Proceedings of the 2007 IEEE Symposium on Foundations of Computational Intelligence (FOCI 2007)

Rule Selection in Fuzzy Systems using Heuristics and Branch Prediction

Keerthi Laal Kala
Center for VLSI & Embedded System Technologies
International Institute of Information Technology
Hyderabad, India
klkala@research.iiit.ac.in

M. B. Srinivas
Center for VLSI & Embedded System Technologies
International Institute of Information Technology
Hyderabad, India
srinivas@iiit.ac.in

***Abstract*—Rule bases, providing complete information about the system at hand in a Fuzzy Logic Controller, tend to be huge. Selecting rules that should be applied to the current inputs of the system becomes an increasingly complex task, as the rule base size increases. Techniques have been developed to provide the relevant rules for inference to improve the speed of operation of fuzzy systems. This paper proposes an approach using a simple heuristic to identify a most probable set of rules and then predict the rule that will be used for the current system inputs. A prediction strategy, used for branch prediction in processors, is employed in predicting the rule to be used. The proposed approach has been compared with few approaches for rule selection and results are provided.**

***Keywords*—fuzzy logic controllers, rule selection, branch prediction, fuzzy reasoning, rule extraction, rule weighting**

I. INTRODUCTION

Fuzzy Logic Controllers (FLC) use a rule base, composed of rules derived from expert knowledge, to infer the control actions needed for the current state of system inputs. The rule base is essentially an exhaustive set of '*if …, then…*' conditions that direct the actions to be taken by the system. This set of rules is considered representative of all the possible situations the system will encounter during its normal course of operation. While encompassing the whole range of possible scenarios, the size of the rule base grows prohibitively large and picking a suitable set of rules that should be applied for any given inputs will be a complex task. The complete rule base has to be searched and relevant rules selected for inference. This process is time consuming and might compromise the speed of operation or reaction of the system. To improve response time two approaches have been addressed in literature:

- condensing the rule base to minimize the number of rules, yet store the information needed
- improving the process of selecting the rule to be applied for given system inputs

[1] describes one such condensing approach using rule base refraction. [2] – [4] highlight the use of rule weighting to improve the efficiency in rule selection.

In the current work we focus only on rule selection. Section II provides summary of the rule base condensing approach used to reduce the size of the rule base. In addition, brief summaries of few rule selection approaches proposed in literature are also provided. This section is dedicated to providing brief summaries of existing techniques for handling large rule bases and shows the motivation behind the current work. Section III provides a discussion of branch prediction in modern day processors and gives details of some of the techniques used thereof.

TABLE I. RULE TABLE AFTER ENCODING [1]

Code Word	Tag
0101	1000
0001	1000
1001	0100
1101	0100
0100	1000
0000	0100
1000	0100
1100	0010
0110	0100
0010	0010
1010	0010
1110	0001
0111	0010
0011	0010
1011	0001
1111	0001

Section IV gives details of the proposed approach highlighting the heuristic strategy employed and the adaptation of a branch prediction strategy for rule selection in an FLC.

Section V provides an analysis of the proposed approach and gives comparative results of experiments with a few data sets.

II. Background

This section provides summaries of:

- rule base condensing using refraction
- rule weighting

A. *Rule base condensing using Refraction* [1]

This approach aims at exploiting the consistency and rule neighboring shown in a rule base. This approach is explained here as an example to show the effectiveness of reducing the rule base into an encoded subset, which can lead to faster inference and also to showcase previous work which has been done in order to improve the efficiency of inference in a fuzzy system. The proposed approach in Section IV, was tested on rule bases condensed using this method.

In rule base condensing approach, first the rule base is encoded and then prime-implicants are derived out of the encoded rule base. The prime-implicants are identified as follows:

- Scan the rule table for code words with a Hamming distance of 1 and mark the different bit as a "don't-care" using a '-'. The encoding mechanism involves using 2-bit sequences to represent each possible state and a 4-bit tag to indicate the leading implicant in the code.
- Add these new codes to the new list and check off both original codes (Table I & II). The data presented gives only the main steps in the approach. Interested readers are requested to refer [1] for details.
- Setup a cover table in which the rows are all prime-implicants and the columns are min-terms
- Enter an 'x' at the intersection of row i and column j if the i^{th} prime-implicant P_i covers the j^{th} min-term m_j
- Identify all essential rows and select them for inclusion in the potential solution ser $\boldsymbol{T} = \{S_1, S_2, S_3, \ldots, S_q\}$, where $S_1 \sim S_q$ are inference rules
- Delete all columns covered by these rows and the essential rows themselves. Delete any rows without an 'x'
- Reduce $\boldsymbol{T}$ by removing all columns that dominate a retained column and continue applying this procedure until no further reduction is possible

Once the prime-implicants are reduced to dominant implicants that cover all cases, the implicant table is decoded to generate a fresh rule base that will be a reduced version of the original one.

B. *Rule Weighting*

Weighted linguistic models make use of expert knowledge and a deductive process. The weights applied to a particular rule are used to modulate the firing strength of the rule, in other words a degree of importance is associated with each rule. In addition, complex multidimensional problems have redundant, inconsistent & conflicting rules and the size of the rule base may also be large. The use of weights in such cases can be seen as a local tuning of rules enhancing the robustness, flexibility and control compatibility of such systems [2].

Weight assignment of rules can be achieved in different ways. [2] suggests using genetic algorithms (GA) to select a subset of rules showing good co-operation & deriving the weights associated with these rules. A GA based on the steady-state approach is used, where the fitness function considers objective weighting.

TABLE II. Rule Table After Reduction [1]

Group	Code Word	Tag
0	-000	0100
1	0-01	1000
	010-	1000
	100-	0100
	-010	0010
	001-	0010
2	1-01	0100
	0-11	0010
3	111-	0001
	1-11	0001

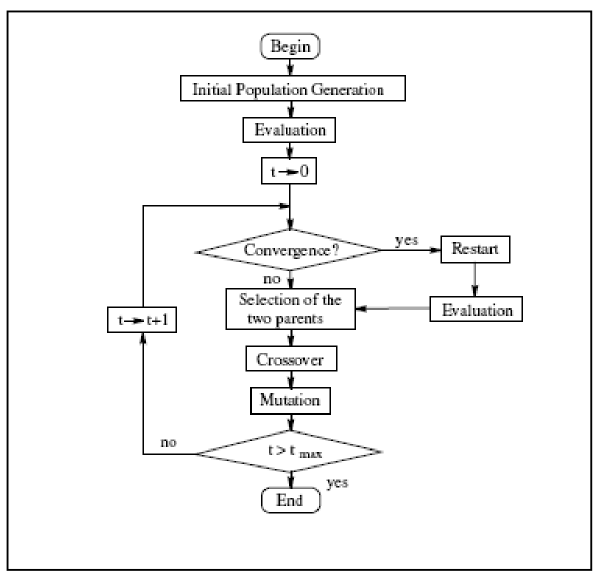

Figure 1. Flowchart of the GA Process

The selection scheme used in the GA is based on Baker's stochastic universal sampling together with elitist selection. Fig. 1 shows a flowchart of this method.

A gene is coded as $C = C1 + C2$, where $C1$ is a binary vector representing the subset of rules obtained from expert knowledge and $C2$ is a real-coded string corresponding to the weight used for each rule in $C1$. The fitness function used decrements the importance of each individual fitness value when it reaches its goal and penalizes each objective whenever its value worsens with respect to the initial solution. The crossover operator for $C1$ is the standard two-point crossover while for $C2$ a hybrid between a BLX-α and an arithmetical crossover is used [5]. The restart block prevents getting trapped in local minima. When the population converges to similar results on multiple iterations, the best individual is kept and the rest of the population is generated again.

[3] on the other hand suggests heuristic rule weight specifications based on the concepts of *confidence* and *support* used in data mining. Assume that a set of antecedent fuzzy sets is given for each attribute. The antecedent part for each rule ($\boldsymbol{A}_q$) is computed by combining the fuzzy sets for all n attributes. The consequent $\boldsymbol{C}_q$ for the fuzzy rule $\boldsymbol{A}_q \Rightarrow \boldsymbol{C}_q$ is determined by the class with maximum confidence for the antecedent $\boldsymbol{A}_q$

$$c(A_q \Rightarrow C_q) = \max\{c(A_q \Rightarrow Classh) \mid h = 1,2,...M\} \quad (1)$$

where c is the measure of the confidence in an M-class classification problem. Hence, there are M consequent classes and each rule attributes the result to one of these classes with a certain confidence. The rule weight is now defined as the certainty grade $\boldsymbol{CF}_q$ given by

$$CF_q = c(A_q \Rightarrow C_q) \quad (2)$$

Another definition for the rule weight in [3] is

$$CF_q = c(A_q \Rightarrow C_q) - c_{2nd} \quad (3)$$

where c_{2nd} is the second largest confidence for the antecedent $\boldsymbol{A}_q$ defined by the following equation

$$c_{2nd} = \max\{c(A_q \Rightarrow Classh) \mid h = 1,2,....M; h \neq C_q\} \quad (4)$$

Both the approaches discussed here provide rule weights to the antecedent rules and thereby make the process of inference efficient. The purpose of discussing these approaches here was to throw light on some of the methods used for rule weighting as described in literature. This section acts as a prelude to the concept of rule weighting. These methods, though not used in the proposed approach, are shown here to qualify the need for effective rule base handling. The results obtained by using one of these methods are used for comparison with the proposed approach.

Our proposed rule weighting method (Section IV) uses a heuristic method to come up with a set of probable rules and then predicts which rule to use from them. The approach we propose is suitable for single winner classification problems.

III. BRANCH PREDICTION [6]

As mentioned earlier, we use a prediction strategy borrowed from branch prediction methods used in current day processors. This section therefore presents a brief overview of branch prediction and strategies used.

The goal of branch prediction is to allow the processor to resolve the outcome of a branch early, thus preventing control dependencies from causing stalls and to aid pipelining of operations. The effectiveness of a branch prediction scheme depends not only on the accuracy, but also on the cost of a branch when the prediction is correct or incorrect. The type of the predictor used plays a major role in deciding the branch penalties involved. The simplest dynamic branch-prediction scheme is a *branch-prediction buffer* or *branch history table*. It is a small memory indexed by the lower portion of the address of the branch instruction that contains a bit which says whether the branch was recently taken or not. This buffer is effectively a cache whose performance depends on both how often the prediction is for the branch of interest and how accurate the prediction is when it matches. Branch predictors that use the behavior of other branches to make a prediction are called *correlating predictors* or *two-level predictors*.

TABLE III. COMBINATIONS AND MEANING OF TAKEN/NOT TAKEN PREDICTION BITS

Prediction Bits	Prediction if last branch not taken	Prediction if last branch taken
NT/NT	NT	NT
NT/T	NT	T
T/NT	T	NT
T/T	T	T

To illustrate the working of a correlating prediction scheme consider the following code fragment:

```
if (d==0)
    d=1;
if (d==1)
```

A typical instruction sequence generated for the code fragment for execution on the processor would look like:

```
    BNEZ      R1, L1         ; branch b1 (d! =0)
    DADDIU    R1, R0, #1     ; d=0, so d=1
L1: DADDIU    R3, R1, #-1
    BNEZ      R3, L2         ; branch b2 (d! =1)
……………..
L2:
```

BNEZ instruction stands for 'branch on not equal to zero' and the DADDIU instruction stands for 'data addition'. The value of d is assumed to be stored in register R1 initially and

R0 & R3 are registers used by the processor for execution of instructions. L1 & L2 are the branch labels. For ease of understanding, the branches corresponding to the two if statements are labeled *b1* and *b2*. In the current example we use two-bit prediction scheme, where the first bit predicts assuming that the last branch executed in the pipeline was not taken (misprediction) and another prediction that is used if the last branch executed was taken. Table III shows all the possible combinations, where T stands for branch taken and NT stands for not taken. Assuming *d* has values 0, 1 and 2, the possible sequences for execution of the code fragment are shown in Table IV. To illustrate how a correlating predictor works, assume the sequence above is executed repeatedly. In general, the last branch executed is *not* the same instruction as the branch being predicted.

The action of the correlating predictor when initialized to *NT/NT* is shown in Table V. In this case, the only misprediction is on the first iteration, when *d*=2. The correct prediction of *b1* is because of the choice of values for *d*, since *b1* is obviously not correlated with *b2*. The correct prediction for *b2*, however, shows the advantage of correlating predictors. Even if different values for *d* were chosen, the predictor would correctly predict *b2* when *b1* is not taken on every execution of *b2* after one initial incorrect prediction.

TABLE IV. POSSIBLE EXECUTION SEQUENCES FOR CODE FRAGMENT

d	***d* = 0?**	***b1***	***d***	***D* = 1?**	***b2***
0	Yes	NT	1	yes	NT
1	No	T	1	yes	NT
2	No	T	2	No	T

TABLE V. ACTION OF CORRELATING PREDICTOR INITIALIZED TO NT/NT

d	***b1* pred**	***b1* action**	***New b1* pred**	***b2* pred**	***b2* action**	***New b2 pred***
2	**NT**/NT	T	T/NT	NT/**NT**	T	NT/T
0	T/**NT**	NT	T/NT	**NT**/T	NT	NT/T
2	**T**/NT	T	T/NT	NT/**T**	T	NT/T
0	T/**NT**	NT	T/NT	**NT**/T	NT	NT/T

The prediction scheme described above has been tested on various benchmarks and on an average the misprediction of the scheme is 5-9% [6].

IV. PROPOSED APPROACH

The approach we propose in this paper uses a heuristic credit allocation scheme to select a set of probable rules that can be applied to current state of the antecedents. The approach is closely related to assigning rule weights and the selection of rules thereof, as will be shown.

Each time a condition is to be evaluated, an auction is held where all the rules in the rule base bid for their solutions to be accepted. Each rule is allowed only a single bid, which is a certain percentage of its current bank balance, known as its *strength*. This value is analogous to the rule weights discussed in previous sections. So, strong rules are more likely to be successful in their quest to buy the right to put forward their *action*. All rules are initialized to the same *strength* and hence have equal probability for being chosen. The rule base adapts itself to the working environment after a few cycles and the efficiency of the system increases.

Once the auction is complete, instead of looking for the strongest bidder and use it to make an inference as intuition would suggest, we take the top four bidders and form a small secondary rule base called the *rule cache*. The size of the cache is a variable in the system and needs to be chosen keeping in mind the size of the rule base and the average number of rules active for given set of inputs. Experiments have shown that a large *rule cache* does not guarantee better performance. Currently the choice of the size of the *rule cache* is limited to Monte Carlo experiments with varying sizes.

This *rule cache* is now subjected to the correlating prediction strategy described in Section III. The predictor is initialized to NT/NT as described earlier.

Once a rule has been predicted and if it is indeed evaluated to be the correct solution, the rule is paid a *reward* reinforcing its *strength* and making it stronger. This update makes the rule more probable for the next set of antecedents. Thus, the system is adapting to the environment in which it is working by giving more importance to relevant rules at a particular instant in time.

The reinforcement update is driven as:

$$S_{new} = S + r \tag{5}$$

where S is the strength of the rule and r is the reward value. The failing bidding rules are penalized and their *strength* is reduced by applying a *penalty tax*, thus reducing its probability for bidding in the next iteration. This operation if performed as:

$$S_{new} = (1 - p) * S \tag{6}$$

where p is the percentage of penalty being used. To prevent unpopular rules from maintaining high *strength* values, an *existence tax* is levied on all non-bidding rules. This strength update is affected as follows:

$$S_{new} = (1 - e) * S \tag{6}$$

where e is the percentage of existence tax being used.

To enable the usage of both the heuristic and prediction strategies, each rule has an associated *strength* value and a two bit prediction. All the rules are initialized to equal strength values and values for *r, p* and *e* are user-defined. The predictions are initialized to *NT/NT* and are updated upon every inference for all rules in the rule cache. The whole process is repeated again for the next antecedent inputs.

The proposed approach makes rule selection a two-level process. The first level involves sorting the rule base and ascertaining four highest bidders. The second level involves predicting which of the rules is to be used. Using the proposed approach, the strengths of the rules in the database are constantly updated. Thus the rule base is constantly adapting itself to provide suitable classifiers very efficiently for a given state of the antecedents. The rule *strength* adoption mechanism can be summarized as:

$$S_{new} = \begin{cases} S + r & ; \quad correct \quad rule \quad found \\ (1-p) * S & ; \quad wrong \quad bid \\ (1-e) * S & ; \quad existence \end{cases} \tag{7}$$

V. Performance Evaluation

A. Results

The performance of the system was evaluated against the heuristic rule weighting and GA based rule weighting schemes described in Section II. The data sets used were the wine data and glass data available from the UC Irvine ML database.

The Wine dataset has 13 variables with 178 samples from three classes of wines. The glass dataset has nine variables with 214 samples from six classes. For the purpose of our experiment, we chose four random variables in each set as antecedents. The datasets were chosen for comparison with the results on the same data sets used in [3]. The rule base in each of the experiments was hand-written by analyzing the data.

The results on the wine dataset are shown in Table VI. From the results it is evident that the proposed approach performs better than the GA based rule weight scheme and outperforms one of the heuristic based weight update schemes presented in Section II.

TABLE VI. Results on Wine Dataset

Number of rules	No rule weights	(2) based	(3) based	GA based	Proposed
3	92.63	93.55	93.51	91.86	93.53
6	91.84	92.13	92.87	92.14	92.56
9	92.48	93.89	94.17	92.27	93.94
12	94.26	94.71	94.96	94.19	94.92
20	95.62	96.18	96.35	95.23	96.37

TABLE VII. Results on Glass Dataset

Number of rules	No rule weights	(2) based	(3) based	GA based	Proposed
3	91.79	91.95	91.95	91.34	92.13
6	89.91	90.22	90.94	90.51	91.18
9	92.57	92.98	92.98	92.56	93.04
12	86.73	87.11	87.35	86.59	87.03
20	85.28	85.54	86.27	85.93	86.31

The results on the glass dataset are shown in Table VII. From the results it can be seen that the proposed approach outperforms the GA based rule weight scheme and also both the heuristic based weight updated discussed.

The simulations were performed using four variables as antecedents for both the datasets. The *strengths* for each system were initialized to 10, the *reward* value was 1, the *penalty* was 3% and the *existence* tax was 1%.

B. Conclusion

In this work, we propose a rule selection approach which uses heuristics and branch prediction strategies. The efficiency of the approach has been tested on two datasets and the results were compared with three other approaches detailed in literature. The results show that the performance of the proposed approach is better than the rest of the approaches in certain cases and close to best in others.

References

[1] Kao-Shing Hwang and Ming-Yi Lu, "Rulebase Refractoring Design for Fuzzy Logic Controllers," International Journal of Intelligent Automation and Soft Computing.

[2] Rafael Alcala, Jorge Casillas, Oscar Cordon, Antonio Gonzalez and Fransisco Herrera, "A Genetic Rule Weighting and Selection Process for Fuzzy Control of Heating, Ventilating and Air Conditioning Systems."

[3] Hisao Ishibuchi and Takashi Yamamoto, "Rule Weight Specification in Fuzzy Rule-based Classification Systems," IEEE Trans. on Fuzzy Systems, vol. 13, no. 4, pp. 428-435, August 2005.

[4] Hisao Ishibuchi and Tomoharu Nakashima, "Effect of Rule Weights in Fuzzy Rule-Based Classification Systems," IEEE Trans. on Fuzzy Systems, vol. 9, no. 4, pp. 506-515, August 2001.

[5] L. J. Eshelman and J. D. Schaffer, "Foundations of Genetic Algorithms," Morgan Kaufmann Publishers, USA, 1993

[6] John L. Hennessy and David K. Patterson, "Computer Architecture: A Quantitative Approach," Morgan Kaufmann Publishers, USA, 2003.