A New Learning Method for the Design of Hierarchical Fuzzy Controllers Using Messy Genetic Algorithms

Frank Hoffmann, Gerd Pfister
University of Kiel
Institute of Applied Physics
24098 Kiel, Germany
Email: hoefi@ang-physik.uni-kiel.de

Abstract

An automatic design method for fuzzy controllers with a hierarchical prioritized structure is proposed. A messy genetic algorithm is used to learn different types of behaviour which are represented by a hierarchical set of fuzzy rules. We demonstrate that messy genetic algorithms are well suited to the task of learning because they allow a flexible representation of the hierarchical prioritized structure.Finally we have applied the method to the problem of controlling a physical autonomous vehicle, which given the task of reaching a given location while avoiding obstacles on the way.

1. Introduction

There is a great deal of current interest in developing the design of fuzzy logic controllers using genetic algorithms [1][2][3]. The learning method described in this paper uses only an objective function, which evaluates the performance of a fuzzy controller. Our method differs from other approaches in two special ways. First of all a hierarchical prioritized structure is used to represent the rules [4]. As in fuzzy classifier systems the rule base is constructed from general rules, and specialized ones dealing with exceptional situations. Secondly we use a messy genetic algorithm [5] which process the variable-length strings, in contrast to standard genetic algorithms which work with a fixed length coding scheme. Messy genetic algorithms therefore allow a flexible representation of fuzzy rules in the controllers rulebase. We demonstrate an application of our method in teaching a controller for a behaviour-based autonomous agent [6].

2. Messy GA

Genetic algorithms are optimisation methods which are used to process a population of strings by applying genetic operators such as selection, recombination and mutation. Solutions of the optimisation space are coded as fixed-length, fixed-locus strings defined over an alphabet of alternatives at each position. The goodness of fit of an individual is calculated by a scalar objective function which evaluates the quality of solutions coded by strings. GAs are particularly well suited for nonlinear fitness functions with many local maxima, where the overall fitness can not be decomposed into contributions from single genes.

In classical genetic algorithms the meaning of a gene is completely defined by its position within the string. Using fixed-length strings ensures that each gene occurs exactly once in the string and is always located at the same position. In messy genetic algorithms each gene is tagged with an additional number. Each geneI is a pair of a number determining its meaning and the value itself. For example the binary string (**1001**) translates to ((**1,1**) (**2,0**) (**3,0**) (**4,1**)) in a messy GA. The genes in a messy string can be permutated in any way, without altering their meaning. The two strings ((**1,1**) (**2,0**) (**3,0**) (**4,1**)) and ((**3,0**) (**1,1**) (**4,1**) (**2,0**)) are equivalent. Messy strings are not restricted to contain a full gene complement, and the same gene can occur more than once in a string. The strings ((**2,0**) (**1,0**) (**4,1**)) and ((**4,0**) (**3,0**) (**4,1**) (**1,1**) (**2,1**)) are both valid although in the first example gene number **3** is missed, and in the second one the gene with number **4** is present twice.

While the selection mechanism remains unchanged, the crossover operator is replaced with two simpler operators: splice and cut. A schematic diagram of the cut and splice operator are displayed in Fig. 1. The cut operator simply cuts the string in two parts at randomly chosen position. The splice operator concatenates two strings, which could have been previously cut, in a randomly chosen order. When the cut and splice operators are applied simultaneously to two parent strings they alt in a similar way to the ordinary crossover operator. In messy GAs the positions of cuts in strings,

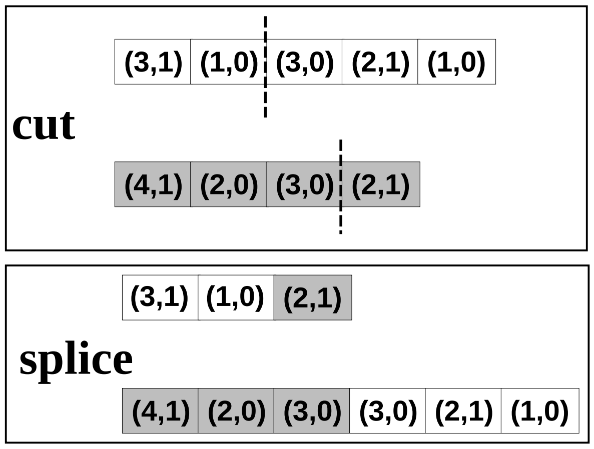

Figure 1: Schematic of cut and splice operators

which are to be joined can be chosen independently, whereas in classical GAs the crossover points must coincide.

3. Hierarchical Fuzzy Controller

A fuzzy controller is composed of a set of rules of which each suggest a certain control output when a given input condition is matched. The knowledge of a fuzzy system is stored in the rule base in which rule j is of the form:

if $X_1 = A_{(1,j_1)}$ **and** $X_2 = A_{(2,j_2)}$ **and** $\ldots$
$\ldots$ **and** $X_n = A_{(n,j_n)}$ **then** $Y = B_{(j_0)}$

where the X_i are fuzzy input variables and the $A_{(i,j)}$ is the jth fuzzy set of the ith variable. Y is the fuzzy output variable with fuzzy sets $B_{(j)}$. The number of clauses n occuring in the conditional step can differ from rule to rule, because some of the input variables can be omitted. This means that the output suggested by this rule is not effected by these specific input variables. One can regard a fuzzy controller as a representation of a control function $y = f(x)$ mapping an input state vector x to a control output y. Each rule suggests a control action which can be applied to those inputs which fall in the region of input space covered by the fuzzy sets in the condition part.

Now we consider two rules where the first one depends on a single input variable X_k alone. It be can regarded as a rule of thumb as it covers a large region of input space. The second rule depends on all of the input variables with the same fuzzy set $A_{(k,j)}$ as the first one associated to X_k. It can be viewed as an exceptational rule which applies in a small number of situations. In the normal defuzzification process the general rule will contribute with at least the same amount to the output, as the highly specified one, even if the input matches completely the remaining fuzzy sets. A human controller would always favour the control action proposed by the highly specific rule if it applies and would disregard the general rule of thumb for this situation. Yager [4] suggested a hierarchical prioritized structure (HPS) to overcome this problem. The overall rulebase is organized in several levels of decreasing priority. Rules with highly specific antecedents are attached to the lowest level, while the higher levels are containing rules with more general information. If the input matches a rule in the lowest level, all general rules in the lower priority levels are ignored. For future reference we simply assume that as soon as the degree of membership of any rule in a lower level is different from zero, all rules in the next levels are ignored. The level to which a rule is attached, depends on the number of clauses in the conditional part. In the highest level all rules only contain a single variable in the antecedent part. Any rule constituted by two clauses is put into the next lower level and so on. Finally the highest priority level consists of rules with the maximum number of input variables in their conditional part.

4. Coding of Fuzzy Rule Base

The coding scheme allows a representation of a fuzzy rule base in a genetic string. Karr [1] suggested a method in which the outputs of rules are coded as integers and has successfully applied it to design a fuzzy controller for a cart pole balancing problem. The number of rules in the rule base is fixed and increases algebraically with the number of input variables. An earlier approach [7] demonstrates that by using a hierarchical fuzzy controller the number of rules can be reduced. A reordering operator for the genetic algorithm has been proposed to reduce the disruptive effect of crossover when combining different rulebases. Leitch [2] suggested a coding scheme for fuzzy rule bases with a varying number of rules for which the position and shape of the input sets are included in the chromosomal representation. The strings are interpreted by a parser and contain additional codons used as context switches to determine the context of a section of code. The coding scheme that we have adopted uses the capability of messy GAs to encode information of variable structure and length. Our fuzzy controller has both input and output variables, where the universe of discourse of each of the variables is covered by fixed fuzzy sets given by the designer in advance. The basic element of our coding is a fuzzy clause, which is represented as a pair of integers. The first one determines the variable

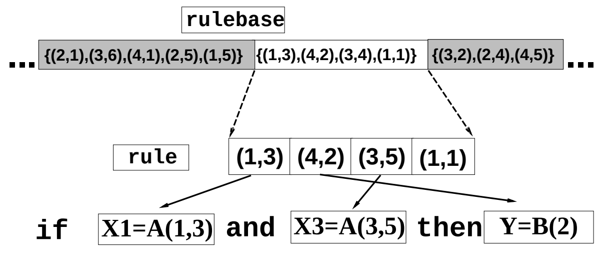

Figure 2: Coding scheme for the hierarchical fuzzy controller

and the second one refers to the fuzzy set of this variable. For example the clause **(3,5)** stands for the fifth fuzzy set of the third variable. No distinction is made between input and output clauses. Clauses are combined to constitute rules in a way that input clauses form the condition part of a rule, whereas output clauses form the consequence. The coding scheme is best illustrated in the following example. Assume our controller has three input variables (X_1, X_2, X_3) and a single output variable Y. The messy string (**(1,3) (4,2) (3,5)**) corresponds to the following fuzzy rule

if $X_1 = A_{(1,3)}$ **and** $X_3 = A_{(3,5)}$ **then** $Y = B_{(2)}$

Overspecification of messy strings can be handled by using a first-come-first-served precedence, which means that if the same variable occurs more than once, only the left most occurrence is expressed in a rule. For example in the messy string (**(1,3) (4,2) (3,5) (1,1)**) $(\mathbf{1},\mathbf{1})$ is be suppressed by the first clause $(\mathbf{1},\mathbf{3})$. Underspecification arises if for some input variables no corresponding clause occurs in the messy string. These variables are simply omitted in the conditional part of the rule. In the above example this is the case for the second variable for which no clause is present in the string. A further problem concerns the situation where messy strings in which no clause corresponding to an output variable appear. The simplest solution is to skip this rule. We actually provide a repair mechanism which inserts a randomly chosen clause, whenever there is no output clause present. The rule strings themselves can be viewed as genes of an overall messy string coding the complete rule base. In this second hierarchy of messy coding the basic elements are fuzzy rules. Depending on the number of input clauses each rule is attached to one of the levels of the fuzzy controller with a hierarchical priortized structure. Underspecification occurs when none of the rules in any of the levels matches the current input state. In this situation a default control action is assumed. If the messy coding contains two rules with an identical conditional step which suggests conflicting control output again a first-come-first-served precedence mechanism is used to overcome the overspecification problem. The complete coding scheme is illustrated in Fig. 2. The cut and splice operators essential for the messy genetic algorithm are applied to both levels of the code. On the lower level in which rules are constituted by clauses cutting and splicing leads to new more or less complex rules. On this level rules compete among each other, where those that provide good control actions for some input are favoured. During learning the algorithm must search for those combinations of input clauses that cover the part of input space in which the control action can actually be usefully applied. At a higher level rules have to cooperate to solve the common control task. A control action proposed by a single rule can be useful for some input states of the dynamical system. Different rules for different parts of the input space are necessary to successfully control the system in all possible situations. Cooperation among rules also emerges, when a less specific rule is covered by a highly specific one on a lower level. The individual fitness of a rule highly depends on the context in which it is used. This nonlinear dependency among the rules is known as epistasis. The GA has to learn which rules are generally useful, and afterwards which combination of rules is able to cooperate successfully.

5. Learning Behaviour

The interest of designing behaviour-based autonomous agents has grown in the last few years [6]. Robots which are capable of learning can react more flexible when confrontated with new situations. They can adapt their behaviour to changing circumstances like noisy sensor data or incomplete information about the environment.

A hierarchical fuzzy controller was designed with the above described methodology for an autonomous agent. The aim was to get the vehicle (Fig. 3) to reach given locations in indoor environments. Obstacles like walls or objects need to be recognized and collisions are to be avoided.

Input variables for the agent are distances in forward direction to obstacles obtained from five ultrasonic sensors. Further input variables are distance and relative orientation of vehicle to the goal the vehicle should reach. Steering angle is used as output variable from which actual velocities of wheels are calculated.

The autonomous agent has to learn two types of behaviour in order to avoid collisons and reach the goalpoint. Whenever an obstacle lies in the path the agent must make an evasive manouevre. If no

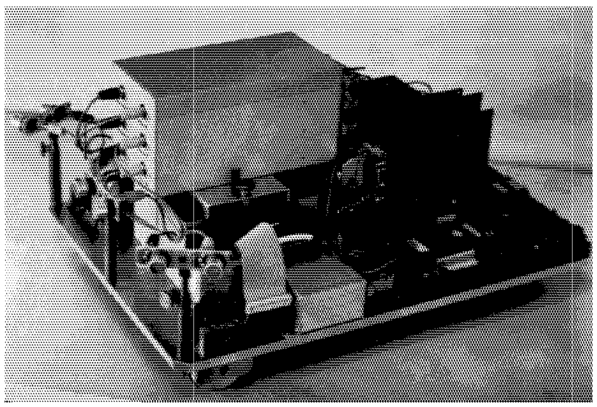

Figure 3: Autonomous robot

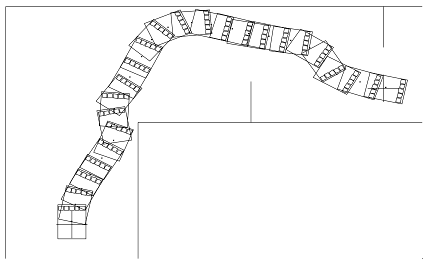

Figure 4: Run of the autonomous vehicle

obstacles are located near the vehicle it can switch safely to goal point behaviour. The environments in which learning takes place are closed rectangle rooms in which obstacles are located. The environments differ in size and position of the rectangles, as well as the locations of starting and goal point, and the initial orientation of the vehicle. The fitness function describes the performance of the agent with respect to a given task. An agent achieves a good fitness, if the vehicle reaches the goal point without collision with one of the obstacles. Agents are favoured that take short paths. Each agent is tested for a set of environments. When using a fixed test set the GA tends to learn the geometry of the environments, instead of the general control task. The agents performed well in those environments presented during the evolution process, but showed poor performance for unseen positions of obstacles and goals. This problem can be partially alleviated by creating new test sets after some generations have passed in the evolution process. Adding noise in the simulation of sensors will also increase the robustness of the controllers. Leitch [2] has suggested to using another GA to evolve the test sets themselves. The fitness of an environment is the inverse of the product of agents fitnesses tested against it. During evolution more test sets arise and the ability of the controllers to cope with more difficult situations increases. The controller obtained from the simulation has been transfered to the real autonomous vehicle. Fig. 4 shows a typical run of the robot in our floor, driving round a corner and passing two gaps.

6. Conclusion

Our paper describes a design method for hierarchical fuzzy rule bases using genetic algorithms. We presented a new coding scheme using messy strings which allows a natural representation of fuzzy rule-bases with a hierarchical priortized structure. It has been demonstrated that an autonomous agent is able to learn simple reactive behaviours for a certain class of environments and tasks.

References

[1] C. L. Karr, "Design of a Cart-Pole Balancing Fuzzy Logic Controller using a Genetic Algorithm" *SPIE Conf. on Applications of Artificial Intelligence*, Bellingham, WA, 1991

[2] Donald Leitch, Penelope Probert, "New Techniques for Genetic Development of Fuzzy Controllers", submitted to *IEEE Trans. on Syst., Man and Cybern.*

[3] J.L. Castro, M. Delgado and F. Herrera, "A Learning Method of Fuzzy Reasoning by Genetic Algorithms", *EUFIT 93*, Aachen 1993, vol. 2, pp. 804-809

[4] Ronald R. Yager, "On a Hierarchical Structure for Fuzzy Modeling and Control", *IEEE Trans. Syst., Man and Cybern.*, vol. 23, no. 4, pp. 1189-1197,1993

[5] David E. Goldberg, Bradley Korb, Kalyanmoy Deb, "Messy Genetic Algorithms Motivation, Analysis and First Results", *Complex Systems* vol. 3, pp. 493-530, 1989

[6] Rodney A. Brooks, "Intelligence Without Reason", *A.I. Memo No. 1293*, MIT, April 1991

[7] F. Hoffmann, G. Pfister, "Automatic Design of Hierarchical Fuzzy controller Using Genetic Algorithms", *EUFIT 94*, Aachen, Germany 1994