating *a* address bits to *h* bits of branch outcome history.

Example 5

If the branch address is ... 10 and the global history bits are ... 0 1 1 0 0, then the current context in the 4-level context tree shown in Figure 6 is 0 0 1 0. It consists of two address bits and two history bits.

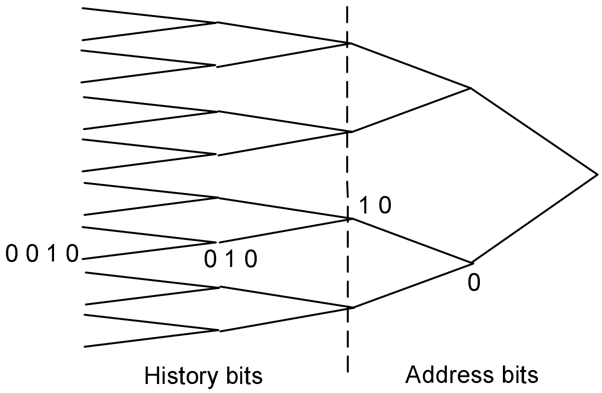

Figure 6 - Context tree using history bits and address bits.

This method of adding address information may be applied to both static trees and adaptive trees, described in the next section.

4. 5 Prediction Using Adaptive CTW and PPM Models

We limit the length of the sequence "remembered" by the context tree to the length of a history window. We "forget" updates done by bits that leave the window, and deallocate nodes that no longer contain information within the history window; this check is only necessary at nodes on the context path of a bit leaving the window. This method creates an *adaptive* context tree.

From equations (2) and (3) we get

$$P_e(x_1^{T+1}) = \frac{b+1/2}{a+b+1} P_e(x_1^T)$$

where a is the count of zeros and b is the count of ones in x_1^T. Using our shorthand notation $P_e^{'}$ for $P_e(x_1^{T+1})$ we have

$$P_e^{'} = \frac{b+1/2}{a+b+1} P_e. \qquad (13)$$

This update is done along the current context path for every new branch outcome, for both static and adaptive trees. For adaptive trees only, we maintain a shadow tree that runs behind the context tree by the length of the history window (that is, the shadow tree is updated by bits that leave the history window). To "forget" updates in the context tree, we undo the operation in equation (13) for every bit that leaves the history window (along the context path of that bit), by performing the reverse operation:

$$P_e = \frac{a_{\text{shadow}} + b_{\text{shadow}} + 1}{b_{\text{shadow}} + 1/2} P_e^{'}.$$

We extend the adaptive tree by adding two childrens at the leaf node of the current context path. We need two nodes because the adaptive CTW algorithm uses two nodes in the calculation of P_w.

Example 6

Assume the current sequence is ... 0 1 0 0 1 1 and the context tree is as shown in Figure 7a. The leaf node on the current context path is 1 0 0 1 1. Now that the context 1 0 0 1 1 occurs for a second time (it must have occurred once earlier, that's how it became part of the tree), we add two children nodes to it. The result is the extended tree in part (b) of the figure.

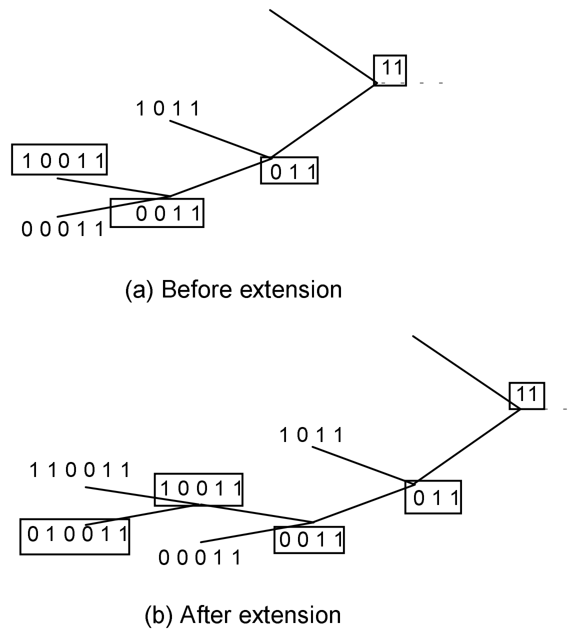

Figure 7 - Extending an adaptive tree. Nodes on the current context path shown in rectangles.

Below are the adaptive CTW and PPM algorithms.

1. Create a small balanced context tree.
2. Use the context tree to predict the next branch outcome, as described in section 4. 1 for CTW and in section 4. 3 for PPM.
3. Update the context tree with the observed branch outcome.
4. If the memory limit and maximum tree depth have not been exceeded, add two children nodes at the leaf node of the current context path.
5. "Forget" bits leaving the history window. Deallocate empty nodes.

Up to now we have described the application of two known data compression algorithms, CTW and PPM, to branch prediction. We have extended these algorithms by adding address information and by making the context tree adaptive. In the rest of the paper we describe the simulation of these algorithms and performance results.

5. Performance Evaluation

We have simulated the following branch prediction algorithms.

- Local

  Two level algorithm. The first level, indexed by the lower address bits of the branch address, consists of local history registers. The second level consists of a single table of 2-bit counters shared by all the local history registers.

- Global

  A table of 2-bit counters indexed by the global history register.

- Global CTW (GCTW)

  CTW algorithm using a balanced static tree.

- Global PPM (GPPM)

  PPM algorithm using a balanced static tree.

- Adaptive Global CTW (AGCTW)

  CTW algorithm using an adaptive tree.

- Adaptive Global PPM (AGPPM)

  PPM algorithm using an adaptive tree.

- Adaptive (tree), (branch) Address, Global (history) CTW (AAGCTW)

  CTW algorithm using an array of adaptive trees indexed by the lower branch address bits. The global branch history is used to update the context information in the context trees.

- Adaptive (tree), (branch) Address, Global (history) PPM (AAGPPM)

  PPM algorithm using an array of adaptive trees indexed by the lower branch address bits. The global branch history is used to update the context information in the context trees.

- Combined Local and AAGPPM

Regarding the simulation parameters shown in Table 1 we note the following.

1. The static tree algorithms (GCTW, GPPM) use a preallocated, fixed sized, tree of depth 14.
2. The adaptive algorithms (AGCTW, AGPPM, AAGCTW, AAGPPM) expand the tree up to the maximum depth indicated. We allow a depth of 20 for AAGCTW and AAGPPM versus a depth of 25 for AGCTW and AGPPM because AAGCTW and AAGPPM use an array of trees, while AGCTW and AGPPM use a single tree.
3. Both static and adaptive trees have the same upper limit on the number of tree nodes ($2^{15}-1$).
4. We have experimented with various counters. GCTW uses infinite counters for zeros and ones. GPPM, AGCTW, and AAGCTW use separate counters for zeros and ones that count up to 10 (decimal), after which the values of both zeros and ones counters is divided by two. The remaining algorithms use 2-bit saturation counters.
5. The adaptive tree algorithms employ a history window of 25000 branches.
6. Both global and local use history registers of length 14.
7. Local uses the lower 14 bits of the branch address to index the array of local history registers. AAGCTW and AAGPPM use the lower 8 bits of the branch address to index and array of adaptive trees.
8. The combined Local and AAGPPM method uses a table of saturated counters, similar to [10], to select the successful predictor.

5. 1 Benchmarks

We have used the following traces (see Tables 2 and 3):

- Five of the eight SPECint95 traces (go, compress, m88ksim, gcc, and perl)
- CorelDraw 6.0 (a desktop graphics program), from the SYSmark/32 benchmark
- Borland Paradox 7.0 (a database program), from the SYSmark/32 benchmark
- Microsoft Word 6.0 (word processing), from the SYSmark/NT benchmark
- Microsoft Excel 7.0 (spreadsheets), from the SYSmark/NT benchmark
- Transactional processing benchmark

We show branch prediction results for all the programs in Tables 4 and 5. To prevent overloading the graphs, in figures that show the behavior of the algorithms (Figure 8 to Figure 11) we only display the SPECint95 programs.

In all the simulations, the predictor's input is conditional branches only. The prediction rate is the ratio of the number of times the prediction was correct and the number of conditional branches.

5. 2 Simulation Results

Figure 8 illustrates the non-stationary behavior of gcc as the program executes close to 5,000,000 branches. The most striking observation is the large variation in the instantaneous prediction rate, which changes between 87% and 98% even in advanced stages of the program. This behavior motivated us to develop the adaptive tree algorithms (AGCTW, AGPPM, AAGCTW, AAGPPM), which adapt the context tree structure and parameters to the program behavior.

Figure 9 and Figure 10 display the data we have used to tune the parameters of the adaptive tree algorithms: the depth of the tree, the length of the history window, and the number of nodes in the tree. As shown, there are large differences between the programs in terms of the effect of the parameters and the point at which asymptotic behavior is reached. Considering this data, for the final results shown in Table 4 we have selected parameters (see Table 1) that allow programs to either reach asymptotic behavior or close to it.

Figure 11 shows that by using address bits (and an array of trees rather than a single tree) we need fewer levels in the context tree to achieve asymptotic results. For example, if we use one address bit we need an 15-level tree, but if we use 8 address bits we can reduce the depth of the tree to 10 levels. As shown in the figure, using address bits improves branch prediction by up to 7% for GO.

Table 4 shows results for two 2-level branch prediction methods (global and local), three context tree weighting algorithms (global CTW, adaptive global CTW, adaptive local CTW), three prediction by partial matching algorithms (global PPM, adaptive global PPM, adaptive local PPM), and a combined method (local and AAGPPM). It can be seen that PPM and CTW often achieve close results, although CTW uses more information to weight the context. The reason for this is that P_n, which determines how the contexts are weighted, is close to zero in inner nodes. As a result, CTW uses statistics taken mostly from leaf nodes, and often makes predictions close to PPM, which always uses only the leaf nodes.

Adaptive methods generally perform better than static methods, producing improvements of up to 7% (see the difference between GCTW and AGCTW for GC). The effectiveness of using address bits is highly program dependent, and the improvement ranges from 0% (CO) to 7% (GO). The combined predictor improves the results of GO, the program with the highest misprediction rate, by 0.85%, but is less effective for the other programs, whose prediction rate is already high.

6. Conclusions

Based on the close relationship between universal coding and universal prediction, we have applied a recent universal coding algorithm, context tree weighting, to branch prediction. To our best knowledge, this paper is the first investigation of using context tree weighting to predict branches. We have introduced the *adaptive* CTW and PPM algorithms. In trees with a limited number of nodes, the adaptive algorithm makes better use of the available resources by dynamically extending the tree in the direction of the current context path. While earlier work on PPM and branch prediction has been reported in [2], this paper is the first investigation of applying adaptive PPM to branch prediction.

The highest potential for improvement is in programs with difficult to predict branch behavior. This is where our results are most promising. Of the ten traces selected from the SPECint95, SYSmark/32, SYSmark/NT, and transactional processing benchmarks, the results for "go," the program with the highest misprediction rate, show that the combined local/AAGPPM predictor produces a prediction rate of 82.77%, up 3.5% from local (a two-level predictor) alone.

We hope these initial results will generate interest in the systematic application of information theory results to branch prediction. A lot remains to be done. We have not done sufficient "fine tuning" of a large number of parameters that may affect the prediction performance, and have not looked at many other possible variations of the algorithms. Another major issue is the practical implementation of the algorithms and their cost in hardware, a topic we are currently investigating.

7. Acknowledgements

This research was partially supported by a grant from Intel Israel, Haifa. We thank Michael Gutman, Lihu Rappoport, and Uri Weiser, for helpful discussions and for providing the traces used in this research.

8. Bibliography

[1] C. Bloom, PPMZ, http://www.vms.utexas.edu/~cbloom

[2] I-C. K. Chen, J. T. Coffey, and T. N. Mudge, "Analysis of Branch Prediction via Data Compression," *Proc. 7th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII),* Cambridge, MA, Oct. 1996.

[3] J. G. Cleary and I. H. Witten, "Data Compression Using Adaptive Coding and Partial String Matching," *IEEE Trans. on Communications,* pp. 396-402, vol. 32, no. 4, 1984.

[4] T. M. Cover, "Universal Gambling Schemes and the Complexity Measures of Kolmogorov and Chaitin," Dept. of Statistics, Standford University, no. 12, Oct. 1974.

[5] K. M. Curewitz, P. Krishanan, and J. S. Vitter, "Practical Prefetching via Data Compression," *ACM SIGMOD Int'l Conf. On Management of Data,* pp. 257-266, May 1993.

[6] L. D. Davisson, "The Prediction Error of Stationary Gaussian Time Series of Unknown Covariance," *IEEE Transactions on Information Theory*, pp. 527-532, Oct. 1965.

[7] M. Feder, N. Merhav, and M. Gutman, "Universal Prediction of Individual Sequences," *IEEE*

Transactions on Information Theory, pp. 1258-1270, July 1992.

[8] M. Feder and N. Merhav, "Relations Between Entropy and Error Probability," *IEEE Transactions on Information Theory*, pp. 259-266, Jan. 1994.

[9] J. L Kelly Jr., "A New Interpretation of Information Rate," Bell Systems Tech. Journal, pp. 917-926, 1956.

[10] S. McFarling, "Combining Branch Predictors," *WRL Technical Note TN-36,* Digital Equipment Corp., June 1993.

[11] R. E. Krichevsky and V. K. Trofimov, "The Performance of Universal Encoding," *IEEE Trans. Information Theory,* pp. 199-207, vol. 27, no. 2, 1981.

[12] A. Moffat, "Implementing the PPM Data Compression Scheme," *IEEE Trans. Comm.,* pp. 1917-1921, vol. 38, no. 11, 1990.

[13] T. N. Mudge, I-C. K. Chen, and J. T. Coffey, "Limits to Branch Prediction," *Technical Report CSE-TR-282-96*, University of Michigan, Jan. 1996.

[14] S. T. Pan, K. So, and J. T. Rahmeh, "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation," *Proc. ASPLOS V,* Boston, MA, Oct. 1992.

[15] J. Rissanen and G. G. Langdon, "Universal Modeling and Coding," *IEEE Transactions on Information Theory*, pp. 12-23, 1984.

[16] C. E. Shannon, "Prediction and Entropy of Printed English," *Bell Sys. Tech. Journal,* Vol. 30, pp. 51-64, 1951.

[17] J. S. Vitter and P. Krishnan, "Optimal Prefetching via Data Compression," *Proc. Of the 32nd Annual IEEE Symposium on Foundations of Computer Science,* pp. 121-130, Oct. 1991.

[18] F. Willems, Y. Shtarkov, and T. Tjalkens, "The Context-Tree Weighting Method: Basic Properties", *IEEE Transactions on Information Theory,* Vol. IT-41, No. 3, May 1995.

[19] R. N. Williams, *Adaptive Data Compression*, Kluwer, 1991.

[20] T.Y. Yeh and Y.N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," *Proc. 19th Annual Int'l Symp. on Computer Architecture,* Gold Coast, Australia, May 1992.

[21] C. Young, N. Gloy, and M. D. Smith, "A Comparative Analysis of Schemes for Correlated Branch Prediction," *22nd Annual Int'l Symp. on Computer Architecture,* Santa Marguerita, Italy, 1995.

[22] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Transactions on Information Theory*, pp. 530-536, Sept. 1978.

	Local	Global	GCTW	GPPM	AGCTW	AGPPM	AAGCTW	AAGPPM
Tree	N/A	N/A	Static	Static	Adaptive	Adaptive	Adaptive	Adaptive
Max tree depth	N/A	N/A	14	14	25	25	20	20
Number of nodes	N/A	N/A	$2^{15}-1$	$2^{15}-1$	$2^{15}-1$	$2^{15}-1$	$2^{15}-1$	$2^{15}-1$
Counters	2-bit	2-bit	infinite	Threshold 10	Threshold 10	2-bit	Threshold 10	2-bit
History window size	N/A	N/A	N/A	N/A	25000	25000	25000	25000
History reg. length	14	14	N/A	N/A	N/A	N/A	N/A	N/A
Lower address bits	14	N/A	N/A	N/A	N/A	N/A	8	8

Table 1 - Simulation parameters

Program	Label Used in Figures	Conditional Branches
go	GO	3414567
compress	CO	4291626
m88ksim	MK	4834240
gcc	GC	4919173
perl	PE	4224558

Table 2 - SPECint95 traces

Program	Label Used in Figures	Conditional Branches
CorelDraw 6.0	CD32	2551154
Borland Paradox 7.0	PD32	2048448
Microsoft Word 6.0	WDNT	3217719
Microsoft Excel 5.0	MXNT	1210418
Transactional processing	TPC	1329471

Table 3 - SYSmark/32, SYSmark/NT, and transactional processing traces

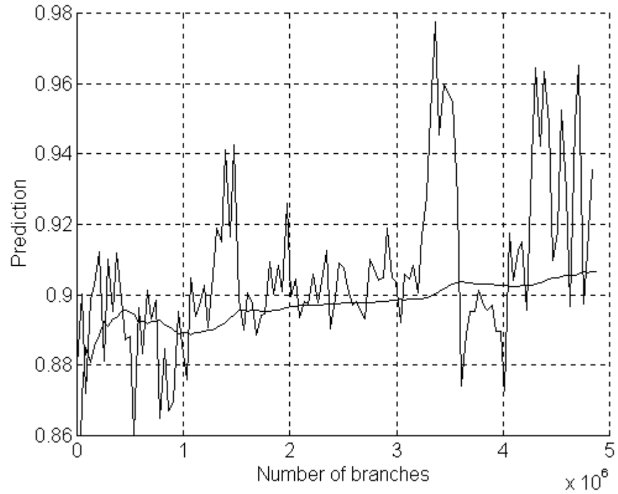

Figure 8 - Cumulative and instantaneous branch prediction rate for gcc (GC), using the adaptive PPM (AGPPM) algorithm.

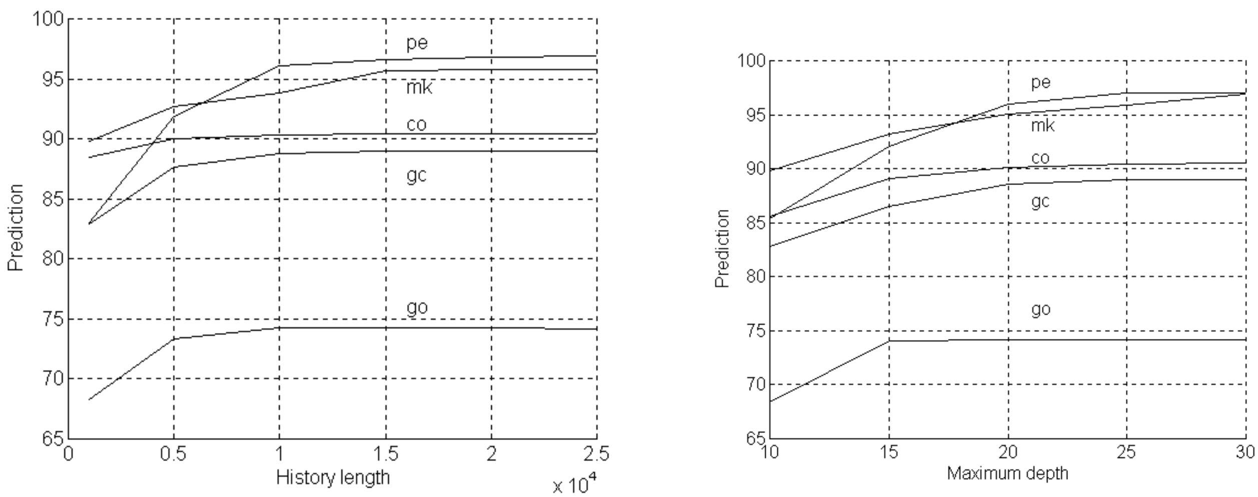

Figure 9 - History window length and tree depth parameters (AGPPM algorithm).

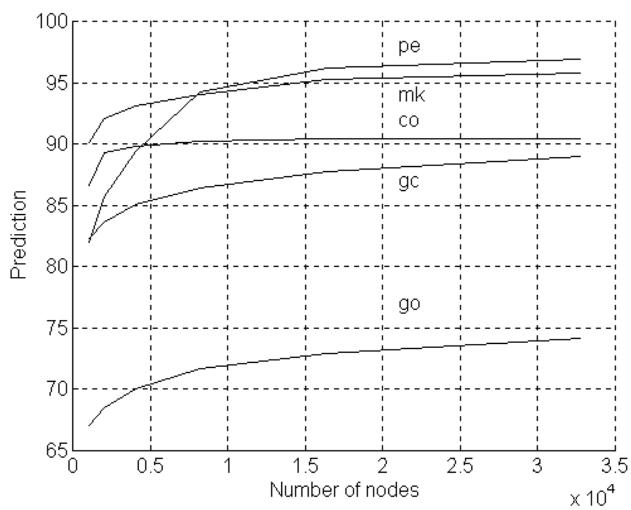

Figure 10 - The number of nodes in the tree and its effect on the branch prediction rate (AGPPM algorithm).

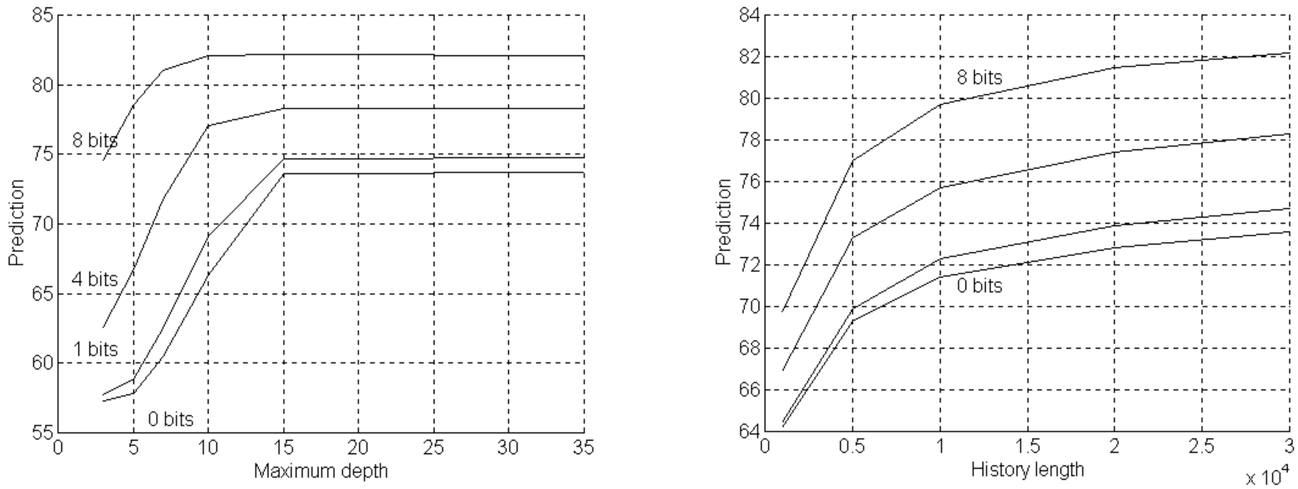

Figure 11 - Varying the number of address bits for GO, using the adaptive AAGPPM algorithm.

Algorithm\Program	GO	CO	MK	PE	GC
Global	73.17	89.23	92.71	91.50	86.58
Local	79.30	88.12	97.05	96.07	90.90
GCTW	71.39	88.27	93.48	92.02	82.97
GPPM	72.76	89.80	93.25	91.55	86.42
AGCTW	73.46	91.24	95.89	97.27	90.09
AGPPM	74.46	90.39	95.84	96.98	90.67
AAGCTW	81.52	91.31	96.84	97.80	93.22
AAGPPM	81.93	90.49	96.85	97.67	93.57
Local + AAGPPM	82.77	90.54	97.17	97.65	93.64

Table 4 - Branch prediction rate, SPECint95 programs. The predictor's input is conditional branches only. The prediction rate is the ratio of the number of times the prediction was correct and the number of conditional branches.

Algorithm\Program	CD32	PD32	WDNT	MXNT	TPC
Global	93.7	88.5	91.1	95.8	95.24
Local	98.66	95.35	96.11	92.2	89.97
GCTW	93.8	87.9	90.2	95.6	88.0
GPPM	93.8	88.3	91.2	95.5	89.5
AGCTW	96.78	87.33	94.31	95.14	89.48
AGPPM	96.65	87.71	94.41	95.16	89.26
AAGCTW	97.76	91.72	96.79	95.3	92.52
AAGPPM	97.5	91.97	96.81	95.05	92.4
Local + AAGPPM	98.2	94.48	97.20	95.07	94.87

Table 5 - Branch prediction rate, SYSmark/32, SYSmark/NT, and transactional processing traces. The predictor's input is conditional branches only. The prediction rate is the ratio of the number of times the prediction was correct and the number of conditional branches.