Potentials of Branch Predictors — from Entropy Viewpoints —

Takashi Yokota[1], Kanemitsu Ootsu[1], and Takanobu Baba[1]

Department of Information Science, Utsunomiya University,
7–1–2 Yoto, Utsunomiya-shi, Tochigi, 321–8585 Japan
{yokota, kim, baba}@is.utsunomiya-u.ac.jp

Abstract. Predictors essentially predicts the most recent events based on the record of past events, history. It is obvious that prediction performance largely relies on regularity–randomness level of the history. This paper concentrates on extracting effective information out from branch history, and discuss expected performance of branch predictors. For this purpose, this paper introduces entropy point-of-views to quantitative characterization of both program behavior and prediction mechanisms. This paper defines two new entropies independent of prediction methods and other two entropies dependent on predictor organization. These new entropies are useful tools for analyzing upper-bound of prediction performance. This paper shows some evaluation results for typical predictors.

1 Introduction

Predictors are inevitable in the state-of-the-art microprocessor cores. Prediction mechanism is essential for any speculation features in micro-architecture. However, speculation essentially incorporates prediction accuracy, i.e., more precise prediction does better performance and vice versa.

The essential and fundamental property of today's most predictors is that it predicts based on past events, history. Many prediction methods have been proposed, however, most of them discuss relative performance improvements to some typical and well-known prediction method. Until now, no one knows the possible performance of predictors, i.e., absolute maximum. For example, assuming that predictor A performs 5% better than predictor B, we cannot discuss any more for further improvements, because we cannot know possible maximum performance.

This paper presents theoretical views on branch predictors so that we can discuss potentials of branch predictors. Our major focus is to represent information of past events and to clarify possible maximum performance of branch predictors. We introduce classical information theory from Shannon. Originally, entropy quantitatively represents essential information of the forthcoming symbol, based on the existing data.

A branch predictor intends to extract essentially the same information with Shannon's entropy, based on the past branch results. Shannon discussed relatively large set of symbols S, say the alphabet. Fortunately, a branch predictor

uses a binary symbol, i.e., a branch will be taken or untaken. This one-bit symbol helps us to discuss potentials of prediction performance.

The remainder of this paper is organized as follows. We first give the overview of Shannon's information theory and describe our targeted branch predictors in Section 2. After the preliminaries we discuss information from two aspects: entropies independent of prediction mechanisms (in Section 3), and entropies based on predictor organization (in Section 4). Section 5 shows evaluation results from various perspectives. Section 6 shows related work and Section 7 concludes this paper.

2 Preliminaries

2.1 Information Entropy

This paper stands on Shannon's information entropy[1]. We summarize the fundamentals. Assume that we are discussing an entropy $H(S)$ of a Markovian information source S that produces a string of symbols. Instead of the entropy of S itself, we first discuss the augmented adjoint source of S. An n-th order augmented adjoint source of S, denoted by $\overline{S}^n$, generates n consecutive symbols. The entropy of the n-th order augmented adjoint source $H(\overline{S}^n)$ is given as the following equation:

$$H(\overline{S}^n) = -\sum_i p(S_i^n) \log_2 p(S_i^n), \tag{1}$$

where $p(S_i^n)$ represents the probability of an individual symbol S_i^n that comprises consecutive n original symbols.

$H(\overline{S}^n)$ represents information in consecutive n symbols, and it monotonically increases as n increases. Differential coefficient of $(H\overline{S}^n)$ at n shows information of single symbol. Thus, when $(n+1)$-th order augmented adjoint entropy $H(\overline{S}^{n+1})$ is given, the n-th approximation of the entropy $H(S)$ of the objective Markovian information source is given by

$$H^n(S) = H(\overline{S}^{n+1}) - H(\overline{S}^n). \tag{2}$$

Therefore, the true value of the targeted entropy is given by limiting n to infinity:

$$H(S) = \lim_{n \to \infty} H^n(S). \tag{3}$$

The entropy $H(S)$ provides essential information of the next symbol. The entropy also presents the predictability of the forthcoming symbol.

2.2 Branch Predictors

This paper discusses performance issues in branch predictors. Our approach in this paper is to discuss generic and practical issues. To simplify the discussion

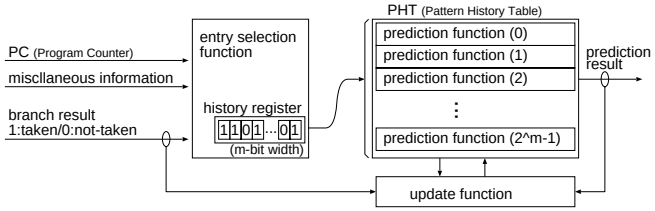


Fig. 1. Generalized organization of table-formed branch predictor.

without loss of generality, we assume some typical organization and mechanisms in branch predictors.

We specifically use table-formatted predictors as shown in Figure 1. A predictor is organized by three major functions. The predictor has one or more prediction functions, where each prediction function is basically independent from each other. The entry selection function selects an appropriate prediction function according to given selection rule. Update function updates the prediction function. Following the conventions in two-level predictors, we use m bits of 'history register' (HR for short) as a result of selection function to point one of the prediction functions. We also use 'pattern history table' (PHT) that is an aggregation of prediction functions. Each entry of PHT consists of a prediction function. We use bimodal (`bimode`)[2], two-level (`2level`)[3], gshare (`gshare`)[4], and perceptron (`perceptron`)[5,6] branch predictors. All of them follow the simple organization shown in Figure 1.

`bimode`, `2level` and `gshare` differ in selection function; `bimode` uses a simple hash function of program counter, `2level` uses the latest m results of branch execution, and `gshare` uses the exclusive-or result of branch history and program counter. However, they typically use the same prediction function: a two-bit saturation counter that counts the number of 'taken' results and if the result is untaken, the counter is decremented.

`perceptron` uses hash function of program counter as its selection function. Its prediction function is distinguishing; it is based on a neural-network Perceptron. Each prediction function makes use of the latest h branch results and corresponding h weight values. Each weight value w_i is a signed integer with an arbitrary length, say 8 bits. It calculates the weighted sum: $s = \sum_i w_i \cdot b_i$ where b_i is 1 if the corresponding branch result is 'taken' and $b_i = -1$ otherwise. If the resulting sum s is positive, 'taken' is predicted, otherwise, 'untaken' is predicted. Each PHT entry consists of a set of weight values. Update function modifies each of weight values according to the prediction result (hit or mishit).

3 Entropies Independent of Prediction Mechanisms

Section 2.1 discusses information in a string of symbols that are generated sequentially. We can apply the discussion to a string of branch results by simple

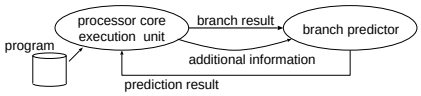


Fig. 2. Processor core as an information source to branch predictor.

substitution of 'branch results' for 'symbols.' And, by careful observation of program execution but prediction mechanisms, we offer the following two entropies.

3.1 Execution Unit as Information Source

First viewpoint is an execution unit in a processor core. As the processor execute a program, branch instructions are executed according to the program. The execution unit explicitly generates branch results so that the results are used for prediction in the branch predictor. Figure 2 illustrates it.

We can discuss entropy for the series of branch results. We consider Markovian information source B, i.e., the execution unit. By considering n consecutive branch results, we can define n-th order augmented adjoint information source $\overline{B}^n$. By simple application of Section 2.1, we can define the entropy of the n-th order augmented adjoint source $H(\overline{B}^n)$, and the n-th approximation of the targeted entropy $H(B)$, $H^n(B)$. The essential entropy $H(B)$ is given by limiting n to infinity as Equation (3). We call the new entropy **Branch History Entropy** (BHe).

3.2 Branch Instructions as Information Sources

Another viewpoint is individual branch instruction. Each branch instruction has its *own* branch history. Thus, we can define an entropy for each branch instruction. We consider that each branch instruction is a Markovian information source I_i. By applying the original entropy definition given in Section 2.1, we can define the entropy of i-th branch instruction, $H(I_i)$. Overall entropy is given as the average of $H(I_i)$, i.e., $H(I) = \frac{1}{N_b}\sum_i n_i \cdot H(I_i)$ where n_i is the execution count of the i-th branch instruction and N_b is the total number of branch executions. We call the entropy **Branch Instruction Entropy** (BIe).

3.3 Fundamental Properties of Proposed Entropies

These entropies have a common important feature: these are independent of any prediction mechanisms. They represent essential information that is extracted through program execution, i.e., they represent program behavior.

BHe represents information only from branch history. This means that BHe shows the certainty degree of the forthcoming branch result, with no any additional hints. The branch history itself contains no information on individual

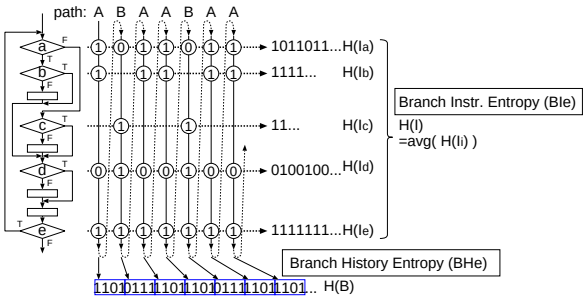


Fig. 3. Example path repetition and BHe, BIe entropies.

branch instructions but sequence of branch results. In practical situations, a string of branch history often shows a particular execution path.

On the other hand, BIe is basically defined for each branch instruction. It essentially does not represent 'execution path' information, but it represents the local regularity of program behavior.

We claim that BHe represents information of *global* history, and BIe shows *local* (or *per address*) history information. We will discuss further with a practical example.

As an example situation, assume that a loop has two frequently-executed paths A and B that show branch history '1011' and '0111,' respectively. These paths are regularly executed as simple repetition of A→A→B. Figure 3 shows BHe and BIe in such situation. Solid vertical arrows show loop iterations and circles designate branch instructions and their branch decisions. Each horizontal dotted line shows per-address branch history at the corresponding instruction.

Executed paths are recorded in the BHe trace, and each BIe trace shows local history at branch instruction. In this example, high regularity in path execution reflects the regularity of the local history. This observation tell us that BHe and BIe are not strictly orthogonal but correlated at a considerable level.

4 Entropies in Prediction Functions

4.1 Information to Each Prediction Function

We will enter specific discussion on fundamental organization of predictors. A string of branch results, originated by the processor core, inherently contains the first-order information. We can consider that the information flows along the predictor organization. Figure 4 illustrates the flow. The information is poured into the 'entry selection function' and reaches individual entry of PHT, prediction function. But the original information is divided and only a segment of information is delivered into each prediction function.

This observation drives us to different entropy definition. Each prediction function has its own information of branch results, on which we can define en-

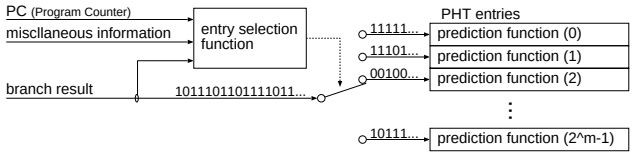


Fig. 4. Flow of branch result information.

tropy. Similarly to the Branch Instruction Entropy discussion, we can define entropy of each prediction function. Input sequence to a prediction function E_i has entropy $H(E_i)$. The overall entropy is given by $H(E) = \frac{1}{N_e} \sum_i e_i \cdot H(E_i)$ where e_i is the reference count of the i-th prediction function E_i and N_e denotes the total number of references to PHT. We call the entropy **Table Entry Entropy** (TEe).

4.2 Information in Imbalanced References

The first-order information bifurcates at the entry selection function, and some portion of the original information is lost. To compensate this, we define **Table Reference Entropy** (TRe). TRe represents effective number of active entries. Following to the discussion in the previous section, e_i is the number of references to i-th prediction function, and N_e is the total number of PHT references. $r_i = e_i/N_e$ shows the probability of reference on the i-th entry, prediction function. Table Reference Entropy is given by $H(R) = -\sum_i r_i \log_2 r_i$.

4.3 Discussion

TEe and TRe have different standpoints, but their origin is common, the first-order branch information. TEe shows the practical information poured into each prediction function, i.e., 'per-predictor' information. This is very likely to BIe as a (quasi-)orthogonal measure to BHe. Low TEe means that each predictor input has low information and, thus, the predictor is ease to predict.

Figure 5 shows the same example with Figure 3, but the figure shows sequences of branch history at each prediction function. The first-order information is delivered to individual PHT entry, which is prediction function, similarly to individual branch instruction. Most paths contain several or more lengths of branch history, thus, the first-order information is delivered to more destinations than those in the BIe condition. However, we can expect lower information entropy, thus higher prediction accuracy, from this entropy.

We can further discuss TEe and TRe entropies under the specific branch prediction mechanisms given in Section 2.2. `bimode` and `perceptron` predictors use a hash function of program counter. If these predictors have sufficient PHT entries, most of active entries correspond to their own branch instructions and, thus, TEe is very close to BIe. If only a limited number of instructions dominate, TRe becomes low.

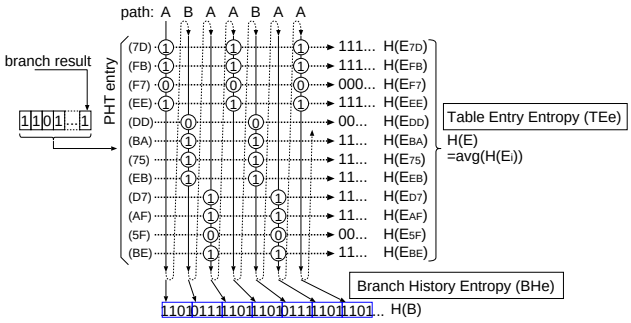


Fig. 5. Example path repetition and BHe, TEe entropies.

In `2level` predictors, history register (HR) shows the latest m branch results. Since TRe is based on the values of HR, the entropy is just the same as m-th order augmented adjoint source entropy, i.e., $H(\overline{S}^m)$ given by Equation (1).

`gshare` predictor uses an XOR'ed value of branch history and program counter. This scatters the use of PHT entries. Thus, TRe shows wide variations of local history at branch instructions.

5 Evaluation

5.1 Evaluation Environment

We extended the `sim-bpred` simulator in the SimpleScalar toolset[7] so that our defined entropies are measured. We also implemented the perceptron predictor in `sim-bpred`. PISA instruction set was used. Some programs in SPEC CPU2000[8] benchmark are compiled by gcc 2.7.2.3 PISA cross compiler. The benchmark set has variety of problem size: we used 'train.'

Entropies and prediction hit ratio are measured in every 1,000,000 (1M) branches time-window. Size of the time-window is important for accuracy of measured entropy. Although more samples produces more precise results, long time-window may bury important characteristics of 'phases' in program execution. We consider the 1 million branch time-window is proper[5, 6].

Predictors use the same size of PHT entry, $2^{12} = 4096$ entries, thus HR is 12-bit width. Since theoretical entropy definition (Equation (3)) is not practical, we measured 14-, 15-, 16-, 17-, and 18-th order augmented adjoint entropies and the essential entropy is calculated by the least-square method of these five entropies.

5.2 Potentials of Branch Predictors

We firstly show time-sequence plot of predictor hit ratio and some of proposed entropies in Figure 6. We can find that they are considerably correlated to each

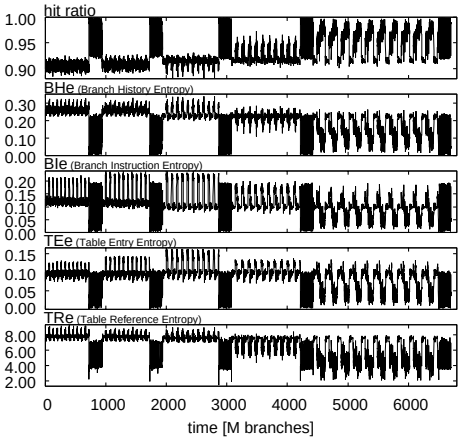


Fig. 6. Time sequence plot of prediction hit ratio, BHe, BIe, TEe and TRe.

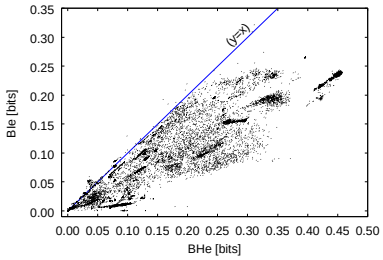


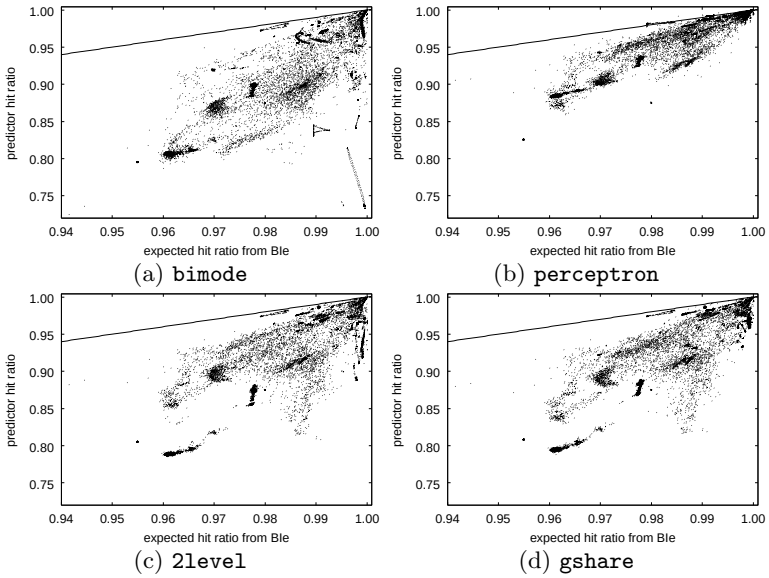
Fig. 7. Correlation between BHe and BIe.

other. We will discuss potentials of branch predictors by analyzing correlations in detail.

As discussed in Section 3.3, BHe and BIe have different standpoints, but they are not always orthogonal. Figure 7 shows the correlation between BHe and BIe. Each dot shows BHe and BIe at the corresponding time-window. Dots widely scatter, but clear correlation is observed.

Important fact is that most dots are below the $y = x$ line. This means that BHe is larger than BIe at most measured points. Since BHe and BIe correspond global and local history, respectively, this result says that local history shows higher level of regularity and, thus, higher prediction performance than those of global history.

Essential entropy definition for binary-event system is given as $f(p) = -p \log_2 p - (1-p) \log_2 (1-p)$, where p denotes the probability of the event. When an entropy value ε is given, we can estimate the originated probability p by the inverse function of $f(p)$, $f^{-1}(\varepsilon)$. We call the estimated probability **expected hit ratio** in this paper.



(a) `bimode` (b) `perceptron`

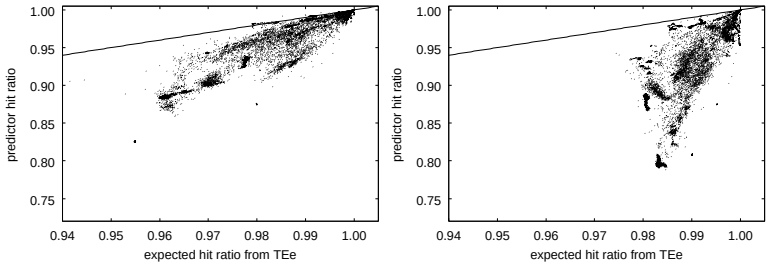
(c) `2level` (d) `gshare`

Fig. 8. Expected hit ratio from BIe and predictor hit ratio.

Figure 8 shows correlation of the expected hit ratio by BIe and actual prediction hit ratio. Similarly to Figure 7, each dot represents expected hit ratio by BIe and predictor's performance (hit ratio) at the corresponding time-window. Each graph in the figure shows $y = x$ line. The line represents performance criterion: dots located above the line show that the predictor performs beyond expectation. In Figure 8 cases, no predictor exceeds the criteria and the $y = x$ line shows potential performance.

Expected hit ratio can also be derived by other entropy metrics; BHe and TEe. These entropies also show similar plots to that of BIe. Figure 9 shows TEe plots of `perceptron` and `gshare` predictors. Note that expected hit ratio by BIe shows a generic criteria independent of predictor organization, and that expected ratio by TEe shows a specific potential.

Actual and expected hit ratios in each application are summarized in Table 1. Each fraction shows an average value throughout the execution of the corresponding application. In most application except `255.vortex`, TEe is the best in expected hit ratio. Practically, TEe values show potentials of predictors. Note that BIe values are very close to those of TEe; their difference is less than 1 percent in most applications. However, populations of those entropies are very different: population of BIe is the number of executed branch instructions in the time-window, and TEe population scatters according to the changes in history register. As a typical example of `164.gzip`, populations of BIe and TEe are about 100 and 1,000, respectively. This means that, in actual programs, branch instructions act very regularly as well as prediction functions.



(b) `perceptron` (d) `gshare`

Fig. 9. Expected hit ratio from TEe and predictor hit ratio.

Table 1. Summary of prediction hit ratios and expected hit ratios from proposed entropies.

	actual hit ratios				expected hit ratios		
benchmark	`bimode`	`2level`	`gshare`	`perceptron`	BHe	BIe	TEe(`gshare`)
(CINT2000)							
164.gzip	.9241	.9370	.9363	.9467	.9702	.9853	.9890
175.vpr(pl)	.8963	.8720	.8812	.9333	.9532	.9777	.9808
175.vpr(rt)	.8707	.9013	.8965	.9110	.9402	.9720	.9837
176.gcc	.9134	.9098	.9140	.9760	.9758	.9890	.9892
181.mcf	.8959	.9382	.9261	.9468	.9723	.9806	.9892
197.parser	.8939	.9203	.9252	.9509	.9601	.9837	.9912
254.gap	.9359	.9475	.9519	.9799	.9832	.9927	.9942
255.vortex	.9769	.9318	.9642	.9976	.9881	.9989	.9967
256.bzip2	.9824	.9820	.9841	.9877	.9913	.9937	.9956
300.twolf	.8123	.7979	.8042	.8902	.9108	.9636	.9839
(CFP2000)							
168.wupwise	.8556	.9275	.9540	.9859	.9874	.9962	.9994
171.swim	.9913	.9926	.9941	.9972	.9963	.9976	.9979
172.mgrid	.9745	.9755	.9754	.9807	.9791	.9811	.9860
173.applu	.7569	.9719	.9736	.9985	.9899	.9986	.9993
177.mesa	.9829	.9861	.9876	.9930	.9965	.9985	.9987
179.art	.9071	.9911	.9910	.9916	.9959	.9980	.9986
183.equake	.8839	.9660	.9763	.9801	.9881	.9937	.9981
188.ammp	.9659	.9787	.9800	.9854	.9820	.9905	.9931
301.apsi	.9727	.9707	.9833	.9925	.9845	.9919	.9925

6 Related Work

Major contribution of this paper can be described in two perspectives; first one is quantitative representation of program behavior from a sequence of branch results, and the other one is estimation of potential prediction performance.

Many researches have concentrated to imbalanced feature in program execution. Tyson et al.[9] show some imbalanced feature in taken/untaken sequences

of branch results. They show four typical cases; long consecutive takens, long consecutive untakens, a small number of untakens in long consecutive takens, and other patterns. Kise et al.[10] discuss a new predictor based on extremely biased branches. Such classifications help prediction, but no quantitative discussions are made on imbalanced features.

Periodicity is possibly a quantitative measure of program behavior. Freitag et al.[11] proposed Dynamic Periodicity Detector (DPD) by examination of sequences of data values that appear during program execution. Fourier Analysis Branch (FAB) predictor, proposed by Kampe et al.[12], uses the Fourier coefficients for representing periodicity for branch prediction purpose. Periodicity offers quantitative representation, however, it does not show essential information.

Mudge and Chen et al.[13, 14] present limits in the prediction performance based on prediction using the Partial Matching (PPM) method. They use m-th order Markov predictor and underlying idea is very similar to ours. Driesen et al.[15, 16] discuss limits of indirect branch prediction from a different point of view from that used in this paper. Jha et al.[17] also use a Markovian model to represent an optimal prediction mechanism for a given history length. Vintan et al.[18] discuss prediction limits based on unbiased branches. Idealistic predictors discuss substantial limits on prediction performance in Championship Branch Prediction competition (CBP [19, 20]). These researches are not successful for quantitative representation of regularity/randomness features in program execution, as our defined entropies do. Our preliminary results are found in [21, 22]. Our approach to the limits on prediction is unique in its theoretical and quantitative approach based on information entropy.

7 Concluding Remarks

Prediction performance essentially relies on the nature of past events. Thus, modern predictors enter detailed discussion to effectively extract useful information on prediction and improves performance. But theoretical limit on prediction performance was unclear.

This paper introduces information entropy concept to clarify theoretical limits in branch prediction. Our approach has two aspects: one is independent of prediction methods and the other one is dependent on predictor organization. We proposed two entropies, BHe and BIe, to represent *global* and *local* features in branch history. Furthermore, we defined TEe and TRe entropies for typical table-formatted predictors. BHe, BIe and TEe entropies can derive expected prediction performance, i.e., limits on prediction. BHe and BIe show prediction limits by global and local history, respectively. TEe shows theoretical limits on the predictor organization.

Potentials in branch prediction is calculated on a time-window basis. This offers a detailed criterion for program execution phases as well as applications themselves. And the evaluation results reveal the potentials are high and we have still large rooms to improve prediction performance.

References

1. C. E. Shannon, "A mathematical theory of communication," Bell System Technical Journal, Vol.27, pages 379–423 and 623–656, Jul. and Oct. 1948.
2. J. E. Smith, "A Study of Branch Prediction Strategies," Proc. ISCA '81, pp.135–148, May 1981.
3. T.-Y. Yeh, Y. N. Patt, "Two-Level Adaptive Branch Prediction," Proc. MICRO24, pp.51–61, Nov. 1991.
4. S. McFarling, "Combining Branch Predictors," Tech. Report TN–36, Digital Equipment Corp., Western Research Lab., Jun. 1993.
5. D. A. Jiménez, C. Lin, "Dynamic Branch Prediction with Perceptrons," Proc. HPCA–7, pp.197–206, Jan. 2001.
6. D. A. Jiménez, "Piecewise Linear Branch Prediction," Proc. ISCA '05, pp.382–393, 2005.
7. SimpleScalar LLC, http://www.simplescalar.com/
8. Standard Performance Evaluation Corporation, "SPEC CPU2000 V1.3," http://www.spec.org/cpu2000/, 2004.
9. G. Tyson, et al., "Limited Dual Path Execution," Tech. Report CSE–TR–346–97, Univ. Michigan, 1997.
10. K. Kise, et al., "The Bimode++ Branch Predictor Using the Feature of Extremely Biased Branches," IPSJ SIG Tech. Report, Vol.2005, No.7, pp.57–62, Jan. 2005.
11. F. Freitag, et al., "A Dynamic Periodicity Detector: Application to Speedup Computation," Proc. IPDPS'01, Apr. 2001.
12. M. Kampe, et al., "The FAB Predictor: Using Fourier Analysis to Predict the Outcome of Conditional Branches," Proc. HPCA'02, pp.223–232, Feb. 2002.
13. I-C. K. Chen, et al., "Analysis of Branch Prediction via Data Compression," Proc. ASPLOS–VII, pp.128–137, Oct. 1996.
14. T. Mudge, et al., "Limits to Branch Prediction," Tech. Report CSE–TR–282–96, Univ. Michigan, Feb. 1996.
15. K. Driesen, U. Hölzle, "Limits of Indirect Branch Prediction," Tech. Report TRCS97–10, Computer Science Dept., Univ. California, Santa Barbara, Jun. 1997.
16. K. Driesen, U. Hölzle, "Multi-stage Cascaded Prediction," Tech. Report TRCS99–05, Computer Science Dept., Univ. California, Santa Barbara, Feb. 1999.
17. S. Jha, et al., "Formal Analysis of Branch Prediction Algorithm," Tech. Report, Computer Science, Carnegie Mellon Univ., 1998.
18. L. Vintan, et al., "Understanding Prediction Limits Through Unbiased Branches," Proc. ACSAC 2006, LNCS 4186, pp.480–487, 2006.
19. "The 1st JILP Championship Branch Prediction Competition," http://www.jilp.org/cbp/, 2004.
20. "The 2nd JILP Championship Branch Prediction Competition," http://camino.rutgers.edu/cbp2/, 2006.
21. T. Yokota, et al., "Entropy Properties in Program Behaviors and Branch Predictors," Proc. IASTED PDCS 2006, pp.448–453, Nov. 2006.
22. T. Yokota, et al., "Introducing Entropies for Representing Program Behavior and Branch Prediction Performance," Proc. 2007 Workshop on Experimental Computer Science, ACM digital library, Jun. 2007.