Published in "Romanian Journal of Information Science and Technology", vol.3, No.3, pg.287-301, ISSN: 1453-8245, 2000, Romanian Academy, Bucharest, Romania

TOWARDS A POWERFUL DYNAMIC BRANCH PREDICTOR

Lucian N. VINTAN
University "L. Blaga", Dept. of Comp. Sc., Sibiu, ROMANIA, E-mail: *vintan@jupiter.sibiu.ro*

Abstract

Dynamic branch prediction in high-performance processors is a specific instance of a general Time Series Prediction problem that occurs in many areas of science. In contrast, most current branch prediction research focuses on Two-Level Adaptive Branch Prediction techniques, a very specific solution to the branch prediction problem. An alternative approach is to look to other application areas and fields for novel solutions to the problem. In this paper we examine the application of neural networks to dynamic branch prediction. Two neural networks are considered, a Learning Vector Quantisation (LVQ) Network and a Multi-Layer Perceptron Network accomplished by the Backpropagation learning algorithm. We demonstrate that a neural predictor can achieve prediction rates better than conventional two-level adaptive predictors and therefore suggest that neural predictors are a suitable vehicle for future branch prediction research.

1. Introduction

As the average instruction issue rate and depth of the pipeline in multiple-instruction-issue (MII) processors increase, accurate dynamic branch prediction becomes more and more essential. Very high prediction accuracy is required because an increasing number of instructions are lost before a branch misprediction can be corrected. As a result even a misprediction rate of a few percent involves a substantial performance loss.

If branch prediction is to improve performance, branches must be detected within the dynamic instruction stream, and both the direction taken by each branch and the branch target address must be correctly predicted. Furthermore, all of the above must be completed in time to fetch instructions from the branch target address without interrupting the flow of new instructions to the processor pipeline. A classic Branch Target Cache (BTC) [Hen96] achieves these objectives by holding the following information for previously executed branches: the address of the branch instruction, the branch target address and information on the previous outcomes of the branch. Branches are then predicted by using the PC address to access the BTC in parallel with the normal instruction fetch process. As a result each branch is predicted while the branch instruction itself is being fetched from the instruction cache. Whenever a branch is detected and predicted as taken, the appropriate branch target is then available at the end of the instruction fetch cycle, and instructions can be fetched from the branch target in the cycle immediately after the branch itself is fetched. Straightforward prediction mechanisms based on the previous history of each branch give a prediction accuracy of around 80 to 95% [Hen96]. This success rate proved adequate for scalar processors, but is generally regarded as inadequate for MII architectures.

The requirement for higher branch prediction accuracy in MII systems and the availability of additional silicon area led to a dramatic breakthrough in the early 90s with branch prediction success rates as high as 97% [Yeh92] being reported. These high success rates were obtained using a new set of prediction techniques known collectively as Two-Level Adaptive Branch Prediction that were developed independently by Yale Patt's group at the University of Michigan [Yeh91] and by Pan, So and Rahmeh from IBM and the University of Texas [Pan92]. Two-Level Adaptive Branch Prediction uses two levels of branch history information to make a branch prediction. The first level consists of a History Register (HR) that records the outcome of the last k branches encountered. The HR may be a single global register, HRg, that records the outcome of last k branches executed in the dynamic instruction stream or one of multiple local history registers, HRl, that record the last k outcomes of each branch. The second level of the predictor known as the Pattern History Table (PHT) records the behaviour of a branch during previous occurrences of the first level

predictor. It consists of an array of two-bit saturating counters, one for each possible entry in the HR. 2^k entries are therefore required if a global PHT is provided, or many times this number if a separate HR and therefore PHT is provided for each branch.

Although a single term is usually applied to the new predictors, this is misleading. Since the first level predictor can record either global or local branch history information, two distinct prediction techniques have in fact been developed. The global method exploits correlation between the outcome of a branch and the outcome of neighbouring branches that are executed immediately prior to the branch. In contrast, the local method depends on the observation that the outcome of a specific instance of a branch is determined not simply by the past history of the branch, but also by the previous outcomes of the branch when a particular branch history was observed.

In complete contrast to earlier work, this paper explores the possibility of using neural networks to predict branch outcomes. Conventional two-level branch predictors rely completely on one of only two correlation mechanisms. One of our main research objectives is therefore to use neural networks to identify new correlations that can be exploited by branch predictors. We also wish to determine whether more accurate branch prediction is possible and to gain a greater understanding of the underlying prediction mechanisms. Finally, we hope to design and evaluate hardware implementations of simplified neural branch predictors. Alternatively, the results of our research may be useful in designing more sophisticated two-level branch predictors.

In this paper we explore the suitability of two neural networks, a Learning Vector Quantisation Network (LVQ) and a Multi-Layer Perceptron (MLP), for branch prediction. Through trace driven simulation we demonstrate that neural predictors can achieve success rates that are better than those obtained by conventional two-level adaptive predictors. We therefore suggest that neural networks are indeed a suitable vehicle for future branch prediction research.

2. Previous Work

Since most previous research on branch prediction has concentrated on two-level adaptive techniques [Cha95, Eve95, Lee97, McF93, Pan92; Sec95, Spr97, Yeh91, 92a, 92b, 93,], it is useful to explore some of the drawbacks of two-level predictors. The main disadvantages can be found in the following areas:

- Size of PHT
- Branch interference
- Slow Initialisation
- "Hard-to-Predict" Branches

With a global predictor using a single Pattern History table (PHT), a total of 2^n entries are required where n is the number of bits in HRg. This is the GAg predictor in Patt's classification system [Yeh92a]. To achieve a 3% misprediction rate with the Spec benchmarks an 18-bit HRg is required [Yeh92a] giving a PHT with 2^{18} or 256K entries. Moreover, if a separate PHT is used for each branch, as in Patt's GAp configuration, a far greater area must be devoted to the PHT. For example if HRl is 12 bits long and 16 bits are required to distinguish each branch, a total of 2^{28} entries is required. Similar high storage requirements apply to local branch predictors - PAg and PAp in Patt's system - even though fewer local history register bits are generally required to achieve a given success rate.

The high cost of the PHT suggests that alternative configurations should be considered. One possibility is to replace the complete PHT with a conventional cache structure [Chen96, Vin99b]. Although a tag field must be added to each PHT entry, the very large size of conventional PHTs suggests that the total number of entries and therefore the total cost of the PHT could be significantly reduced. Furthermore, the impact of PHT cache misses can be minimised by adding a default prediction counter field to the conventional BTC which must still be included in all two-level predictors to furnish a target address for each branch [Ste99]. Alternatively, an entirely different approach, such as the neural branch predictors presented in this paper, can be investigated.

The second problem, branch interference, is a direct result of excessive PHT sizes. Because of the high cost of PHTs, researchers are reluctant to provide a separate PHT for each branch. Instead each PHT is usually shared between groups or sets of branches, giving rise to the GAs and PAs configurations in Patt's classification. Unfortunately, prediction information is now shared between multiple branches, leading to interference between different branches. We note that this problem can also be avoided by caching the PHT information.

The third problem is PHT initialisation. In the worst case, 2^n prediction counters, where n is the length of the HR, must be initialised for each branch before the predictor is fully effective. Even allowing for the fact that a PHT is effectively a sparse matrix with many unused entries, this situation contrasts sharply with a classic BTC which is fully initialised after one execution of each branch. This problem can be avoided by combining explicit PHT counter initialisation with the simple default predictor in the BTC mentioned earlier [Ste99].

Finally, some branches remain stubbornly hard-to-predict [Mud96, Vin99b]. There are two cases. The outcome of some data dependent branches is effectively random and these branches will never be accurately predicted. However, it should be possible to predict certain branches that are currently hard-to-predict more accurately by identifying new correlation mechanisms and adding them to the prediction process. We believe that neural predictors are an excellent vehicle for investigating potential new correlation mechanisms.

We have emphasised earlier that most branch prediction research is based on two closely related correlation mechanisms. Yet branch prediction is a specific example of a far more general time series prediction problem that occurs in many diverse fields of science. It is therefore surprising that there has not been more cross-fertilisation of ideas between different application areas. A notable exception is a paper by Mudge et al [Mud96] which demonstrates that all two-level adaptive predictors implement special cases of the Prediction by Partial Matching [PPM] algorithm that is widely used in data compression. Mudge uses the PPM algorithm to compute a theoretical upper bound on the accuracy of branch prediction, while Steven et al [Ste99] demonstrate how a two-level predictor can be extended to implement the PPM algorithm with a resultant reduction in the misprediction rate. Time series prediction is also an important research topic in neural networks. It therefore appears natural to look to neural networks for a further cross-fertilisation of ideas.

As far as we are aware, only one paper by Calder et al [Cal95] has discussed the application of neural networks to the problem of branch prediction. Calder is concerned entirely with static branch prediction at compile time. His predictions are therefore based entirely on information about a program's structure that can be readily determined by a compiler. For example, a branch successor path that leads out of a loop or function is less likely to be followed than a path that remains within the loop or function. Using a neural network, Calder achieves a misprediction rate of only 20%, remarkably low for static branch prediction. Clearly, Calder was unable to feed the dynamic branch histories used by two-level predictors into his neural networks. Therefore, possibly the most useful contribution of his paper is to suggest a wide range of alternative inputs that might correlate with branch outcomes and which might therefore be usefully added to dynamic predictors.

In contrast, we apply neural networks to dynamic branch prediction. Since our objective is to demonstrate that neural networks can achieve better prediction accuracy than a conventional two-level adaptive predictor, we restrict our neural network inputs to using the same dynamic global HR information as a conventional two-level predictor.

3. Branch Prediction Models

In this paper we compare the performance of a GAp two-level adaptive predictor with a Multi-Level Perceptron (MLP). We also briefly consider the performance of a simple LVQ (Learning Vector Quantisation) neural predictor.

3.1 A Conventional GAp Predictor

The GAp predictor (Fig.1) uses an k bit shift register, HRg, to record the outcome of the last k branches executed. The per-address PHT is accessed by concatenating the PC address (considered on l bits) with HRg, the global history register. Each PHT entry consists of a tag field, the branch target address and only one prediction bit. To avoid placing unnecessary restrictions on the performance of our GAp predictor, the PHT is of unlimited size.

3.2 An LVQ Neural Predictor

The first neural network we examined was an LVQ (Learning Vector Quantisation) model (Fig. 2). Since LVQ [Gal93] is one of the simplest neural networks it is likely to be easier to implement in hardware than more complex neural nets. The neural predictor uses three input parameters: PC, HRg and HRl. Unusually, the local history register,

HRl, is a function of the global history register, HRg (Fig. 2). A distinct local history pattern is therefore accumulated for every global history path leading to a branch.

The LVQ predictor contains two "codebook" vectors: the first vector, Vt, is associated with the branch taken event and the second, Vnt, with the not taken event. Vt is initialised to all ones and Vnt to all zeros. During the prediction process, the three input parameters are concatenated to form a single input vector, X. Hamming distances are then computed between X and the two codebook vectors.

$$HD = \sum_{i} (Xi - Vi)^2$$

The vector with the smallest Hamming distance is defined as the winning vector, Vw, and is used to predict the branch. A win for Vt therefore indicates predict taken, while a win for Vnt indicates predict not taken.

When the branch outcome is determined, the codebook vector Vw that was used to make the prediction is adjusted as follows:

$$Vw(t+1) = Vw(t) +/- a(t)[X(t) - Vw(t)]$$

To reinforce correct predictions, the vector is incremented whenever a prediction was correct and decremented otherwise. The factor a(t) represents the learning factor and is usually set to a small constant less than 0.1. In contrast, the losing vector is unchanged. The neural predictor will therefore be trained continuously as each branch is encountered. It will also be adaptive since the codebook vectors always tend to reflect the outcome of the branches most recently encountered.

3.3 An MLP Neural Predictor

For our main neural predictor we used an MLP (Multilayer Perceptron) [Gal93] with a single intermediate layer (Fig. 3). To allow for a direct comparison with a conventional GAp predictor, the PC address and global history register, HRg, were concatenated to form the input vector. The MLP then produces a true output for predict taken and a false output for predict not taken. The training process used the well-known backpropagation algorithm. As it is usually known, backpropagation algorithm is dedicated for learning in feedforward networks using mean squared error (MSE) and gradient descent. It mainly consists in two steps: forward propagation step and respectively backward propagation step. The first step makes a bottom-up pass through the network to computed weighted sums and activations. The second step, starting with the outputs, makes a top-down pass through the output and intermediate cells computing gradients. Once we know the gradient we can take a small step to update the weights using a learning step. This process continues until the MSE become sufficiently small. After considerable experimentation we standardised on an intermediate layer which always has four more cells than the number of bits in the input vector.

4. Trace Driven Simulation Results

4.1 Simulation Environment

Our simulation work uses the Stanford integer benchmark suite, a collection of eight C programs designed to be representative of non-numeric code, while at the same time being compact. The benchmarks are computationally intensive with an average dynamic instruction count of 273,000. About 18% of the instructions are branches of which around 76% are taken. Some of the branches in these benchmarks are known to be particularly difficult to predict; see for example Mudges' detailed analysis [Mud96] of the branches in *quicksort*.

The benchmarks were compiled using a C compiler developed at the University of Hertfordshire, UK, for the HSA (Hatfield Superscalar Architecture) [Ste97]. Instruction traces were then obtained using the HSA instruction-level simulator, with each trace entry providing information on the branch address, branch type and target address. These traces were used to drive a series of trace-driven branch predictors developed at the University of Sibiu, RO. The trace-driven simulators are highly-configurable, the most important parameters being the number of HRg bits and the size of the PHT. As output the simulators generate the overall prediction accuracy, the number of incorrect target addresses and other useful statistics.

4.2 An LVQ Branch Predictor

We were initially attracted to LVQ networks by their simplicity. We wished to determine whether respectable branch prediction success rates could be delivered by a simple LVQ network that was dynamically trained after each branch prediction and used a minimal amount of branch history information [Vin99a].

The input vector for the neural network was constructed by concatenating the least significant bits of the PC with HRg and HRl (Fig. 2). Initially the values of the learning step a(t) were varied between 0.1 and 0.001. Eventually the value a(t) = 0.01 was standardised after it had been demonstrated that the predictor was largely insensitive to slight variations in a(t).

The LVQ predictor achieved success rates ranging from 88.3% to 89.5% (Fig. 4). These results are highly encouraging, particularly since the LVQ predictor is dynamically trained after each prediction and the predictions are based on only 5 to 13 bits of HR information. We also simulated an equivalent two-level adaptive predictor for comparative purposes (Fig. 4). The conventional predictor achieved marginally better results. Nonetheless the neural predictor outperformed the conventional predictor on some individual benchmarks and, on average, less than 1.5% separated the two predictors in all configurations. Since the input vector was not tailored in any way for the neural network, we again find these results very encouraging.

4.3 A Statically Trained MLP Predictor

Our first objective was to determine whether respectable prediction results could also be obtained using an MLP branch predictor [Vin99a]. We therefore initially trained the MLP predictor off line to gain a feel for its potential. Initially the MLP's weights were chosen in a randomly manner but all these initial weights are belonging to [-2/l+k, 2/l+k], according to [Gal93]. For calculating the output's value we used the classical sigmoidal activation function (squashing function) given by the well-known expression 1/ (1+exp(-x)).

Static training was achieved by generating a training vector set, consisting of PC-HRg pairs, for each benchmark. First the benchmark's instruction trace was pre-processed to determine how frequently individual branches were taken for specific values of HRg. The following fragment generated from the benchmark *sort* illustrates the output generated as a result of this analysis:

Branch PC	HRg	Taken	Not Taken	Taken(%)	Not Taken(%)
68	4055	121	41	74.69	25.31
68	3935	127	30	80.89	19.11
68	3453	17	138	10.97	89.03
68	1525	124	107	53.68	46.32
68	3925	109	143	43.25	56.75
68	1367	124	360	25.62	74.38
68	1373	210	234	47.30	52.70
68	765	4	3	57.14	42.86
68	3061	3	0	100.00	00.00
68	1399	72	200	26.47	73.53
68	1501	142	181	43.96	56.04
68	1909	126	196	39.13	60.87
68	3541	44	174	20.18	79.82
68	1213	4	1	80.00	20.00

As can be seen some of the branch outcomes are highly biased with branches being taken most of the time. In contrast, other branches are very hard to predict on the basis of the information given. For example, in the fourth line of the example, the branch is only taken 54% of the time. This input pair therefore provides virtually no useful training information. Only highly biased PC-HRg pairs, where the branch is either taken 70% of the time or not taken 70% of the time, were therefore included in the training set. The remaining PC-HRg pairs were discarded. For the above fragment, the following pairs were therefore retained for training:

Branch PC	HRg	Taken/Not Taken
68	4055	1
68	3935	1
68	3453	0
68	1367	0
68	3061	1
68	1399	0
68	3541	0
68	1213	1

Repeated training passes were then made using the training vectors until the mean squared error became smaller than 0.01. This training was performed separately for each benchmark. The statically-trained MLP predictor was then used in a final trace-driven prediction run. To improve the success rate, the predictor was also trained dynamically, by running traces. This training consisted of a single backpropagation learning step after each individual branch prediction.

The prediction results are summarised in Fig. 5 for values of HRg ranging from four to ten. For individual benchmarks the prediction accuracy ranges from 74% to 97% with an average accuracy of around 90%. Surprisingly, however, the average success rate rises only slowly from 89.0% to 89.5% as the number of bits in HRg is increased from four to ten. Table 1 presents the influence of the branches "polarisation degree" (PD) on the obtained prediction accuracies for three different HRg's lengths. As it can be observed it is optimal recommended to include in the static training set PC-HRg pairs that are taken / not taken ("polarised") about 60%-70% of time.

PD	60%	70%	80%	90%
K=4	89.44	89.55	89.46	89.41
K=6	89.91	89.80	89.81	89.73
K=8	89.83	89.91	89.83	89.82

Table 1. The influence of the branches "polarisation degree" on the prediction accuracy

4.4 A Dynamically Trained MLP Predictor

While the results obtained with a statically trained MLP predictor give a very encouraging indication of the potential of an MLP predictor, we are also interested in dynamically trained branch prediction, thus, in other words, without the benefit of a static learning process. The trace-driven simulations of the previous section were therefore repeated without any prior static training of the neural network. The entire training therefore consisted of the single backpropagation pass after each individual branch prediction. Detailed results for individual benchmarks are presented in Fig. 6 for sizes of HRg between four and ten. Average results ranged from 88.76% with HR = 4 to 89.1% with HR = 10. The prediction accuracy therefore growths slowly as the size of HRg is increased.

The benefit of our static training developed technique can be appreciated by comparing the performance of the statically and dynamically trained MLPs (Fig. 7). While dynamic training performs well for small sizes of HRg, performance growths slowly as the size of HR is increased beyond four, and the dynamically trained MLP is increasingly unable to match the success rate achieved by the statically trained predictor. At average, for an HRg on 10 bits, the statically trained MLP obtains a prediction accuracy of about 90% compared with the 89% prediction accuracy obtained by the (only) dynamically trained MLP.

4.5 MLP versus conventional GAP

Finally, we compared our MLP predictor with an equivalent GAp predictor. To avoid compromising the GAp predictor in any way the PHT table was of unlimited size having only one bit predictors (considering two bit predictors involves an unfair comparison between the GAp and MLP branch predictors due to the local history information used by the classical scheme). The prediction results for individual benchmarks are given Fig. 8. On average as the size of HRg was increased from four to ten the prediction accuracy also increased from 84.8% to 86%.

These results are compared with a statically trained MLP predictor in Fig. 9. Surprisingly, the MLP predictor is consistently more accurate than the GAp predictor by a margin of around 4%. Clearly a neural network is able to extract additional prediction information from a limited number of HRg bits. The situation is practically the same for a dynamically trained neural predictor, it also outperforms a GAp predictor as it can be seen comparing performances pointed out by figures 7 and 9.

5. Conclusions

In this paper we sought to determine whether a neural network could mimic a two-level adaptive branch predictor and achieve respectable branch prediction success rates. Two types of neural predictors were simulated, an LVQ predictor and an MLP predictor. While the LVQ predictor achieved results comparable to an equivalent conventional predictor, the statically trained MLP predictor and also the (only) dynamically trained MLP, outperformed its conventional counterpart. The dynamic MLP predictor was a bit less successful but still managed to outperform a conventional predictor. These results suggest that not only can neural networks generate respectable prediction results, but in some circumstances a neural predictor may be able to exploit correlation information more effectively than a conventional predictor. As a further work we intend to develop a new (static) training algorithm based on genetic algorithms (GA). We'll use GA in order to find an optimal set of network weights for our neural predictor. Thus, we hope obtaining better results with these obtained weights than with the randomly obtained weights. Also in the nearest future we'll investigate other NN architectures like some recurrent nets. These NN contain connections from the hidden cells to the context cells in order to store the state of the net on the previous step of time. Some recent research prove that some recurrent nets with asymmetric connections or partially recurrent nets, perform sequence recognition far better than other (simple recurrent) NN.

Clearly, therefore, neural networks are a useful vehicle for future branch prediction research. The challenge is to construct composite input vectors for neural network predictors that will enable them to outperform conventional predictors. This task involves both identifying new correlation mechanisms that can be exploited by neural prediction and tailoring the input information to fully exploit the capabilities of the neural predictor. Describing a neural network predictor that can be integrated into a high-performance superscalar processor is an even greater challenge (recent information on some efficient NN hardware implementations can be found at http://msia02.msi.se/~lindsey/nnwAtm.html or http://amesp02.tamu.edu/~sanchez/pubs/nnet.html). However, the creditable performance achieved by the simple LVQ neural predictor suggests that this approach should not be discounted. Anyway, and most important at this point of our "integrated" research, neural predictors could be a useful approach in establishing, estimating and understanding better, the processes of branch predictability. Also this concept could be used as a performance measurement of the predictability of branches and it is useful to compare its performance with the performance obtained through other prediction schemes. Thus a complex structure like a MLP together with its learning algorithm can serve as a diagnostic tool to measure some upper values of predictability, very important vehicle for computer architects.

Acknowledgements

This work was partially supported by the Romanian National Agency for Science, Technology and Innovation (ANSTI) grants MCT No. 4086/1999 and by the Romanian National Council of Academic Research grants CNCSU No. 489/1999 and CNCSIS No. 008/2000. Also our gratitude to Professor Gordon B. Steven from the University of Hertfordshire, UK, for providing HSA Stanford traces and for his useful concrete suggestions, guidance and encouragement related to our MII and branch prediction research.

References

[Cal95] **Calder, B., Grunwald, D. and Lindsay, D.** *Corpus-based Static Branch Prediction*, SIGPLAN notices, June 1995, pp79-92.

[Cha95] **Chang, P., Hao, E. and Patt, Y. N.** *Alternative Implementations of Hybrid Branch Predictors*, Micro-29, Ann Arbor, Michigan, November 1995, pp252-257.

[Chen96] **Chen, C. and King, C.** *Designing Dynamic Two-Level Branch Predictors Based on Pattern Locality*, EuroPar-96, Lyon, pp757-764.

[Eve98] **Evers, M., Patel, S. J., Chappell, R. S. and Patt, Y. N.** *An Analysis of Correlation and Predictability: What makes Two-Level Branch Predictors Work*, ISCA '25, Barcelona, Spain, June 1998, pp52-61.

[Gal93] **Gallant S.I.** *Neural Networks and Expert Systems*, MIT Press. 1993.

[Hen96] **Hennessy, J. L. and Patterson, D. A.** *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 2nd edition, 1996.

[Lee97] **Lee, C. C., Chen, I. -C. K. and Mudge, T. N.** *The Bi-Mode Branch Predictor,* Micro-30, Research Triangle Park, North Carolina, December 1997, pp4-13.

[McF93] **McFarling, S.** *Combining Branch Predictors*, WRL Technical Note, TN36, DEC, June 1993.

[Mud96] **Mudge T.N., Chen I., Coffey J.** *Limits of Branch Prediction,* Technical Report, Electrical Engineering and Computer Science Department, The University of Michigan, Ann Arbor, Michigan, USA, January 1996.

[Pan92] **Pan, S. , So, K. and Rahmeh, J. T.** *Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation*, ASPLOS-V, Boston, October 1992, pp76-84.

[Sec95] **Sechrest, S., Lee, C. and Mudge, T.** *The Role of Adaptivity in Two-Level Branch Prediction,* Micro-29, Ann Arbor, Michigan, November 1995, pp264-269.

[Spr97] **Sprangle, E., Chappell, R. S. , Alsup, M. and Patt, Y. N.** *The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference*, ISCA'24, Denver, Colorado, June 1997, pp284-291.

[Stev97] **Steven G. B., Christianson D. B., Collins R,. Potter R. and Steven F. L.** *A Superscalar Architecture to Exploit Instruction Level Parallelism*, Microprocessors and Microsystems, Vol.20, No 7, March 1997, pp391-400.

[Ste99] **Steven, G. B., Egan, C, Quick, P. and Vintan, L.** *Reducing Cold Start Mispredictions in Two-level Adaptive Branch Predictors,* CSCS-12, Bucharest, May 1999, pp145-150.

[Vin99a] **Vintan L.** *Predicting Branches through Neural Networks: A LVQ and a MLP Approach*, University "L. Blaga" of Sibiu, Faculty of Engineering, Dept. of Comp. Sc., Technical Report, February 1999.

[Vin99b] **Vintan, L. N. and Egan, C.** *Extending Correlation in Branch Prediction Schemes*, International Euromicro'99 Conference, Milano, Italy, September 1999.

[Vin99c] **Vintan L., Iridon M.** *Towards a High Performance Neural Branch Predictor,* International Joint Conference on Neural Networks (IJCNN CD-ROM, ISBN 0-7803-5532-6), Washington DC, USA, 10-16 July, 1999

[Vin99d] **Egan C., Steven G., Vintan L.** *A Cost Effective Cached Correlated Two Level Adaptive Branch Predictor,* Eighteenth IASTED International Conference, AI '2000, February 14-17, Innsbruck, Austria, 2000

[Vin2000] **Vintan, L.** *Instruction Level Parallel Architectures* (in Romanian), ISBN 973-27-0734-8, Romanian Academy Publishing House, Bucharest, 2000

[VinL2000] **Vintan, L.** – *Developing a New Branch Prediction Scheme*, Eighteenth IASTED International Conference, AI '2000, February 14-17, Innsbruck, Austria, 2000

[Yeh91] **Yeh, T. and Patt Y. N.** *Two-Levels Adaptive Training Branch Prediction*, Micro-24, Albuquerque, New Mexico, November 1991, pp51-61.

[Yeh92a] **Yeh, T. and Patt Y.** *Alternative Implementations of Two-Level Adaptive Branch Prediction*, ISCA -19, Gold Coast, Australia, May 1992, pp124-134.

[Yeh92b] **Yeh, T. and Patt Y.** *A Comprehensive Instruction Fetch Mechanism for a Processor Supporting Speculative Execution*, Micro-25, Portland, Oregon, December 1992, pp129-139.

[Yeh93] **Yeh, T. and Patt Y. N.** *A Comparison of Dynamic Branch Predictors that Use Two Levels of Branch History*, ISCA-20, San Diego, May 1993, pp257-266.

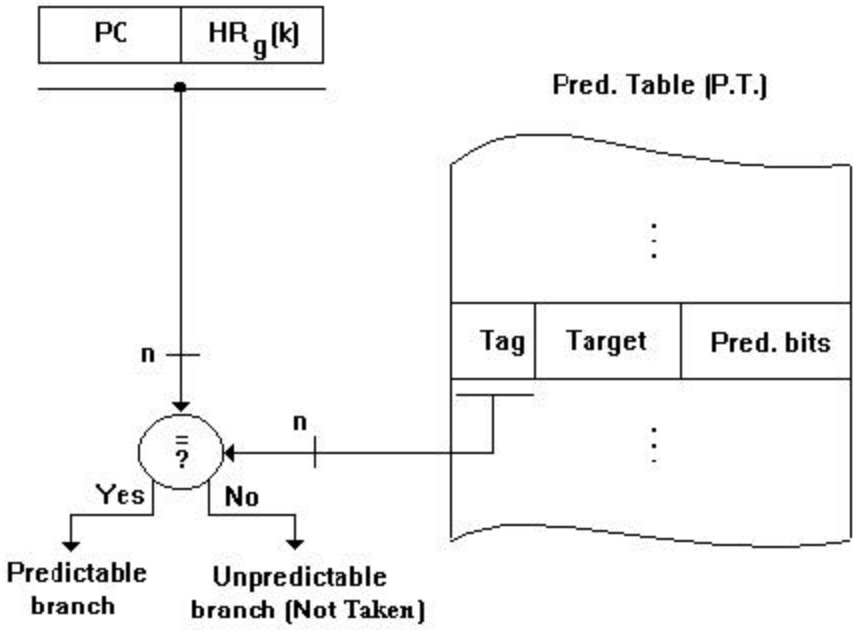

Figure 1. A Fully Associative GAp Scheme

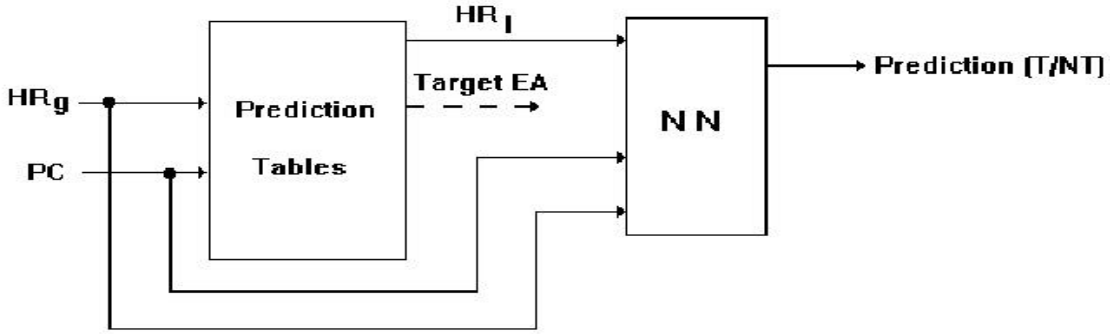

Figure 2. An LVQ Neural Branch Predictor

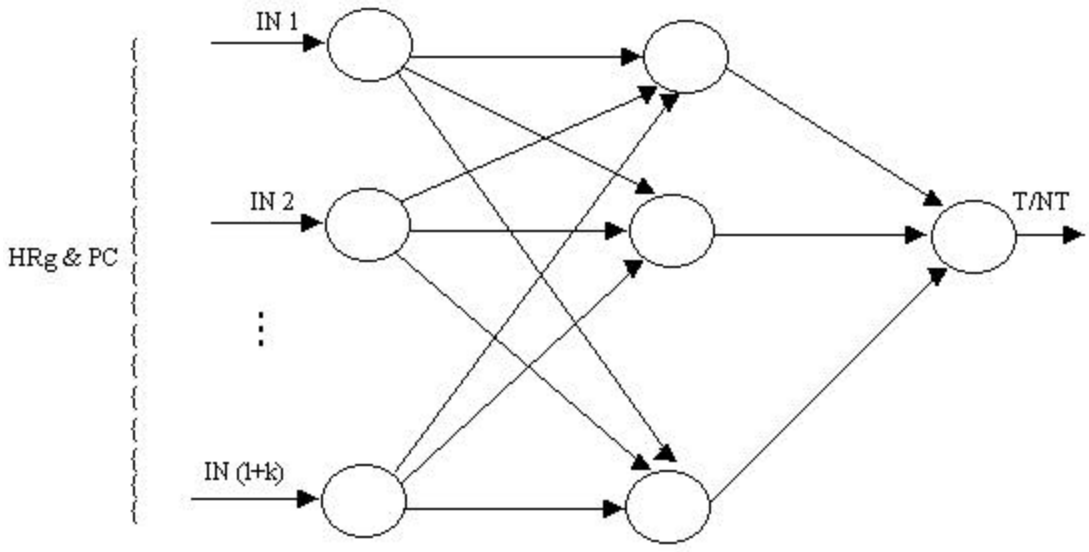

Figure 3. An MLP Neural Branch Predictor

PC	HRg	HRl	LVQ Predictor	Conventional Predictor
2	2	3	87.30	88.60
2	3	3	88.36	89.76
2	3	4	88.76	90.03
3	3	4	88.77	90.06
5	3	4	88.76	89.89
3	6	4	89.29	89.95
3	3	9	89.19	90.57
3	9	4	89.51	89.70

Figure 4. LVQ Predictor Accuracy

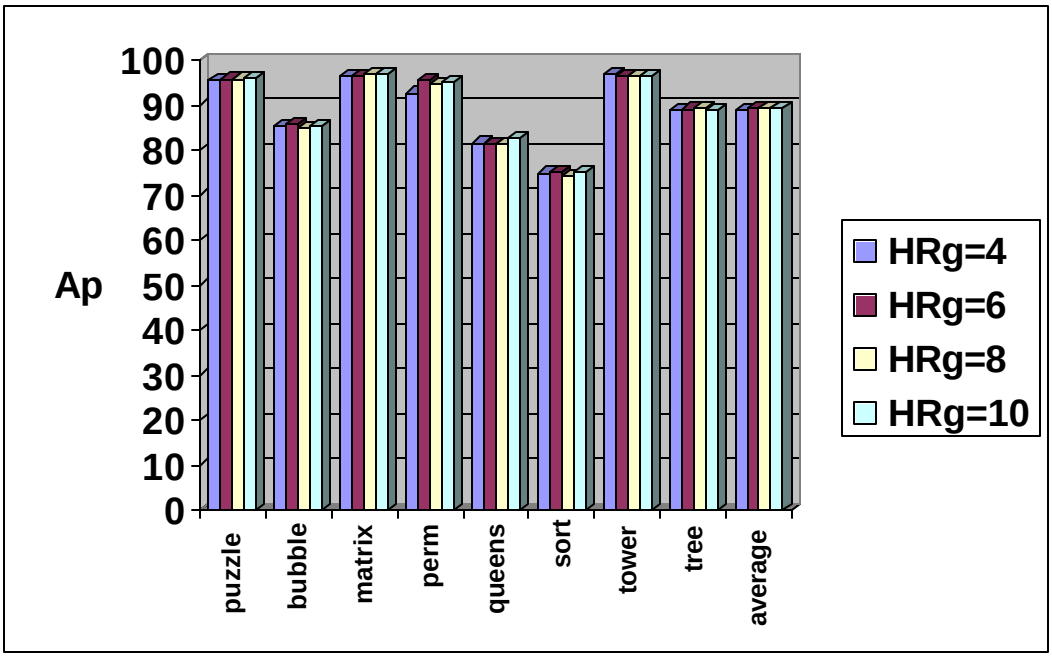

Figure 5. Prediction Accuracies for a Statically Trained MLP Predictor

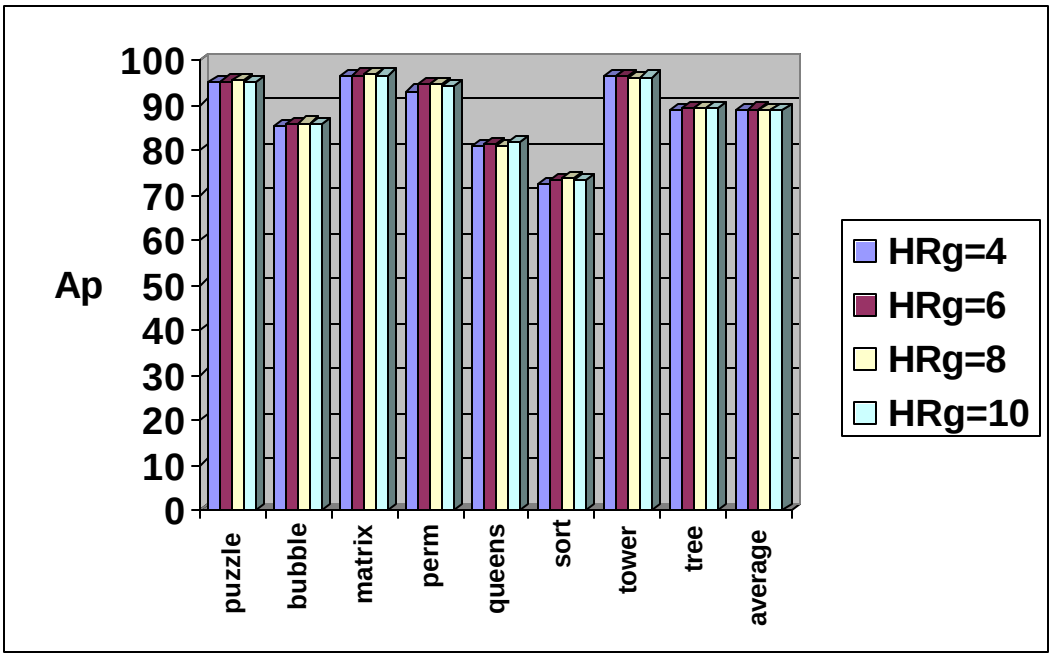

Figure 6. Prediction Accuracy for an Dynamically Trained MLP Predictor

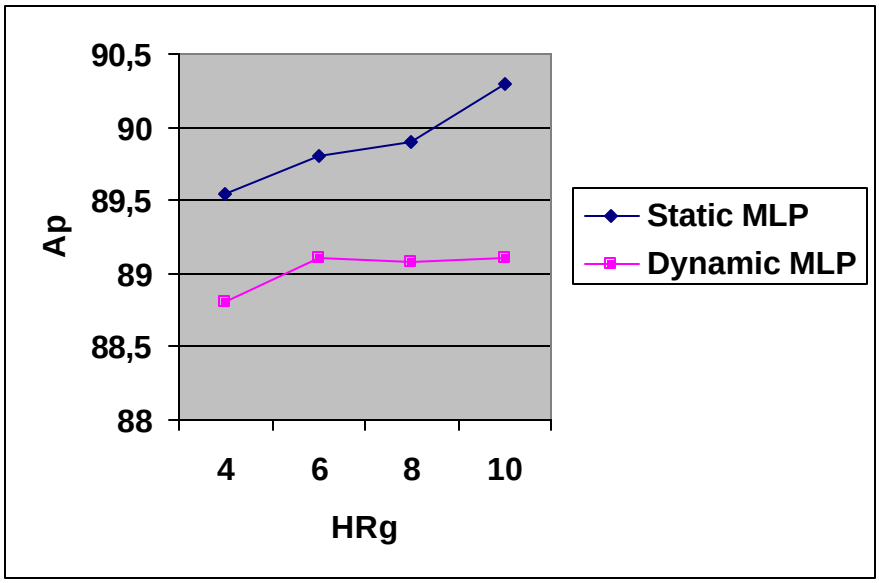

Figure 7. A Statically Trained MLP vs. a Dynamically Trained MLP

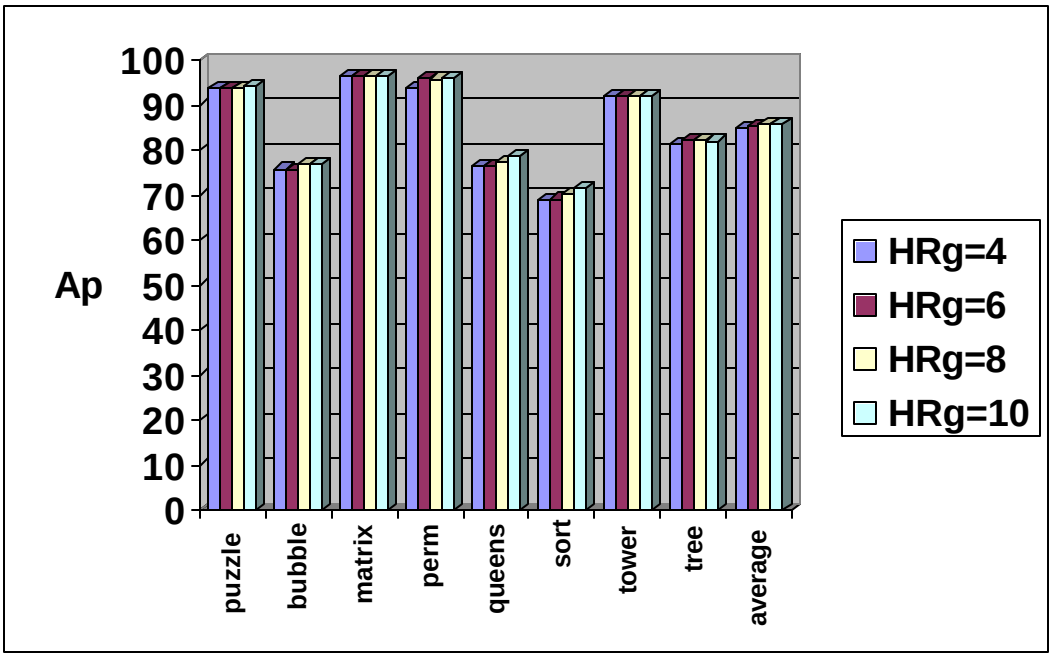

Figure 8. Prediction Accuracy for an Unlimited GAp Scheme

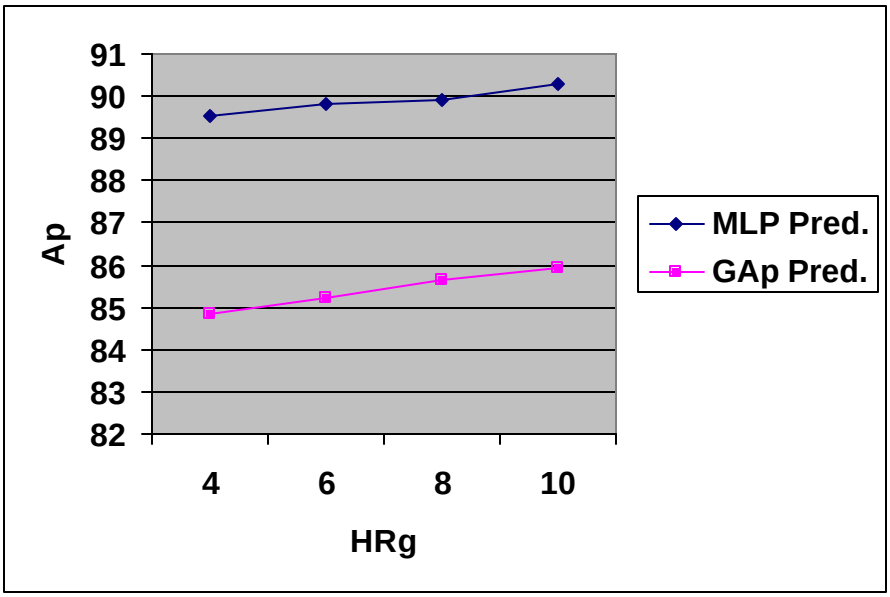

Figure 9.The Statically Trained MLP Predictor vs. the GAp Predictor