A NEW BRANCH PREDICTION APPROACH USING NEURAL NETWORKS

Lucian N. VINTAN, Adrian FLOREA,
"L. Blaga" University of Sibiu, Department of Computer Science, Str. E. Cioran, No. 4, Sibiu-2400, ROMANIA,
E-mail: vintan@vectra.sibiu.ro, aflorea@vectra.sibiu.ro

Abstract: The main aim of this short paper is to propose a new branch prediction approach called by us "neural branch prediction". We developed a first neural predictor model based on a simple neural learning algorithm, known as backpropagation algorithm, using a multilayer perceptron. Based on a trace driven simulation method we investigated the influences of the training processes. Also we compared the neural predictor with a powerful classical predictor and we establish that the neural predictor involves higher performances. Therefore, we conclude that in the nearest future it might be necessary to model and simulate other more powerful neural adaptive predictors, based on more complex neural networks architectures or even time series concepts, in order to obtain better prediction accuracies compared with the previous known classical schemes.

Key Words: MII Architectures, Branch Prediction, Trace Driven Simulation, Neural Algorithms, Backpropagation

1. INTRODUCTION

As the average instruction issue rate and depth of the pipeline in multiple instruction issue (MII) processors increase, the necessity of an efficient hardware branch predictor becomes essential. Very high prediction accuracies are necessary, because taking into account the MII processors characteristics as pipeline depth or issue rates, even a prediction miss rate of a few percent involves a substantial performance loss.

The main aim of this work is to propose a new branch prediction approach called neural branch prediction. Our work hypothesis will consider branch prediction as a particular problem belonging to pattern recognition class and therefore, we consider it is desirable to use neural networks in order to predict branches. Also, we investigate comparatively, through a trace driven simulation method, a classical branch prediction scheme proceeded from Professor Yale Patt's Research Group [Yeh92, Pan92, Cha97, Eve96] with some modifications and the proposed neural branch predictor, both of them integrated into a MII environment. We used the traces obtained based on the eight C Stanford integer benchmarks. These benchmarks were compiled through the HSA (Hatfield Superscalar Architecture) compiler, developed at the University of Hertfordshire, UK, by Dr. G.B. Steven's Research Group in Computer Architecture. Further, the traces were obtained using the HSA simulator, developed at the same university [Ste96]. Based on these tools, we have developed an original simulator to investigate comparatively some branch prediction schemes.

The first efficient approach in hardware branch prediction consists in Branch Target Buffer (BTB) structures [Per93]. BTB is a small associative memory, integrated on chip, that retains the addresses of recently executed branches, their targets and optionally other information (e.g. target opcode). Due to some intrinsic limitations, BTB's accuracies are limited on some benchmarks having unpropitious characteristics (e.g. correlated branches).

In order to improve BTB's efficiency, Yeh and Patt (1992) and independently Pan et al (1992) generalized it through a new approach called Two Level Adaptive Branch Prediction. According to [Yeh92], the Two Level Adaptive Branch Prediction uses two distinct levels of branch history information to make predictions. The first level consists in the History Register (HR), that contains the last k branches encountered (taken/ not taken) or the last k occurrences of the same branch instruction. The second level consists in the branch behavior of the last l occurrences of the specific pattern of these branches. It is implemented by a Pattern History Table (PHT), that contains essentially the branch prediction automaton (usually 2 bit saturating counters).

HR shifts left with a binary position when updated according to the actual branch behavior (taken=1/ not taken=0). There is a corresponding entry in the PHT for each of the 2^k HR's patterns. The prediction of the branch (P) is a function (f) of the actual prediction automaton state S_t.

$$\mathbf{P = f(S_t)} \qquad \mathbf{(1)}$$

After the branch is resolved, HR is shifted left and the prediction automaton state becomes S_{t+1}.

$$\mathbf{S_{t+1} = g(S_t ,B_t)} \qquad \mathbf{(2)}$$

where g represents the automaton's transition function and B_t represents the behavior of the last branch encountered (taken/ not taken). A lot of interesting implementations of these correlated branch prediction schemes are known [Yeh92, Pan92, Ega98, Rec98]. These Two Level Adaptive Branch Prediction schemes are very effective in predicting correlated branches with high accuracy. It's well known that the average prediction rate for these schemes, measured on nine of the ten Spec benchmarks, is about 97%, while BTB schemes achieved at most 94% on the same benchmarks

[Yeh92]. An interesting generalization of these Two Level Adaptive Branch Prediction schemes, based on the universal compression/prediction algorithm called "prediction by partial matching", is given in [Mud96].

Further, we'll try to propose a new distinct branch prediction approach, based on some pattern recognition concepts like neural networks, taking into account their more adaptive behavior in other similar problems. We suppose as a work hypothesis, that a neural network implemented here as a simple multilayer perceptron (MLP) together with the associated backpropagation learning algorithm, could be a better predictor - taking into account its adaptivity and "intelligence" - than a classical branch predictor. Thus, through this new approach, we'll look at branch prediction as a particularly pattern recognition problem.

2. A NEURAL MLP BRANCH PREDICTOR

The simulation work has been centered on the Stanford integer benchmark suite, a collection of eight C programs designed by Professor John Hennessy, to be representative of non - numeric code while at the same time being compact. The benchmarks are computationally intensive with higher dynamic instruction counts. The HSA gnu C compiler that targets the HSA instruction set compiled all these benchmarks. A dedicated HSA simulator [Ste96] that generates the corresponding traces simulated the resulted HSA object code. Some characteristics of the used traces are given in Table 1.

Benchmark	Total instr.	% Branches (%Taken)	Short description
Puzzle	804.620	25(91)	Solves a cube packing problem
Bubble	206.035	20(75)	Bubble sorts an array
Matrix	231.814	9(97)	Matrix multiplication
Permute	355.643	15(80)	Recursive computation of permutations
Queens	206.420	19(50)	Solves the eight queens problem
Sort	72.101	17(65)	Quick sorts a randomized array
Towers	251.149	15(76)	Solves Towers of Hanoi problem (recursive)
Tree	136.040	24(73)	Performs a binary tree sort

Table 1. Characteristics of the HSA traces

The average instructions number is about 273.000 and the average percentage of total instructions that are branches is about 18%, with about 76% of them being taken. Derived from HSA traces, special traces were obtained, containing exclusively all the processed branches. Each branch belonging to these modified HSA traces is stored in the following format: branch's type, the PC of the branch and it's target address.

As in the Two Level Adaptive Branch Prediction, in this case the run-time prediction process is based on the same three "orthogonal" information: the branch's PC low (PCl, on i bits), the history of the k previous branches named HRg (Global History Register on k bits) and the branch's own history (taken/not taken). Also, similarly to the Two Level Adaptive Branch Prediction, the considered simple MLP predictor will use some of these information (HRg, PC) in order to predict branches. In contrast, this time it's not necessary to implement the classical set of prediction automata stored into the Prediction Table. These automata will be replaced in our present approach with a single global MLP (neural) prediction structure, as it will be described further.

Therefore through this short paper we propose a new branch prediction approach based on neural networks. The predictor consists of a multilayer perceptron having one intermediate layer and using the well - known backpropagation learning algorithm. As it is 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. In this first MLP predictor approach, the considered inputs are the following: the branch global history information (HRg content, on k bits) and respectively the branch's PC, considered on l bits length. Based on a lot of laborious simulations we chose (l+k+4) intermediate cells belonging to the hidden layer as an "optimal" structure. The MLP produces a true ('1' – 0.9) output for a branch predicted as taken respectively a false ('0' – 0.1) output for a branch predicted as non taken. Figure 1 presents intuitively the new MLP branch prediction scheme.

Also we developed an original efficient learning method based on some laborious branch statistics. Thus, each benchmark trace was processed in order to obtain for each static branch and for each HRg pattern belonging to that branch, how many times the branch was taken respectively not taken. As an example, below it is presented a short fragment from a such laborious statistics *("bubble-sort"* trace, branch's PC = 68, for different obtained HRg patterns).

BR	HRg	T	NT	[%]T	[%]NT
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	0.00
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

Derived from these statistics we obtained the training input vector set (HRg & PC) respectively the desired output for each of these vectors (taken / not taken). More precisely, we considered that the desired output is 1/0 if the branch is respectively taken/not taken over 75%, for a certain HRg pattern (therefore, the branch's behaviour in a certain HRg context is strongly polarized on "taken" or "not taken"). If this condition isn't fulfiled for a given (HRg & PC) input vector we considered that vector inadequated for training and therefore it is ignored. Taking into account the statistics previous presented, the corresponding fragment of training input vectors set together with the desired (supervised) output are given below.

BR	T	T/NT
68	3935	1
68	3453	0
68	3061	1
68	3541	0
68	1213	1

The training process was repeated succesively for all the training vectors until each associated mean squared error becomes smaller than 0.01. After this special supervised training process, the MLP predictor will process the whole benchmark's trace. During the simulations we investigated comparatively through a trace driven simulation method, this new MLP branch predictor with a classical so called GAp scheme in Patt's taxonomy, firstly proposed in [Pan92], implemented by us as having an unlimited number of entries, with a structure like that presented in figure 2. Conceptually, a GAp scheme uses a separate Prediction Table for each branch. We called our Prediction Table - GAp, just to point out that it uses both HRg concatenated with PC information to make predictions. Taking into account its infinite capacity, our GAp scheme doesn't need any replacing algorithm (no interferences). The prediction automata associated with each branch are implemented using only one bit (taken / not taken).

Based on the adaptive heuristic nature of the used neural prediction algorithm, the input (PC & HRg) vectors will tend dynamically to a "taken" respectively "not taken" pattern (class). The predictor will learn continuously, even after the dedicated supervised learning process, while the benchmark's trace is processed, therefore the adequate class will "attract" with more and more accuracy the newly income vector, involving thus better predictions. During the first learning step (forward propagation) the prediction is done as a function of the obtained output value. The second step (backward propagation) is started after the branch's result is known (taken or not taken) and the weights are modified correspondingly with this certain behavior. Through this method, we approach the branch prediction problem as a pattern recognition problem, where the input pattern (HRg & PC) must be dynamically recognized belonging exclusively to one of two possible classes (Taken / Not Taken). Therefore, through this approach we established a possible link between branch prediction problems and respectively pattern recognition problems solved through neural networks. Of course, it's possible to extend these ideas to other neural branch predictors also having an unique input vector and as output the prediction itself (one bit)

Figure 3 presents prediction accuracies (Ap) for an infinite entries GAp scheme, considering different HRg's lengths. The average prediction accuracy growths from 84.83% (HRg on 4 bits) to 85.95% (HRg on 10 bits). Figures 4 and 5 shows the prediction accuracies for an untrained MLP branch predictor respectively for a trained MLP predictor, considering different lengths of HRg. In the first case the best average prediction accuracy is 88.47% obtained for a HRg on 4 bits in length and in the second case the best performance is 89.48%, obtained for a HRg on 10 bits (using more correlation bits the statical training process takes too long). Important, on "*quick-sort*" benchmark, the trained MLP predictor obtaines a prediction accuracy of 75.07% (for a HRg on 6 bits). This is quite significant because it is well – known that "*quick-sort*" benchmark is very unpredictable and in [Mud96], 75% is considered the maximum theoretical limit of predictability on this program. We don't know any paper that reports a prediction accuracy greater than 74% on this benchmark. Figure 6 presents comparatively the obtained average prediction accuracies for an untrained neural predictor respectively a trained neural predictor. As it can be observed, the trained neural predictor performs significantly better with about 1% to 7% gain in prediction accuracy. Interesting, for a trained MLP predictor the average prediction accuracy growths slowly correspondingly with growing HRg's length. This is not true for the untrained MLP (correlation information is noisly in this case; perhaps better initial weights values – no purely random like in this approach - could be a better alternative implementation). Anyway, this observation points out clearly that we developed an efficient training algorithm. Using it, our MLP doesn't forget so much some previous learned patterns. Forgetting process is significant when we use a MLP that process directly the trace, without a previous statical controlled training. Figure 7 shows comparatively the average prediction accuracies for the considered classical GAp scheme and respectively the MLP predictor. Note that both these predictors are using the same prediction information, available during the instruction fetch stage (HRg & PC). As it can be seen, the neural branch predictor outperforms obviously the classical one with about 4% gain in prediction accuracy. Finally, figure 8 presents the influence of the learning step (a) for a trained MLP predictor. For a=0.1, 0.5 and 1 we are obtained respectively prediction accuracies (Ap) of 91.20%, 90.70% and 90.83%. During the all the previous simulation, we used a=1 due to some learning time limitations (for small values of “a”and certain small values of the mean squared error, the learning process is too long). Other interesting results are presented in [Vin00].

Because of the additional warm up time our models are likely to perform less succesfully with relatively small benchmarks like Stanford. Anyway, the obtained prediction accuracies are in a perfect concordance with those obtained by other researchers that used Stanford benchmarks in evaluating branch prediction [Ega98]. We would expect to show some improvement in prediction accuracy using larger benchmarks like Spec. Unfortunately, from financial reasons, at this moment we aren’t able to use Spec or other larger benchmarks.

3. CONCLUSIONS AND FURTHER WORK

Therefore our work hipothesis was proved, the neural branch predictor is more adaptive and efficient than a corresponding classical branch predictor. This is not a surprise taking into account that a neural network is a more complex and inteligent structure comparing with a “classical” branch prediction scheme based on a set of prediction automata (saturating counters).

An interesting neural predictor feature could be related to the target prediction for indirect jumps. Taking into account that the target of an indirect branch can change with every dynamic instance of that branch, predicting its target is really difficult. There are few solutions to this problem and all of them involve a great deal of further work. One of the most recent valuable approaches in this sense, improves the indirect jumps prediction accuracy by choosing its target from the most recent targets of the indirect jump that have already been encountered [Cha97], based on a simply heuristic. In contrast, our new approach proposes that the neural network that predicts the branch direction also predict the target effective address, for indirect jumps only, based on the same prediction information: PC, HRg (a global history) and possibly the branch’s own history (HRl – local history). Therefore, this approach is based on a supposed correspondence between the indirect jump's dynamic patterns (PC, HRg, HRl) and its dynamic target addresses. In the nearest future, based on a trace driven simulation method, we'll quantitatively investigate this approach related to target prediction for indirect jumps.

The difficult problem related to these neural predictors is to establish whether they can be implemented on a chip, taking into account the run-time prediction request. More precisely, that means the prediction must be done during the instruction fetch phase for an efficient approach. Based on the present technological progresses, that allow performant neural networks hardware implementations, in our opinion the neural predictor idea could be feasible and, therefore, new investigations in this research area are warranted. At this time, our intuition is that a simplified neural predictor could be designed within the timing restraints of a superscalar. We also suspect that the cost would be less than one of Two Level Adaptive Predictors and it may even be possible to implement multiple cut-down neural predictors, associated which each static cached branch. In fact, our neural predictor replaces the prediction automata that are stored into the Prediction Tables with one global neural prediction structure as we described above. Anyway, and most important at this point of our 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.

ACKNOWLEDGMENTS

This work was supported in part by the Romanian National Agency for Science, Technology and Innovation (ANSTI) grants MCT No. 4086/1998,1999 respectively by the Romanian National Council of Academic Research grants CNCSU No. 391/1998 and CNCSU No. 489/1999. We like to thank Mr. Marius Sbera for his work in our neural predictor research group and for his help in obtaining some quantitative results. Also our gratitude to Professor Gordon B. Steven from the University of Hertfordshire, UK, for providing HSA Stanford traces and for his very useful concrete suggestions and encouragement related to our MII architectures research.

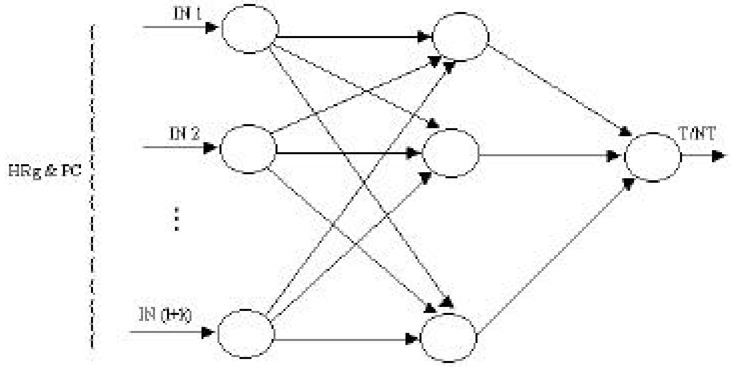

Figure 1. A Simple Neural Branch Predictor

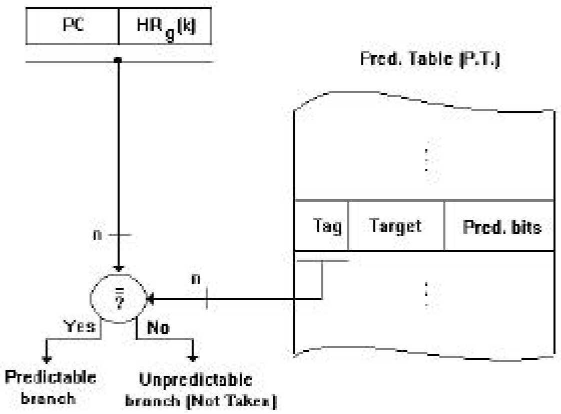

Figure 2. A modified GAp scheme (unlimited no. of entries)

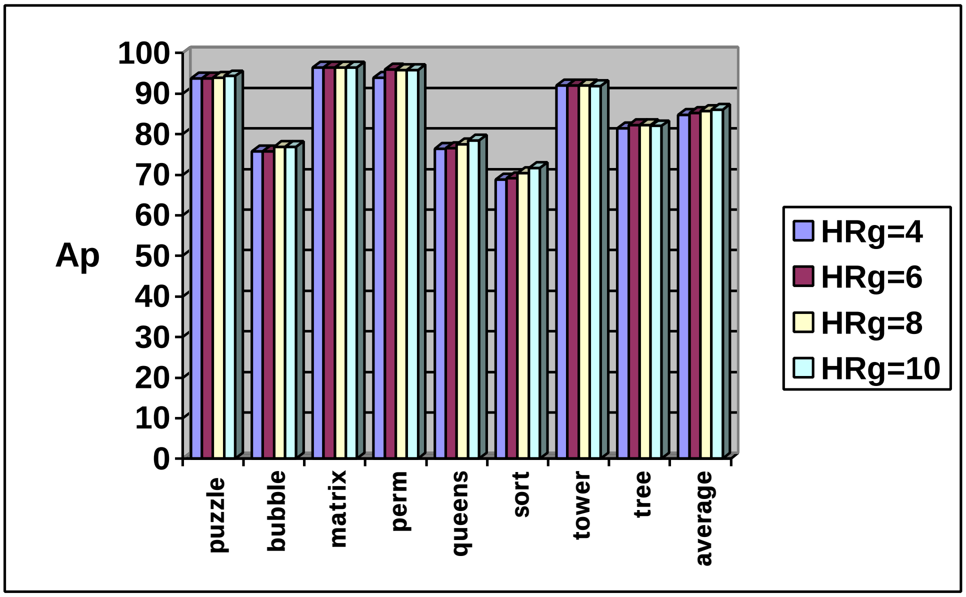

Figure 3. Prediction accuracies for the unlimited GAp scheme

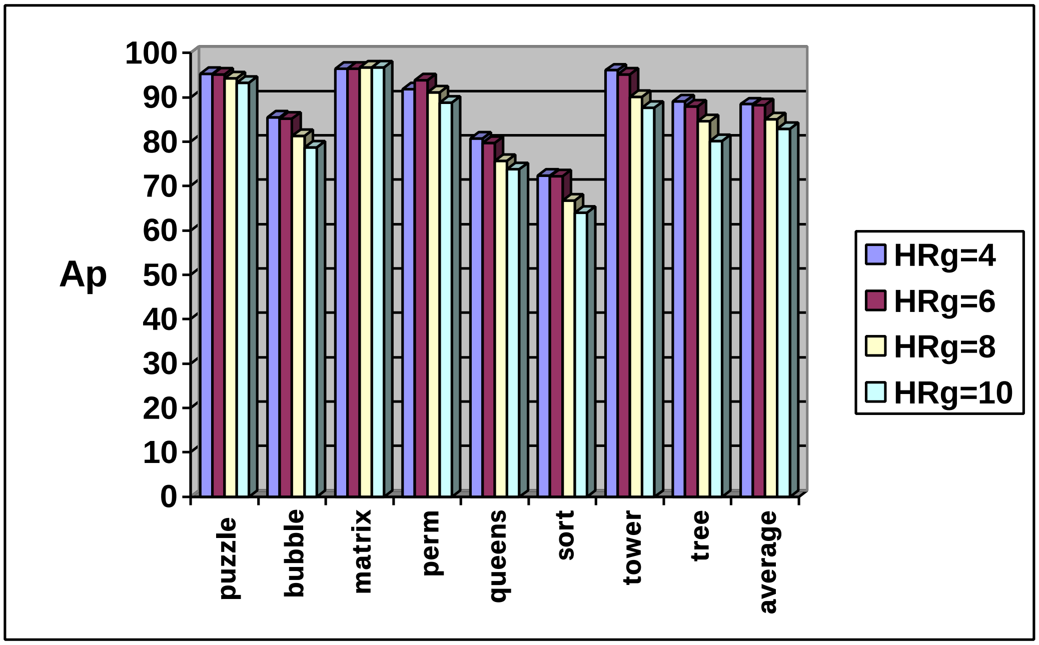

Figure 4. Prediction accuracies for an "only dynamic" trained MLP predictor

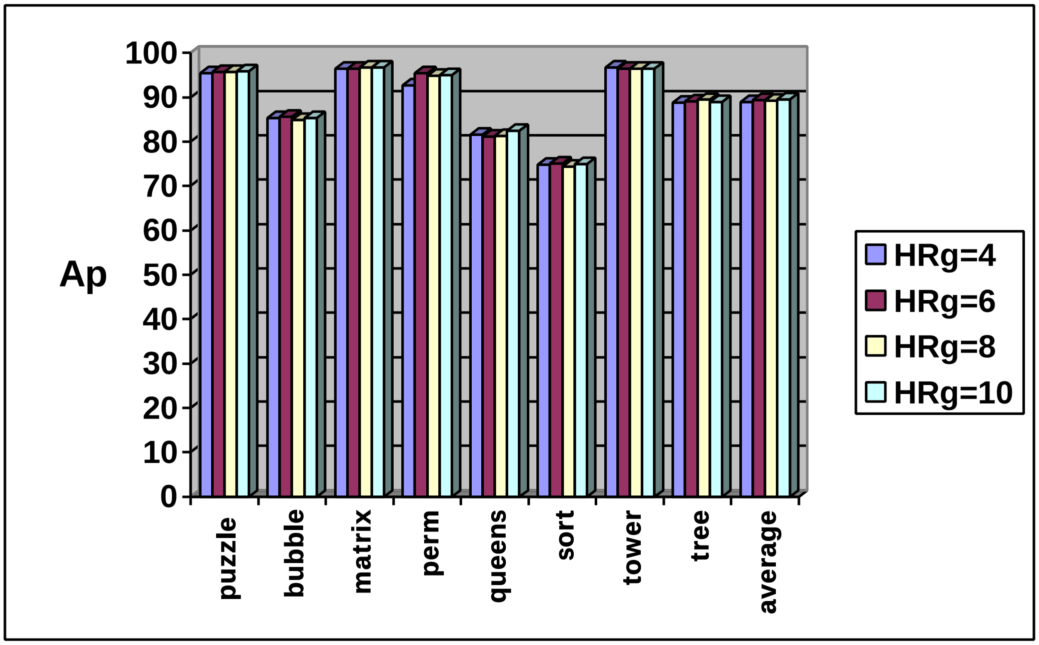

Figure 5. Prediction accuracies for static trained MLP predictor

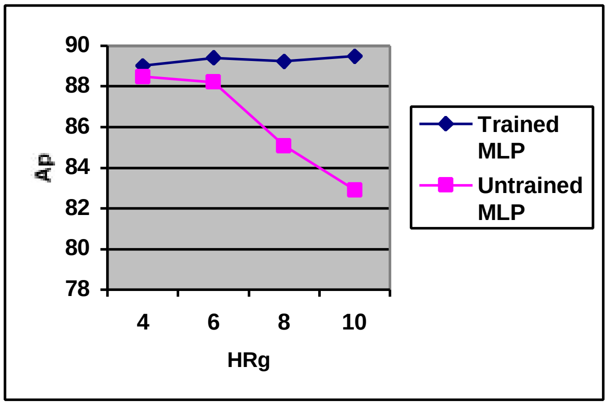

Figure 6. A trained MLP vs. an untrained MLP

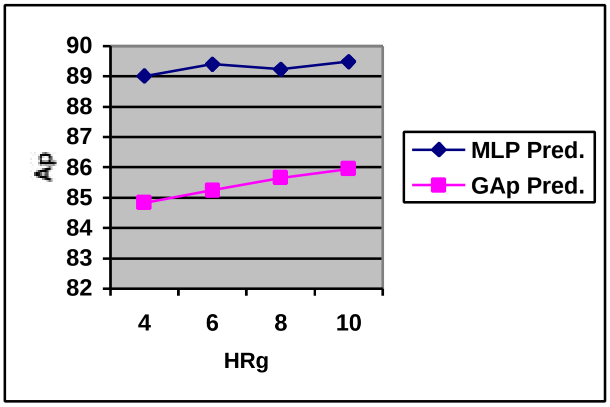

Figure 7.The MLP predictor vs. the GAp predictor

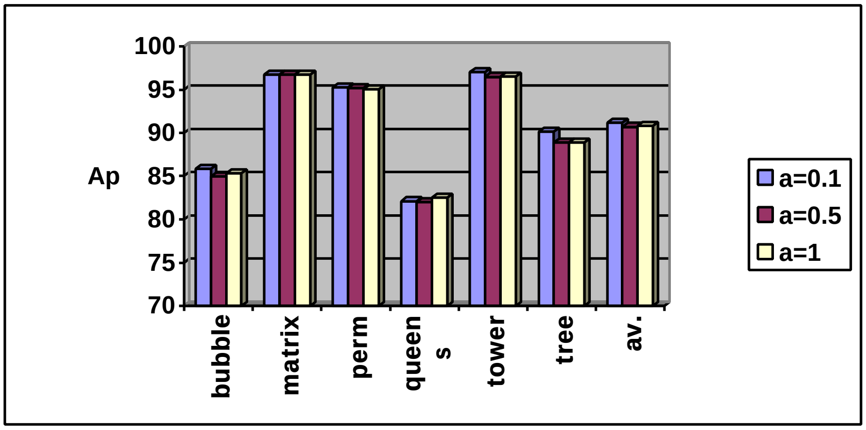

Figure 8.The influence of the learning step value (a)

REFERENCES

[Cha97] **Chang P.Y., Hao E., Patt Y.N**. - *Target Prediction for Indirect Jumps*, ISCA '97 - Ann. Int.'l Symp. Computer Architecture (http://www.eecs.umich.edu/HPS)

[Ega98] **Egan C. -** *Branch Predictor Report*, University of Hertfordshire, Department of Computer Science, UK, November, 1998

[Eve96] **Evers M., Chang P.Y., Patt Y.N**. - *Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches*, ISCA '96 (Ann. Int.'l Symp. on Computer Architecture)

[Gal93] **Gallant S.I**. - *Neural networks learning and expert systems*, The MIT Press, 1993

[Koh95] **Kohonen T., et al -** *Learning Vector Quantization (MLP). Program Package Ver. 3.1*, Helsinki University of Technology, SF-02150 Espoo, Finland, 1995

[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.T., So K., Rahmeh J.T. -** *Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation*, ASPLOS V Conference, Boston, October, 1992

[Per93] **Perleberg C., Smith A. J**. - *Branch Target Buffer Design and Optimization*, IEEE Transactions on Computers, No. 4, 1993

[Rec98] **Reches S., Weiss S.** – *Implementation and Analysis of path History in Dynamic Branch Prediction Schemes*, IEEE Transactions on Computers, No. 8, 1998

[Ste96] **Steven G. B. et al. -** *A Superscalar Architecture to Exploit Instruction Level Parallelism*, Proceedings of the Euromicro Conference, 2-5 September, Prague, 1996.

[Vin99] **Vintan L., Armat C., Steven G. -** *The Impact of Cache Organisation on the Instruction Issue Rate of a Superscalar Processor*, Proceedings of Euromicro 7th Workshop on Parallel and Distributed Systems (http://www.elet.polimi.it/pdp99/), Funchal, Portugal, 3rd -5th February, 1999

[Vin99b] **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

[Vin99c] **Vintan L. -** *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

[Vin00] **Vintan, L.** *Towards a Powerful Dynamic Branch Predictor*, Romanian Journal of Information Science and Technology, Romanian Academy Publishing House, 2000.

[Yeh92] **Yeh T., Patt Y. N. -** *Alternative Implementations of Two Level Adaptive Branch Prediction*, 19 th Ann. International Symp. Computer Architecture, 1992