

Applying Machine Learning for Ensemble Branch Predictors

Gabriel H. Loh¹ and Dana S. Henry²

¹ Yale University, Department of Computer Science

² Yale University, Department of Electrical Engineering
New Haven, CT, 06520, USA

`gabriel.loh@yale.edu`, `dana.henry@yale.edu`

Abstract. The problem of predicting the outcome of a conditional branch instruction is a prerequisite for high performance in modern processors. It has been shown that combining different branch predictors can yield more accurate prediction schemes, but the existing research only examines selection-based approaches where one predictor is chosen without considering the actual predictions of the available predictors. The machine learning literature contains many papers addressing the problem of predicting a binary sequence in the presence of an ensemble of predictors or experts. We show that the Weighted Majority algorithm applied to an ensemble of branch predictors yields a prediction scheme that results in a 5-11% reduction in mispredictions. We also demonstrate that a variant of the Weighted Majority algorithm that is simplified for efficient hardware implementation still achieves misprediction rates that are within 1.2% of the ideal case.

1 Introduction

High accuracy branch prediction algorithms are essential for the continued performance increases in modern superscalar processors. Many algorithms exist for using the history of branch outcomes to predict future branches. The different algorithms often make use of different types of information and therefore target different types of branches.

The problem of branch prediction fits into the framework of the machine learning problem of sequentially predicting a binary sequence. For each *trial* of the learning problem, the branch predictor must make a prediction, and then at the end of the trial, the actual outcome is presented. The prediction algorithm then updates its own state in an attempt to improve future predictions. In each round, the algorithm may be presented with additional information, such as the address of the branch instruction. All predictions and all outcomes are either 0 or 1 (for branch not-taken or branch taken, respectively). The prediction algorithm is called an *expert*.

Algorithms such as the Weighted Majority algorithm and the Winnow algorithm have been proposed for learning problems where there are several experts, or an *ensemble* of experts, and it is desired to combine their “advice” to form

one final prediction. Some of these techniques may be applicable to the prediction of conditional branches since many individual algorithms (the experts) have already been proven to perform reasonably well. This study explores the application of the Weighted Majority algorithm for the dynamic prediction of conditional branches.

The Conditional Branch Prediction Problem Conditional branches in programs are a serious bottleneck to improving the performance of modern processors. Superscalar processors attempt to boost performance by exploiting *instruction level parallelism*, which allows the execution of multiple instructions during the same processor clock cycle. Before a conditional branch has been resolved in such a processor, it is unknown which instructions should follow the branch. To increase the number of instructions that execute in parallel, modern processors make a *branch prediction* and speculatively execute the instructions in the predicted path of program control flow. If later on the branch is discovered to have been mispredicted, actions are taken to recover the state of the processor to the point before the mispredicted branch, and execution is resumed along the correct path.

For each branch, the address of the branch is available to the predictor, and the predictor itself may maintain state to track the past outcomes of branches. From this information, a binary prediction is made. This is the *lookup phase* of the prediction algorithm. After the actual branch prediction has been computed, the outcome is presented to the prediction algorithm and the algorithm may choose to update its internal state to (hopefully) make better predictions in the future. This is the *update phase* of the predictor. The overall sequence of events can be viewed as alternating lookup and update phases.

The Binary Prediction Problem Consider the problem of predicting the next outcome of a binary sequence based on observations of the past. Borrowing from the terminology used in [20], the problem proceeds in a sequence of *trials*. At the start of the trial, the prediction algorithm is presented with an *instance*, and then the algorithm returns a binary prediction. Next, the algorithm receives a *label*, which is the correct prediction for the instance, and then the trial ends.

Now consider the situation where there exists multiple prediction algorithms or *experts*, and the problem is to design a *master algorithm* that may consider the predictions made by the n experts E_1, E_2, \dots, E_n , as well as observations of the past to compute an overall prediction. Much research has gone into the design and analysis of such master algorithms. One example is the Weighted Majority algorithm which tracks the performance of the experts by assigning weights to each expert that are updated in a multiplicative fashion. The master algorithm “homes in” on the best expert(s) of the ensemble.

The binary prediction problem fits very well with the conditional branch prediction problem. The domain of dynamic branch prediction places some additional constraints on the prediction algorithms employed. Because these techniques are implemented directly in hardware, the algorithms must be amenable to efficient implementations in terms of logic gate delays and the storage neces-

sary to maintain the state of the algorithm. In this paper, we present a hardware unintensive variant of the Weighted Majority algorithm, and experimentally demonstrate the performance of our algorithm.

Paper Outline The rest of this paper is organized as follows. Section 2 discusses some of the relevant work from machine learning and branch prediction research. Section 3 briefly explains our methodology for evaluating the prediction algorithms. Section 4 analyzes some of the existing prediction algorithms, and provides some motivation for the use of the Weighted Majority algorithm. Section 5 describes our hardware implementable approximations to the Weighted Majority algorithm, and provides experimental results. Section 6 concludes and describes directions for future research.

2 Related Work

The most relevant work to this paper falls into two categories. Machine learning algorithms for predicting binary sequences comprise the first category. The second group consists of past research in combining multiple branch predictors to build better prediction algorithms. Other areas of artificial intelligence have been applied to the problem of conditional branch prediction, such as perceptron and neural networks [5], [15] and genetic algorithms [9].

Machine Learning Algorithms for Predicting Binary Sequences The problem of predicting the outcomes of a sequence of binary symbols has received much attention in the machine learning research. Littlestone and Warmuth introduced the Weighted Majority algorithm [20], which was independently proposed by Vovk [24]. The Weighted Majority algorithm works with an ensemble of experts, and is able to predict nearly as well as the best expert in the group without any *a priori* knowledge about which experts perform well. Theoretical analysis has shown that these algorithms behave very well even if presented with irrelevant attributes, noise, or a target function that changes with time [12], [19], [20].

The Weighted Majority algorithm and other multiplicative update master algorithms have been successfully applied to several problem domains including gene structure prediction [7], scheduling problems [3], and text processing [11] for example.

Techniques for Combining Branch Predictors Research in the computer architecture community has gone into the design of techniques that combine multiple branch predictor experts into a hybrid or ensemble predictor. The common theme among these techniques is that the master algorithm is an *expert-selection* algorithm. By expert-selection, we mean that the final prediction is formed by choosing one of the n experts based solely on the past performance of the experts. The current predictions made by the experts are not factored into the decision process.

The earliest published branch predictor that took advantage of multiple experts is the tournament predictor [21]. The tournament algorithm is only capable

of choosing between two experts, E_0 and E_1 . For each branch, a small counter (typically only 2 bits wide) keeps track of the past performance of the two experts. The algorithm always selects the expert that has performed the best in recent trials. The predictor is shown schematically in Figure 1a.

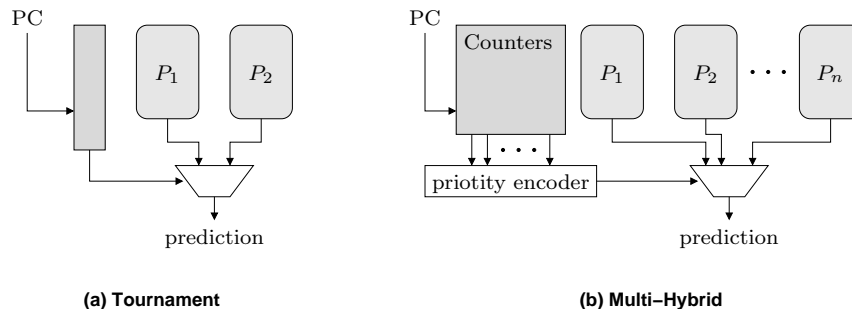


Fig. 1. The tournament predictor (a) chooses between only two experts while the multi-hybrid (b) selects from an arbitrarily sized ensemble.

The multi-hybrid predictor is a generalization of the two-expert tournament scheme [10]. The multi-hybrid is not limited to two experts, but the selection process still ignores the predictions of the experts. For each branch, a small (2-bit) counter is associated with each expert. The experts that are correct more often have higher counter values. The multi-hybrid simply selects the expert with the highest counter value. The predictor is shown schematically in Figure 1b.

3 Methodology

We chose several branch predictor configurations using the best algorithms from the state of the art. From these predictors, we formed different sized ensembles based on the total required storage. Analyzing branch predictors at different hardware budgets is a common approach since the storage needed is roughly proportional to the processor chip area consumed. The branch prediction algorithms used for our experts are the global branch history gskewed [22], Bi-Mode [17], YAGS [8] and perceptron [14], and per-branch history versions of the gskewed (called pskewed) and the loop predictor [10]. Out of all of the branch prediction algorithms currently published in the literature, the perceptron is the best performing algorithm that we examined. For the master algorithms studied in this paper, the sets of predictors used are listed in Table 1.

We simulated each master algorithm with the ensembles of experts listed in Table 1. The notation X_y denotes the composite algorithm of master algorithm X applied to ensemble y . For example, multi-hybrid $_\gamma$ denotes the multi-hybrid algorithm applied to ensemble γ .

Ensemble Name	Hardware Budget	Number of Experts	Actual Size (KB)	Experts in the Ensemble
α	8KB	3	6.4	PC(7,30) YG(11,10) LP(8)
β	16KB	5	14.4	BM(13,13) PC(7,30) YG(11,10) PS(10,11,4) LP(8)
γ	32KB	6	29.9	BM(14,14) GS(12,12) PC(7,30) YG(11,10) PS(11,13,8) LP(9)
δ	64KB	6	62.4	BM(13,13) GS(16,16) PC(7,30) YG(11,10) PS(10,11,4) LP(8)
η	128KB	6	118.9	BM(13,13) GS(16,16) PC(7,30) YG(11,10) PS(13,16,10) LP(9)

Expert Name	Abbreviation	Parameters
Bi-Mode	BM	$\log_2(\text{counter table entries})$, history length
gskewed	GS	$\log_2(\text{counter table entries})$, history length
YAGS	YG	$\log_2(\text{counter table entries})$, history length
perceptron	PC	$\log_2(\text{number of perceptrons})$, history length
pskewed	PS	$\log_2(\text{pattern table entries})$, $\log_2(\text{counter table entries})$, history length
loop	LP	counter width

Table 1. The sets of experts used in our experiments for varying hardware budgets as measured by bytes of storage necessary to implement the branch predictors. Descriptions of the parameters for the listed in the lower table.

The simulation is based on the SimpleScalar 3.0 processor simulation toolset [1], [4] for the Alpha AXP instruction set architecture. The benchmarks used are the integer applications from the SPEC2000 benchmark suite [23], using primarily the training input sets. The binaries were compiled on an Alpha 21264 system, using the cc compiler. The first one billion instructions of each benchmark were simulated. The conditional branch misprediction rate averaged over all of the benchmarks is the metric used to compare the performance of the different master algorithms.

4 Motivation

In this section, we analyze the performance of the multi-hybrid master algorithm and motivate the use of a “smarter” approach using results from machine learning research. Selection based master algorithms compute an index $idx \in [1, n]$, and then the master algorithm’s final prediction equals the prediction of E_{idx} . Note that the computation of idx ignores the current predictions of the experts.

We collected statistics about the branches predicted by the multi-hybrid master algorithm. We classified each predicted branch based on the number of experts that were correct, and the correctness of the overall prediction of the multi-hybrid. The percentage of all branches predicted for the gcc benchmark that fall into each class is tabulated in Table 2. The trends are similar for the other benchmarks.

The first two classes represent those cases where a selection based master algorithm has no impact on the final prediction. If all of the experts agree on the same prediction, it does not matter which expert the master algorithm chooses. The next class consists of the branches that were predicted correctly by the multi-hybrid selector when some decision had to be made. The remaining classes are situations where one or more experts disagreed with the others, and the master algorithm failed to choose an expert that made a correct prediction. These are the most interesting cases because they represent opportunities where we can apply better algorithms to improve the overall prediction rate. A key observation is that if a master algorithm was to look at the predictions made by the experts, and simply sided with the majority, then an additional 0.82%-1.52% of all branches could be predicted correctly, which translates into a 20%-30% reduction in branch mispredictions. The primary conclusion that we draw from these statistics is that there is indeed useful information in the actual predictions of the experts that can and should be used by the master algorithm.

Ensemble	α	β	γ	δ	η
Experts	3	5	6	6	6
% All Wrong	1.49	0.78	0.60	0.62	0.53
% Correct (no choice)	45.14	41.64	40.98	41.35	41.83
% Correct (w/ choice)	49.16	53.47	54.58	54.80	54.50
% Overall Wrong When:					
1 Expert Correct	2.78	1.32	1.01	0.94	0.86
2 Experts Correct	1.44	1.26	0.94	0.80	0.77
3 Experts Correct		1.05	0.83	0.68	0.66
4 Experts Correct		0.47	0.72	0.56	0.58
5 Experts Correct			0.34	0.26	0.28

Table 2. Statistics collected for the multi-hybrid algorithm on `gcc`. Branches are classified depending on the correctness of the selected expert, and the total number of correct experts.

5 Weighted Majority Branch Predictors

The statistics presented in Section 4 motivate the exploration of better master algorithms for branch prediction, particularly algorithms that take into account the current predictions of the ensemble of experts. The algorithm that we choose to use in this study is the Weighted Majority algorithm, and in particular the variant that Littlestone and Warmuth call WML in [20].

The WML algorithm assigns each of the n experts E_1, E_2, \dots, E_n positive real-valued weights w_1, w_2, \dots, w_n , where all weights are initially the same. During the prediction phase of each trial, the algorithm computes q_0 , which is the sum of the weights that correspond to experts that predict 0, and q_1 , which is the sum of the weights for experts predicting 1. If $q_0 < q_1$, then the Weighted Majority

algorithm returns a final prediction of 1. That is, the algorithm predicts with the group that has the larger total weight. During the update phase of the trial, the weights corresponding to experts that mispredicted are multiplicatively reduced by a factor of $\beta \in (0, 1)$ unless the weight is already less than γ/n times the sum of all of the weights at the start of the trial, where $\gamma \in [0, \frac{1}{2})$. The multiplication by β reduces the relative influence of poorly performing experts for future predictions. The parameter γ acts as an “update threshold” which prevents an expert’s influence from being reduced too much, so that the expert may more rapidly regain influence when it starts performing well. This property is useful in situations where there is a “shifting target” where some subset of experts perform well for some sequence of predictions, and then another subset of experts become the best performing of the group. This is particularly applicable in the branch prediction domain where branch behavior may vary as the program enters different phases of execution.

To apply the Weighted Majority algorithm to branch prediction, we maintain a table of weights indexed by the branch address (the *instance*). The weights corresponding to the branch in question are used to compute the weighted sums q_0 and q_1 which determine the final prediction. This allows different instances to use different sets of weights, since the best subset of experts for one branch may be different than the best subset for another. Throughout the rest of this section, the tables used in the simulations have 2048 sets of weights.

We simulated the Weighted Majority algorithm using the same ensembles used in Section 4 for the multi-hybrid algorithm. Out of the combinations of β and γ that we simulated, the best values are $\beta = 0.85$ and $\gamma = 0.499$. Since $\gamma = 0.499$ is very close to the maximum allowable value of $\frac{1}{2}$, we conclude that for the branch prediction domain the best set of experts shifts relatively frequently.

As presented, the WML algorithm can not be easily and efficiently implemented in hardware. The primary reason is that the weights would require large floating point formats to represent, and the operations of adding and comparing the weights can not be performed within a single processor clock cycle. As described, the WML algorithm decreases the weights monotonically, but the performance is the same if the weights are all renormalized after each trial. Renormalization prevents underflow in a floating point representation. Furthermore, the storage needed to keep track of all of the weights is prohibitive. Nevertheless, the performance of the WML algorithm is interesting as it provides performance numbers for an ideal case. Figure 2 shows the branch misprediction rates of the WML algorithm (for $\beta = 0.85, \gamma = 0.499$) compared against the multi-hybrid. The best branch prediction scheme consisting of a single expert (the perceptron predictor) is also included for comparison. Compared to the single predictor, WML_η makes 10.9% fewer mispredictions, and 5.4% fewer than $multi_hybrid_\eta$. A 5-11% improvement in the misprediction rate is non-trivial in the branch prediction field where the average misprediction rates are already less than 5%. These results are encouraging and motivate the exploration of hardware unintensive implementations.

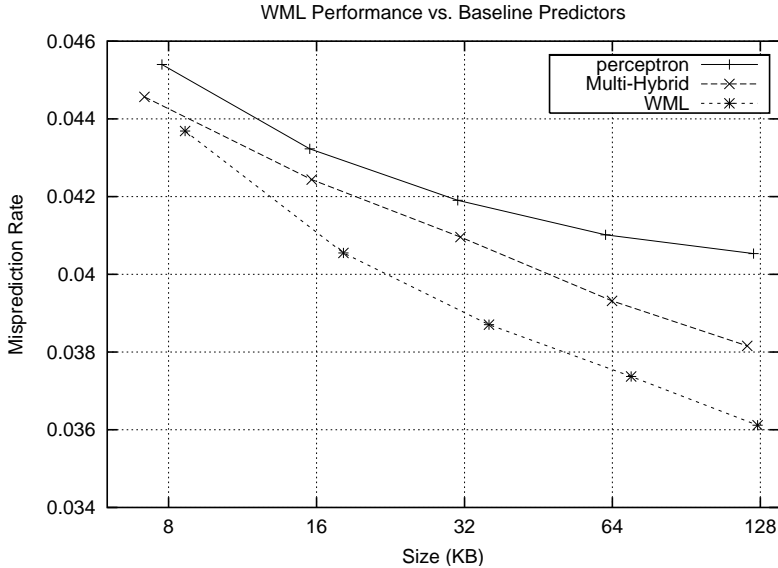


Fig. 2. The WML master algorithm yields lower misprediction rates than the selection based multi-hybrid master algorithm and the best singleton algorithm.

There are several aspects of the WML algorithm that need to be addressed: the representation of weights, updating the weights, the γ update threshold, and normalization. In place of floating point weights, k -bit integers can be used instead. Multiplication is a costly operation to perform in hardware. We therefore replace the multiplication by β with additive updates. This changes the theoretical mistake bounds of the algorithm. We justify this modification because in our application domain, the size of the ensemble of experts is relatively small and so this change in the asymptotic mistake bounds should not greatly affect prediction accuracy. The limited range of values that can be represented by a k -bit integer plays a role similar to the update threshold imposed by γ because a weight can never be decreased beyond the range of allowable values. Instead of normalizing weights at the end of each trial, we use update rules that increment the weights of correct experts when the overall prediction was wrong, and decrement the weights of mispredicting experts when the overall prediction was correct. We call this modified version of the Weighted Majority algorithm aWM (for approximated Weighted Majority).

Figure 3 shows the performance of the aWM algorithm compared to the ideal case of WML. Because the smaller ensembles have smaller hardware budgets, the amount of additional resources that may be dedicated to the weights is limited, and therefore the size of the weights used are smaller. For ensembles α, β and γ , the sizes of the weights are 2, 3 and 4 bits, respectively. Ensembles δ and η each use 5-bit weights. The most interesting observation is how well

the aWM algorithm performs when compared to the ideal case of WML despite the numerous simplifications of the algorithm that were necessary to allow an efficient hardware implementation.

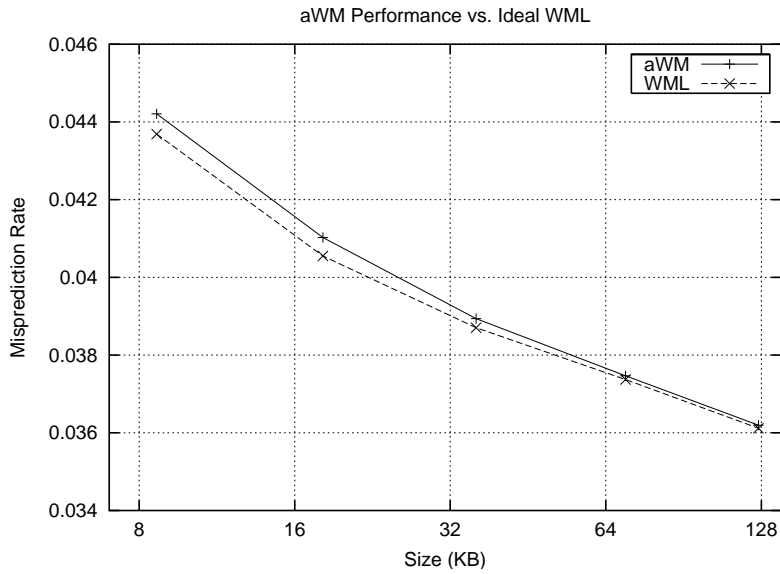


Fig. 3. Despite several simplifications made to adapt the WML algorithm for efficient hardware implementation, the modified version aWM still performs very well when compared to the ideal case of WML.

The Weighted Majority approach to combining the advice from multiple experts has an advantage over selection based approaches. Consider a situation where the expert that has been performing the best over the last several trials suddenly makes an incorrect prediction. In a selection based approach, the master algorithm simply chooses based on the past performance of the experts and will make a misprediction in this situation. On the other hand, it is possible that when the best expert mispredicts, there may be enough experts that predict in the opposite direction such that their collective weight is greater than the mispredicting experts. For the aWM_η predictor executing the gcc benchmark, situations where the weighted majority is correct and the expert with the greatest weight is incorrect occur over 80% more frequently than cases where the best expert is correct and the weighted majority is wrong. This confirms our earlier hypothesis that the information conveyed by the collection of all of the experts' predictions is indeed valuable for the accurate prediction of branches.

The Weighted Majority algorithm can also be useful for the problem of determining *branch confidence*, which is a measure of how likely a prediction will

be correct [13]. For aWM_η on `gcc`, when q_0 and q_1 are within 10% of being equal, the overall accuracy of the predictor is only 54%. This means that when we do not have a “strong” majority, the confidence in the prediction should not be very great.

6 Conclusions and Future Work

The problem of correctly predicting the outcome of conditional branches is a critical issue for the continued improvement of microprocessor performance. Algorithms from the machine learning community applied to the branch prediction problem allow computer architects to design and implement more accurate predictors. Despite the simplification of the Weighted Majority algorithm into a form that amenable to implementation in a processor, the hardware unintensive variant achieves misprediction rates within 0.2-1.2% of an idealized version. The results show the importance of combining the information provided by *all* of the experts.

The Weighted Majority algorithm is but one possible approach to combining the predictions from an ensemble of experts. Future work includes researching more accurate algorithms and techniques which yield even simpler implementations. The research described in this paper may be applied to similar areas in the design of microprocessors. Prediction and speculation are pervasive in the computer architecture literature, for example memory dependence prediction [6], [16], value prediction [18], and cache miss prediction [2]. Any of these areas could potentially benefit from applications of more theoretically grounded machine learning algorithms.

Acknowledgements

This research was supported by NSF CAREER Grant 9702281 (Henry).

References

1. Todd M. Austin. SimpleScalar Hacker’s Guide (for toolset release 2.0). Technical report, SimpleScalar LLC. http://www.simplescalar.com/docs/hack_guide_v2.pdf.
2. James E. Bennett and Michael J. Flynn. Prediction Caches for Superscalar Processors. In *Proceedings of the 30th International Symposium on Microarchitecture*, pages 81–90, Research Triangle Park, NC, USA, 1997.
3. Avrim Blum. Empirical Support for Winnow and Weight-Majority Based Algorithms: Results on a Calendar Scheduling Domain. In *Proceedings of the 12th International Conference on Machine Learning*, pages 64–72, Tahoe City, CA, USA, 1995.
4. Doug Burger and Todd M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report 1342, University of Wisconsin, June 1997.

5. Brad Calder, Dirk Grunwalk, Michael Jones, Donald Lindsay, James Martin, Michael Mozer, and Benjamin Zorn. Evidence-Based Static Branch Prediction Using Machine Learning. *ACM Transactions on Programming Languages and Systems*, 19(1):188–222, 1997.
6. George Z. Chrysos and Joel S. Emer. Memory Dependence Prediction Using Store Sets. In *Proceedings of the 25th International Symposium on Computer Architecture*, pages 142–153, Barcelona, Spain, June 1998.
7. Adam A. Deaton and Rocco A. Servedio. Gene Structure Prediction From Many Attributes. *Journal of Computational Biology*, 2001.
8. Avinoam N. Eden and Trevor N. Mudge. The YAGS Branch Prediction Scheme. In *Proceedings of the 31st International Symposium on Microarchitecture*, pages 69–77, Dallas, TX, USA, December 1998.
9. Joel Emer and Nikolas Gloy. A Language for Describing Predictors and its Application to Automatic Synthesis. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 304–314, Boulder, CO, USA, June 1997.
10. Marius Evers, Po-Yung Chang, and Yale N. Patt. Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches. In *Proceedings of the 23rd International Symposium on Computer Architecture*, pages 3–11, Philadelphia, PA, USA, May 1996.
11. Andrew R. Golding and Dan Roth. Applying Winnow to Context-Sensitive Spelling Correction. In *Proceedings of the 13th International Conference on Machine Learning*, pages 182–190, Bari, Italy, July 1996.
12. Mark Herbster and Manfred Warmuth. Tracking the Best Expert. *Machine Learning*, 32(2):151–178, August 1999.
13. Erik Jacobson, Eric Rotenberg, and James E. Smith. Assigning Confidence to Conditional Branch Predictions. In *Proceedings of the 29th International Symposium on Microarchitecture*, pages 142–152, Paris, France, December 1996.
14. Daniel A. Jiménez and Calvin Lin. Dynamic Branch Prediction with Perceptrons. In *Proceedings of the 7th International Symposium on High Performance Computer Architecture*, pages 197–206, Monterrey, Mexico, January 2001.
15. Daniel A. Jiménez and Calvin Lin. Perceptron Learning for Predicting the Behavior of Conditional Branches. In *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, Washington, DC, USA, July 2001.
16. R. E. Kessler. The Alpha 21264 Microprocessor. *IEEE Micro Magazine*, 19(2):24–36, March–April 1999.
17. Chih-Chieh Lee, I-Cheng K. Chen, and Trevor N. Mudge. The Bi-Mode Branch Predictor. In *Proceedings of the 30th International Symposium on Microarchitecture*, pages 4–13, Research Triangle Park, NC, USA, December 1997.
18. Mikko H. Lipasti, Christopher B. Wilkerson, and John Paul Shen. Value Locality and Load Value Prediction. In *Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 138–147, Cambridge, MA, USA, October 1996.
19. Nick Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm. *Machine Learning*, 2:285–318, 1988.
20. Nick Littlestone and Manfred K. Warmuth. The Weighted Majority Algorithm. *Information and Computation*, 108:212–261, 1994.
21. Scott McFarling. Combining Branch Predictors. TN 36, Compaq Computer Corporation Western Research Laboratory, June 1993.
22. Pierre Michaud, Andre Sez nec, and Richard Uhlig. Trading Conflict and Capacity Aliasing in Conditional Branch Predictors. In *Proceedings of the 24th International*

Symposium on Computer Architecture, pages 292–303, Boulder, CO, USA, June 1997.

23. The Standard Performance Evaluation Corporation. WWW Site. <http://www.spec.org>.
24. Volodya Vovk. Universal Forecasting Strategies. *Information and Computation*, 96:245–277, 1992.