Fifth IFSA World Congress (1993), 761 - 764

RULE-BASE SIZE-REDUCTION TECHNIQUES IN A LEARNING FUZZY CONTROLLER

E. Lembessis[a] & R. Tanscheit[b]

[a]AMBER S.A. Computer Systems, The "Heracles Group", POB 3500, Athens - Greece
[b]Department of Electrical Engineering, PUC-Rio, CP 38.063, 22.452 Rio de Janeiro, RJ - Brazil

Abstract

In this paper we consider techniques for reducing the generated number of rules in learning fuzzy controllers of the state-space action-reinforcement type that can be simply implemented and that behave well in the presence of process noise. Fewer rules lead to better performance, less contradiction in controller action estimation, smaller required execution-time and make it easier for a human to comprehend the generated rules and possibly intervene.

1. INTRODUCTION

The techniques for reduction of the number of rules generated in learning fuzzy controllers were developed through experiments with a Fuzzy Rule-based Self-Organising Controller (SOC) [1,2,3,4], which will be described briefly in the next section. The ideas behind those techniques may also be applied to other controllers that utilise a state-space, reinforcement-type, self-organisation technique. Experimental investigations have shown that fewer rules lead to less contradiction in controller action estimation and to a better overall performance, as well as to an improved execution-time. Besides, a human can comprehend the generated rules more easily when the control strategy is defined by a small number of rules.

2. STRUCTURE OF SOC

The Fuzzy Self-Organising Controller (SOC) is a two-level, hierarchical, rule-based type of controller where the control strategy (i.e. rule-base) is created by the controller itself by applying the self-organising algorithm to the process while executing the control task. SOC can be considered as consisting of an ordinary fuzzy controller at the lower level and of a learning mechanism at the top level.

At each sampling instant, the process output is compared to the setpoint and the 'error' signal is produced; the 'change-in-error' is given by the difference between the present error and the error at the previous sampling instant. Each of these signals is then quantised to one of 13 discrete levels (any number in general) by using scaling factors GE and GCE and is then fed into the SOC system. The lower level fuzzy controller produces a *change in controller output* (u) by using the rules stored in the rule-base, in a coded form, and the current inputs *error* (e) and *change-in-error* (ce). That fuzzy output is then defuzzified by using either the Mean of Maxima (MOM) or the Centre of Gravity (COG) method [5] and the resulting incremental signal is fed, via a summation accumulator, as input to the process being controlled.

At the top level, the learning mechanism is responsible for creating new rules or modifying existing ones. When the process output is not the desired one (as judged by a Performance Index), it is assumed that some action taken by the controller in the past was responsible for that and should therefore be modified. By using a pre-specified *delay-in-reward* (d) parameter, that past action is retrieved, and modified according to the appropriate entry in a *Performance Index Table* ($P \neq 0$). This table is derived from a general set of linguistic rules that express the desired system trajectories in the system's state-space (e x ce). In order to exemplify the learning mechanism, consider an instant i, when the controller inputs are e_i and ce_i and the output is u_i. Assuming that an action u_{i-d}, taken d instants before, is responsible for the current response, the past state at that instant would be given by e_{i-d}, ce_{i-d} and u_{i-d}. By using the

appropriate entry in the performance table, a rule represented in a coded form by (e_{i-d}, ce_{i-d}, u_{i-d} + P(e_i,ce_i)) is added to the rule-base. If there exists a rule with the same antecedents (*e* and *ce*), the old rule is deleted [6]. When P = 0, the performance is deemed satisfactory and no rule modification takes place. On repeating the control task, rules are added or modified with successive improvement of the controller response. Eventually, a satisfactory set of rules is established. In general, learning convergence can be achieved, but this is not a requirement for good control performance.

The system state-space (*e* x *ce*) can be represented by a 2D matrix. In a similar manner, the indices of a 2D matrix can represent which fuzzy sets are associated with the linguistic values of the antecedent variables in a rule. If 13 discrete levels (i.e. 13-point universes) are used for the input variables e and ce, that matrix will have 13 x 13 entries. Each entry corresponds to a particular action u and represents, in the same fashion as for the input variables, the position in the controller output universe where the membership value is 1. Thus there are as many rules stored as there are non-zero entries in this matrix. The shapes of the fuzzy sets used are stored separately or standard forms are implied. These shapes are used as fuzzification kernels during rule retrieval and processing. The spread of these shapes (kernels) define the kernel size, i.e., the number of elements in a quantised universe of discourse for which the corresponding membership function value is non-zero.

Now imagine sumperimposing the 2D representations of the system state-space and the rule storage matrix. Because of the spread of the antecedent fuzzy sets, a "region of influence" for each rule is formed in state-space centering around each rule entry in the 2D storage matrix (that represents membership values of 1). Thus, each point in state-space can be seen to have a "distance" value in relation to each rule-centre. This distance, measured in terms of quanta, dictates whether a rule has effect, and to what extent, when the system is at a particular state (e_i, ce_i).

3. RULE-BASE SIZE-REDUCTION

The identity of a particular controller lies in the rules created and stored, which constitute the amount of information needed to control a particular process. There should exist a minimum amount of information sufficient to achieve this. In a controller that is able to perform different tasks, rules should exist that cover the whole state-space, and, in addition, the state-space should be covered sparsely if a minimum number of rules is to exist. In the other extreme, if rules existed for every point in state-space, then the controller would depart from the fuzzy set description and become a look-up table, an undesirable situation since such a controller could be implemented simply and without the present fuzzy-theory-based structure. The methods considered for reducing the generated number of rules were:

3.1. Static clean-up

The rules that affect the controller output are those either on or near the state-space trajectory. Rules that are far away from the final trajectory are redundant and could be deleted with no effect on the controller performance. "Near" and "far" from the trajectory are defined in relation to the spread of the fuzzy sets defining the rule antecedents [6,7]. In addition, if the MOM defuzzification method is used, two more possibilities exist for reducing the number of rules by static clean-up:

(a) any rules within "effective" distance from the trajectory that have neighbouring rules even nearer to the trajectory can also be deleted, since with MOM only the immediate neighbourig rules contribute to the decision;

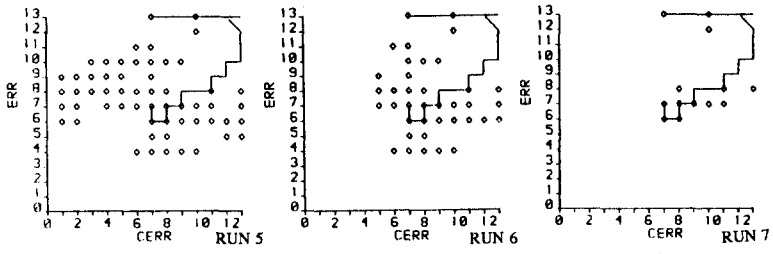
(b) two rules equidistant and on either side of the trajectory can be substituted by a single rule on the trajectory with action $(u^1 + u^2)/2$, where u^1 and u^2 are the individual rule actions.

Static clean-up can be initiated when learning convergence has taken place or when the controller output converges to a repeatable response. An alternative could be to keep track of unused rules and delete these periodically. An example where static clean-up was done is shown in Fig.1, where existing rules and the resulting trajectory in state-space can be seen. Since a constant number of rules was achieved between runs 4 and 5, static clean-up was initiated at run 5. It can be seen that the remaining rules gave identical convergent responses.

One complication that might arise is that noise might make the process visit a state for which there was a rule but that has been deleted as redundant. In such a case learning will be resumed and a new rule created where required.

3.2. Dynamic Deletion

The SOC algorithm can generate rules and alter the actions of existing ones but it cannot delete existing rules that



(a) (b) (c)

Fig. 1. Example of static clean-up
(a) All rules present
(b) Rules within distance 2 from trajectory only present
(c) Effective rules due to MOM procedure only present

might be the cause of bad performance. Rules can be deleted while learning takes place based on system performance, which might be established either on a per sample basis or globally, over N samples. In either case, rules contributing to the decision on that part of the trajectory that does not conform with the required good performance should be located for scrutiny. Unfortunately, the decision about which of these rules should be deleted is not an easy one. Given that the generation of rules is dictated by the structure of the Performance Index Table, the simplest way of avoiding the generation of "bad" rules is through the adjustment of the Performance Table entries.

An alternative scheme is based on the assumption that a new rule generated, by being more recent, would give better performance than an existing rule within "effective" distance and of the same consequent. Since the location of the existing rule is covered by the new rule, the existing rule becomes redundant and should be deleted.

3.3. Dynamic Change of Kernel Size

The basic mechanism behind this scheme is to increase the region of influence of a rule in state-space, when a large generation of rules is sensed, so that fewer rules would be needed to cover the same space. Large rule number generation might be defined according to the complexity of the process controlled and a predefined number of samples over which generation takes place. Both parameters are difficult to set in practice. In addition, since the SOC algorithm only generates rules depending on the demands of the Performance Table, it would require a lot of complexity added so that a rule is not created when the position of the new rule is covered by an existing one. For this reason, the technique did not seem fruitful for reducing the generated number of rules.

On the other hand, this scheme could be used as an independent facility during learning for the purpose of improving control performance: initially, when few rules exist, we maintain a large kernel size and hence a larger area in state-space is covered by the few rules. As more rules are generated, we decrease the kernel size in steps, thus decreasing the area of influence of a rule and making their application more accurate.

3.4. Stopping rule generation (SRG)

This technique involves reducing the number of rules by means of defining positions on the state-space where rules are allowed to exist and others where they are not allowed. Thus, irrespectively of the demands of the Performance Table, the new rules generated will be used to substitute existing rules or inserted directly in the rule-base only if they occupy allowed positions in state-space. If they do not, then the new rules generated are ignored. The simplest method of defining allowed rule positions is by means of a grid pattern superimposed on the state-space.

The simplest grid pattern is one where alternate positions in state-space are occupied by rules, as shown in Fig. 2a. This pattern can be applied over the error or change-in-error universe only, or over both universes. For 13 points universes, the total number of rules will be reduced from 13 x 13 = 169 without the grid to 13 x 7 = 91 in the first case, and to 7 x 7 = 49 in the second. When tried in practice, this scheme proved very effective in reducing the number of rules and hence the execution-time. A problem that may be encountered with the introduction of the grid pattern is the possibility of the system state "sticking" to one position in state-space where there are no rules available to provide a controller output other than the default zero value. Moreover, no rules can be created either, because of the grid restrictions.

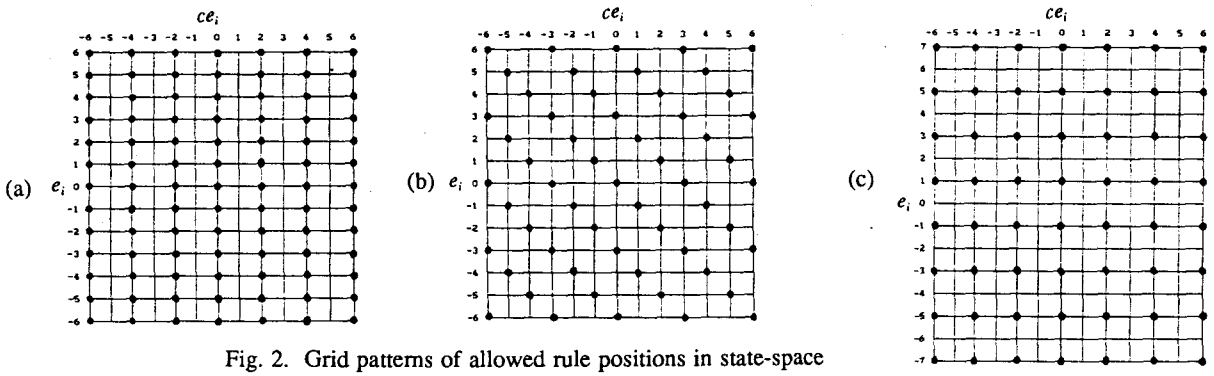


Fig. 2. Grid patterns of allowed rule positions in state-space

The effect of this on the response is that it will possibly show a large steady-state error. If this effect occurs, a way of overcoming it is by adjusting the trajectory by means of the scaling factors - mainly GCE. Alternatively, a grid pattern can be used where there are no complete paths in state-space through which the trajectory could go without creating rules. Such a diagonal type grid is shown in Fig. 2b.

Considering that rules existing on the setpoint might force the trajectory away from it if they are not accurate enough, a useful grid pattern is one that does not allow rules on the setpoint, but only surrounding it, as shown in Fig. 2c. To achieve this pattern and also have a symmetrical grid with an allowable trajectory-starting rule ($|e_i|>>1$, $ce_i=0$) the error universe has to be increased to 15 points. Pairs of rules surrounding the setpoint generally have opposite signs and approximately equal value actions for a general linear process model. Therefore, trajectories in-between those rules will be forced onto $e_i=0$. Such patterns produce good results.

4. CONCLUSIONS

In conclusion, we mention that the techniques of static clean-up and of dynamic deletion involve the deletion of rules from the rule-base after they have been generated, whereas the SRG technique involves the generation of fewer rules to start with. The static clean-up technique can be considered as a technique for tidying-up the rule-base and for improving execution-time. It does not alter the control performance. The dynamic rule deletion technique offers fewer total-rules generated, slightly improved execution-time and slightly improved control performance in certain cases. The SRG technique offers vast reduction in total-rules, greatly improved execution-time, better control performance and can be simply implemented. In addition, SRG proved useful (even with increased grid spacing) when the process was disturbed randomly, since a large amount of unnecessary rules, that would deteriorate control performance, was not created.

5. REFERENCES

[1] Yamazaki, T. & Mamdani, E.H., (1982). "On the performance of a rule-based self-organising controller". Proc. IEE Conf. on Applications of Adaptive and Multivariable Control, Hull, UK.

[2] Mamdani, E.H., Ostergard, J.J., Lembessis, E., (1984). "Use of fuzzy logic for implementing rule based control of industrial processes". In: Fuzzy Sets and Decision Analysis, H.-J. Zimmermann, L.A. Zadeh, & B.R. Gaines (Eds), North-Holland.

[3] Tanscheit, R. & Scharf, E.M., (1988). "Experiments with the use of a rule-based self-organising controller for robotics applications". Fuzzy Sets & Systems, Vol. 26, n. 2: 195-214.

[4] Sugiyama, K., (1988). "Rule-based self-organising controller". In: Fuzzy Computing, M.M. Gupta & T. Yamakawa (Eds), Elsevier Science Publishers.

[5] Lembessis, E. & Tanscheit, R., (1991). "The influence of implication operators and defuzzification methods on the deterministic output of a fuzzy rule-based controller". Proc. 4th. IFSA Congress, Brussels.

[6] Lembessis, E., (1984). "Dynamic learning behaviour of a rule-based self- organising controller". Ph.D. Thesis, University of London.

[7] Tanscheit, R. & Lembessis, E., (1991). "On the behaviour and tuning of a fuzzy rule-based self-organising controller". In: Mathematics of the Analysis and Design of Process Control, P. Borne, S.G. Tzafestas, & N.E. Radhy (Eds), Elsevier Science Publishers.

Acknowledgement

This research was partially supported by MCT and CNPq, Brazil.