ELSEVIER

Fuzzy Sets and Systems 80 (1996) 273–293

FUZZY sets and systems

Evolving fuzzy rule based controllers using genetic algorithms

Brian Carse[a,*], Terence C. Fogarty[b], Alistair Munro[c]

[a] *Intelligent Autonomous Systems Laboratory, Faculty of Engineering, University of the West of England, Bristol, Coldharbour Lane, Frenchay, Bristol BS16 1QY, UK*

[b] *Bristol Transputer Centre, Faculty of Computer Studies and Mathematics, University of the West of England, Bristol, Coldharbour Lane, Frenchay, Bristol BS16 1QY, UK*

[c] *Centre for Communications Research, Department of Electrical and Electronic Engineering, University of Bristol, Bristol BS8 1TR, UK*

Received October 1994; revised May 1995

Abstract

The synthesis of genetics-based machine learning and fuzzy logic is beginning to show promise as a potent tool in solving complex control problems in multi-variate non-linear systems. In this paper an overview of current research applying the genetic algorithm to fuzzy rule based control is presented. A novel approach to genetics-based machine learning of fuzzy controllers, called a Pittsburgh Fuzzy Classifier System #1 (P-FCS1) is proposed. P-FCS1 is based on the Pittsburgh model of learning classifier systems and employs variable length rule-sets and simultaneously evolves fuzzy set membership functions and relations. A new crossover operator which respects the functional linkage between fuzzy rules with overlapping input fuzzy set membership functions is introduced. Experimental results using P-FCS1 are reported and compared with other published results. Application of P-FCS1 to a distributed control problem (dynamic routing in computer networks) is also described and experimental results are presented.

Keywords: Artificial Intelligence; Engineering; Control theory; Evolutionary computation; Genetic algorithms

1. Introduction

Fuzzy control has been employed with success in many diverse practical applications: control of a cement kiln [8], 2-dimensional motion control [9], traffic control [22] and temperature control of air streams [18] to name but a few. Based on Zadeh's theory of fuzzy sets [32], a typical fuzzy controller [16] maintains a rule-base of fuzzy rules and associated fuzzy sets for mapping real-numbered inputs to outputs.

In most existing real application fuzzy controllers, the rule-base is populated with rule-of-thumb fuzzy relations and linguistic fuzzy sets elicited from human experts who have acquired their knowledge through experience. These fuzzy rules and sets are often then fine-tuned by engineers to produce effective fuzzy controllers. A good measure of interest is currently being directed at discovering fuzzy relations and sets automatically using machine learning. Research utilising supervised and unsupervised

*Corresponding author. Tel.: + 44-117-9656261. E-mail: b-carse@csd.uwe.ac.uk.

SSDI 0165-0114(95)00196-4

learning using neural networks [12, 13] has achieved a great deal of success in learning fuzzy relations and sets using known input/output data sets. A major challenge in current fuzzy control research is learning good controllers for large-scale, non-linear systems with many input and output variables where no training data are available from an expert. Genetics-based machine learning is one possible candidate for meeting this challenge.

This paper describes an approach to genetic algorithm (GA) based reinforcement learning in fuzzy control based on the Pittsburgh model of learning classifier systems. The method proposed builds upon a wide base of reported work investigating genetic operations and representations for both fuzzy and discrete classifier systems, using Pittsburgh and Michigan approaches. In the approach presented, genetic operations (selection, recombination and replacement) and credit assignment are carried out at the level of the complete fuzzy rule-set (i.e. Pittsburgh-style) and the number of rules in each rule-set is allowed to vary. The representation employed allows the GA to operate on both fuzzy set membership functions and fuzzy relations (rules). Together, these features endow the classifier system with an ability to evolve rule-sets with economical coverage of the input/output space where the coverage of fuzzy rules and accompanying fuzzy set membership functions is automatically adapted to the complexity of the underlying input/output mapping. This is a useful characteristic if the method is to scale well to complex problems with high dimension input and output spaces.

The paper is organised as follows. Section 2 focuses on different approaches to learning classifier systems and the role of the genetic algorithm in learning. The section also discusses a number of hybrid Pittsburgh–Michigan systems which have been employed with some success in overcoming the deficiencies of each. Sections 3 and 4 lay the foundations on which our current work is built by reporting research already carried out in the field of fuzzy classifier systems using both Pittsburgh and Michigan approaches. In Section 5 we present a detailed description of the representation and genetic operators used in our Pittsburgh-style fuzzy classifier system. In particular, we discuss the role of crossover in the generation of new fuzzy rule-sets and introduce a new crossover operator for use with variable-size fuzzy classifier systems. Section 6 presents and analyses results we have obtained in applying the system to three tasks: firstly, a one-input, one-output function identification task; secondly a two-input, one-output control task; and thirdly a four-input, two-output distributed control problem. Finally, in Section 7 we conclude and recommend areas where future work on genetics-based machine learning applied to fuzzy control may be fruitful.

2. The role of the genetic algorithm in learning classifier systems

In this section we summarise and discuss two alternative ways in which the genetic algorithm (GA) may be applied to learning classifier systems. These two methods, Michigan and Pittsburgh approaches, were first described as long ago as 1978 and 1980, respectively. It should be stated that both approaches are the subject of ongoing research and many significant extensions have been devised and used as the basis for successful learning systems. We begin our discussion by outlining the Michigan approach, since this was employed in the first published report of a learning classifier system.

The first Michigan-style classifier system was Cognitive System One (CS-1) devised by Holland and Reitman [7]. CS-1 maintains a population of classifiers with genetic operations and credit assignment applied at the level of the individual rule. Each classifier in the population has an associated strength, which is used to store an accumulation of credit. The original CS-1 credit apportionment algorithm is epoch-based, where classifiers activated since the last payoff event share the reward collected from the environment at the next. Since CS-1, a large number of alternative credit assignment schemes have been proposed, most notably the bucket-brigade [6] and Q-learning [26, 30, 21] for dealing with environments where reward may be infrequent and/or delayed. These proposed credit assignment schemes have achieved a great deal of success, although many problems regarding their use remain the focus of research. The GA in

a Michigan-style classifier system operates at the level of the individual classifier with selection of parent classifiers for mating based on strengths (and in some cases other parameters such as classifier age or relevance). In addition, classifier strengths in discrete Michigan-style systems are commonly used in controlling the dynamic behaviour of the classifier system by forming the basis for conflict resolution between simultaneously matched classifiers. In the fuzzy case, some practitioners use classifier strengths as weights which influence the level of contribution of rule consequents.

In 1980, Smith [25] published results of an alternative learning classifier system, LS-1, in which the unit of genetic manipulation is a suitably encoded genotype representing a complete set of classifiers. Credit is assigned to complete sets of rules via interaction with the environment. This typifies so-called "Pittsburgh"-style classifier systems. Since the complete rule-set is the basis of credit apportionment, Pittsburgh-style classifier systems sidestep completely the potentially knotty problem of sharing out credit to individual rules. The GA in LS-1 operates at different levels: at the highest level, complete rule-sets are selected as the basis for reproduction to generate new rule-sets; at the lowest level individual rules are chosen by the GA to generate new rules. For the purposes of this discussion we are primarily concerned with the highest of these levels of reproduction. LS-1 uses variable-length classifier sets, and employs modified genetic operators for dealing with these variable-length, position independent (as far as phenotypic expression is concerned) genomes. We will examine these operators in greater detail in Section 5.

Clearly the role of the genetic algorithm in Pittsburgh and Michigan approaches is rather different, and the distinction arises from the difference in level at which the genetic algorithm is applied. Both approaches, at least in their simplest forms, suffer from distinct, known problems which arise from the different way in which the genetic algorithm is applied.

The major problem in the Michigan approach is that of resolving the conflict between the individual and collective interests of classifiers within the system. The ultimate aim of a learning classifier system is to evolve a set of co-adapted rules which act together in solving some problem. In a Michigan style system, with selection and replacement at the level of the individual rule, rules which cooperate to effect good actions and receive payoff also compete with each other under the action of the genetic algorithm. Such a conflict between individual and collective interests of individual classifiers does not arise with Pittsburgh-style classifier systems, since reproductive competition occurs between complete rule-sets rather than individual rules. However, maintenance and evaluation of a population of complete rule-sets in Pittsburgh-style systems can often lead to a much greater computational burden (in terms of both memory and processing time). Wilson and Goldberg [31] propose a classifier system which clusters classifiers into "corporations". Classifiers belonging to the same corporation do not compete with each other under the action of the GA and corporations form and break up under the action of a modified crossover operator. A successful implementation of this approach is Shu and Schaeffer's [24] "hierarchical" classifier system in which classifiers are grouped into "families" which form the basic units of selection by the GA. Such approaches represent a middle ground between Michigan-style and Pittsburgh-style classifier systems.

Problems with the Pittsburgh approach have proved to be at least equally as challenging. Although the approach avoids the problem of explicit competition between classifiers, large amounts of computing resources are required to evaluate a complete population of rule-sets. A further problem with the approach is the small bandwidth of reinforcement information, usually a single scalar fitness value for each complete rule-set. If information about the performance of individual rules happens to be derivable from the pattern of environmental payoffs using some credit apportionment method, this information is not explicitly exploited in the Pittsburgh approach. The disruptive threat to good collections of cooperating rules comes from a different source in Pittsburgh-style systems compared to Michigan-style systems. In the latter, competition at the level of selection and replacement of individual rules can destroy

good rule associations. In a Pittsburgh-style system, although selection and replacement will automatically favour co-adapted rule-sets, crossover can be a major cause of disruption of cooperating collections of rules since the operator is blind to such associations between rules. An elegant solution to both the problems of coarse-grained credit assignment and the disruptive effects of crossover in Pittsburgh systems is proposed by Grefenstette [5] using hierarchical credit assignment. With this method, credit is assigned to individual rules as well as to complete rule-sets. Prior to crossover, the genome encoding the rule-set is ordered so that high strength rules occupy neighbouring loci on the genome. If the underlying assumption that co-adapted rules accrue similar strengths is valid, then crossover is less likely to disrupt these rule associations than if an unordered genome is employed. We shall return to the subject of ordering of rules on the genome prior to crossover in Section 5.

3. Pittsburgh-style fuzzy classifier systems

A number of researchers have investigated the automatic generation of fuzzy relations and fuzzy membership functions using evolutionary algorithms. In this section we present a summary of work applying the GA in the Pittsburgh style, where the complete fuzzy rule-set is the unit for selection and credit assignment. We have attempted to classify this work in decreasing order of constraint (i.e. increasing order of number of parameters to adapt) of the learning system. The categories we have used are:

1. Learning fuzzy set membership functions only, with a fixed set of rules set by hand.
2. Learning fuzzy rules only, with a fixed set of fuzzy membership functions set by hand.
3. Learning both fuzzy rules and fuzzy set membership functions but in stages i.e. first evolving good fuzzy rule-sets using fixed membership functions, then fine tuning membership functions using fixed good fuzzy rule-sets.
4. Learning both fuzzy rules and fuzzy set membership functions simultaneously (possibly with fine tuning of membership functions as a final stage).

3.1. Learning fuzzy membership functions with fixed fuzzy rules

Karr [10] applied GAs to fuzzy controller design by evolutionary adaptation of fuzzy membership functions for a fixed rule-set. This work demonstrated the success of the approach in generating both non-adaptive and adaptive fuzzy controllers for the four-input, one-output cart-pole balancing problem. Using three input fuzzy sets for each input and seven output fuzzy sets, a rule-base of eighty one rules is set by hand and the GA is then used to modify the endpoints of the fuzzy membership functions which are encoded on a binary string. Fitness of each individual is evaluated by running a number of simulations with different initial conditions. Karr's system was able to discover membership functions which controlled the cart-pole system significantly better than membership functions designed by hand. In addition, using a micro-GA, the author reports an adaptive fuzzy controller which was able to successfully cope with changing cart mass by genetically adapting fuzzy membership functions in real-time.

3.2. Learning fuzzy rules with fixed fuzzy membership functions

Thrift [28] describes the design of a two-input, one-output fuzzy controller for centring a cart on a frictionless one-dimensional track. Fixed, triangular input sets Negative-Medium (NM), Negative-Small (NS), Zero (Z), Positive-Small (PS) and Positive-medium (PM) are set by hand for each input variable. Individual control strategies are represented as 5×5 tables with each table entry encoding an output fuzzy set taken from {NM, NS, Z, PS, PM, _} (the _ symbol indicates no fuzzy set entry at this position) for each of the 25 combinations of input fuzzy sets. An individual control strategy's fitness is determined by running simulations of the cart. The GA mutation operator changes a fuzzy code (encoding an output fuzzy set membership function) either up or down a level, or to the blank code. Using a population size of 31, Thrift's system was able to evolve a fuzzy control strategy after 100 generations which compares well with the optimal "bang–bang" control rule.

Pham and Karaboga [20] describe the use of genetic algorithms to optimise the relation matrix of a fuzzy logic controller. This system learns fuzzy rules and output membership functions simultaneously using fixed input membership functions. Optimisation of the controller is carried out in two stages. In the first stage, different populations of controllers are independently evolved (using different initial random seeds) to produce "preliminary" designs. The second stage combines the best individuals from the first stage into a single population to which the GA is applied to evolve a "detailed" design. The authors use the fuzzy logic controller to control a simulated time-delayed second-order plant with one input (the controlled system error) and one control output. Seven input membership fuzzy membership functions (NB, NM, NS, ZE, PS, PM, PB), set by hand, are employed. The relation matrix consists of a 7×11 table (in general, $m \times n$). Each of the seven rows corresponds to one of the input membership functions, and each of the eleven columns corresponds to a quantised value of the output in the interval $[-5, +5]$, thus permitting general shape output membership functions. The experimental results presented convincingly demonstrate that good fuzzy controllers can be evolved using a simple GA, and that substantial improvement in controller performance can be gained using a two-stage evolution process.

3.3. *Learning fuzzy rules and membership functions in stages*

Kinzel et al. [11] describe an evolutionary approach to designing fuzzy controllers, and apply the technique to the cart-pole problem. The authors argue that learning fuzzy rules and membership functions simultaneously, while possible, is difficult due to the complex interactions between the two. Using this premise as a basis, Kinzel et al. describe a three stage task solving process, namely: (1) seed a "good" initial rule-base, (2) apply the GA to rules keeping membership functions fixed, then (3) apply the GA to membership functions. The authors argue against the use of bit-string encoded genomes, due to the destructive action of crossover, and use a non-binary table-based representation which preserves individual table entries (membership functions) under the action of crossover. In experiments, Kinzel et al.'s system discovers good fuzzy rule-sets for balancing the pole after 33 generations using a population size of 200, although it is reported that the membership tuning stage had relatively little effect on the goodness of rule-sets discovered by the GA.

3.4. *Learning fuzzy rules and membership functions simultaneously*

Lee and Takagi [14] employ the genetic algorithm to simultaneously optimise a variable size fuzzy rule base and fuzzy set membership functions of a Takagi–Sugeno [27] controller. This type of controller differs from a traditional fuzzy controller in that fuzzy values are involved only in the condition (antecedent) part of a rule; the consequent is a linear function of the inputs. The output of an n-rule Takagi-Sugeno controller is calculated as

$$\text{Output} = \sum_{i=1}^{n} \bar{w}_i f_i,$$

where

$$\bar{w}_i = \frac{w_i}{\sum_{i=1}^{n} w_i}, \quad f_i = \sigma_i + \sum_{j=1}^{m} q_{ij} X_j, \quad w_i = \prod_{j=1}^{m} A_j^i(X_j).$$

In these equations, A_j^i is the input fuzzy set membership function for the jth input variable in the ith rule, X the vector of inputs to the fuzzy controller, w_i is interpreted as the firing weight of the ith rule. Parameters σ and q_{ij} are rule-consequent parameters and determine the linear mapping of input values to output values. Lee and Takagi's system learns rule input membership functions and rule-consequent parameters simultaneously using bit strings to represent the left base, centre and right base of triangular membership functions. Bit-oriented mutation and crossover are used, and the system is applied with success to the pole-balancing problem. Reproductive fitness of chromosomes depends on phenotypic performance but is also reduced for chromosomes encoding large rule-sets as an attempt to prevent the formation of over-large rule-sets.

Cooper and Vidal [1] use a variable length genome to represent a fuzzy rule-set and accompanying membership functions, thus allowing the GA to automatically adapt the number of rules. The authors point out that domain-based representations which imply complete coverage of the input space and have been employed in most work carried out to date on GA-designed fuzzy controllers, cannot be expected to scale well to high-dimensional problems. To overcome this problem, they propose a "compact encoding scheme" whereby each rule of the genome string encodes (using binary encoding) the centres and widths of membership functions pertaining to that rule. Using variable-size rule-sets (i.e. variable-length genomes) in conjunction with rule creation and deletion operators, allows the GA to evolve economical rule-sets which do not include superfluous or unnecessary rules. Furthermore, the representation permits particular rule conditions to be ignored, allowing the evolution of generalised rules in cases where the appropriate action depends on a small subset of the input variables. Prior to genetic crossover, rules on parents selected for recombination are ordered according to similarity in input variable centres. Cooper and Vidal apply their system to the jointed cart-pole problem and obtain good results, with the controller successfully maintaining the cart-pole system after around 30 generations using a population size of 200 using randomly assigned initial rule-sets.

Liska and Melsheimer [15] use a GA for simultaneously discovering fuzzy rules and membership functions, with a final stage of fine-tuning membership functions using conjugate gradient descent. As with Cooper and Vidal's system, the representation used by Liska and Melsheimer allows both active conditions within a rule and the number of rules within a rule-set to vary dynamically during genetic search. A further similarity between the two systems, and the Pittsburgh-style fuzzy classifier system presented later in this article, is the use of ordering according to the centres of membership functions on the genome prior to GA crossover. However, an important difference between Liska and Melsheimer's system and the other two is that global sets of input and output membership function centres and widths used by all rules, are encoded on separate substrings of a composite genome which also includes a substring representing rules using these membership functions. In Cooper and Vidal's system and the system presented later, membership functions used in individual rules are encoded as part of those rules, rather than globally, introducing additional degrees of freedom for the learning system.

Liska and Melsheimer apply their system to learning a dynamic model of plant using known input–output data. After the GA approaches convergence, conjugate gradient descent is employed to further improve good solutions by fine-tuning membership function parameters. The results obtained compare well with those achieved using a 3-layer feedforward neural network, and in addition, as the authors state, the solutions found provide linguistic descriptions of their operation.

4. Michigan-style fuzzy classifier systems

In this section we summarise previous work which has demonstrated the potential of Michigan-style fuzzy classifier systems. Although the representation schemes and credit apportionment algorithms are different in their detail, all the systems outlined in this section reward or penalize individual rules by altering the rule strengths. Also, all the systems carry out genetic operations (selection, replacement, recombination and mutation) at the level of the individual rule.

4.1. Learning fuzzy rules with fixed fuzzy membership functions

Valenzuela-Rendón gives the first description of a fuzzy classifier system in [29]. Closely modelled on the discrete-valued Michigan-style classifier system, this system contains a fixed size rule-base of fuzzy classifiers, a fuzzy message list and employs credit assignment to individual classifiers. An inidivual rule is represented as a binary string that encodes the membership functions of the fuzzy sets involved in the rule. These fuzzy sets (Gaussian in shape) are fixed at the outset to cover the complete input/output spaces; the number of fuzzy sets is set by hand by the designer for the precision required.

The system therefore learns fuzzy relations between these fixed fuzzy sets. Valenzuela-Rendón applies the fuzzy classifier system, operating as a stimulus-response system, to one-input, one-output function identification. Credit is allocated to individual classifiers according to how closely each classifier predicts the correct output. The payoff distribution scheme is therefore, as the author states, not pure reinforcement learning. The fuzzy classifier system successfully learns straight line and parabola functions with relatively few learning cycles.

Valenzuela-Rendón's fuzzy classifier system discussed above and Parodi and Bonelli's [19] fuzzy classifier system described below in Section 4.2 are, in principle, capable of supporting internal message passing although reported results pertain to stimulus-response one-input, one-output function identification tasks. To learn complex behaviour which goes beyond simple stimulus-response, a classifier system may evolve internal reasoning processes based on chains of classifiers coupled by internal messages [2]. Valenzuela-Rendón [29] identifies the need to investigate fuzzy rule-chaining and a fuzzy version of the bucket-brigade credit assignment algorithm. However, Furuhashi et al. [3] argue that fuzzy rule-chaining gives rise to a problematic accumulation of fuzziness which, if unchecked, can seriously degrade system performance. To overcome this problem, Furuhashi et al. propose a method using multiple stimulus-response fuzzy classifiers operating in tandem. Message passing between individual fuzzy classifier systems is via "crisp" values in order to control excessive fuzziness. In their experiments, the authors describe a multiple fuzzy classifier system for controlling a simulated ship which attempts to steer towards a goal while avoiding other moving ships. The fuzzy controller is decomposed into three separate, communicating fuzzy classifier systems, one for recognising danger of collision, one for making decisions as to which direction to steer, and one for acting on these decisions by considering the dynamics of the ship. In each fuzzy classifier system, the number of rules and the membership functions are set by hand, and the GA is used to evolve a population of good fuzzy relations. Each fuzzy classifier system uses two inputs and one output.

In [17], Nakaoka et al. address the credit apportionment problem for large scale Michigan-style fuzzy classifier systems. In [17], the same ship-steering control problem as described above is tackled but this time with a single fuzzy classifier system with eight inputs and one output. However, as the authors state, coverage problems arise in moving to such high dimension input spaces because out of the set of initial randomly generated rules, only a small proportion will be matched during the course of a simulation. This problem is dealt with by using two different measures of credit for an individual rule – the first is a more traditional one, based on environmental payoff, while the second is based on an accumulation of the level of activation of rule during a simulation, irrespective of the "goodness" of that rule's suggested action. This dual fitness measure provides selection pressure when the GA is applied which favours both high-reward rules and well-matched rules. Low-reward but well-matched rules might well go on, under the action of crossover and mutation, to produce well-matched high-reward offspring. The latter outcome is encouraged by the use of a relatively high (0.5) mutation rate for the consequent part of rules. The authors report that using a 100-rule fuzzy classifier system, with fixed triangular membership functions, the controller consistently steered the ship to the goal while avoiding other moving ships after 69 generations of the GA.

4.2. Learning fuzzy rules and fuzzy membership functions simultaneously

Parodi and Bonelli [19] present a Michigan-style fuzzy classifier system which automatically learns fuzzy relations, fuzzy membership functions and classifier weights. A real-numbered representation for rules is used, with each rule encoding centres and widths of triangualar fuzzy set membership functions for each input and output variable. The GA crossover operator exchanges randomly selected fuzzy sets between pairs of rules. Mutation uses real number "creep" which modifies the centres and widths of membership functions within a rule. The classifier population consists of a fixed size list of such rules. Each rule has associated with it an "output weight" which is set equal to the

strength (fitness) of that rule. The rule strength therefore performs a dual function: firstly it forms the basis of selection and replacement for the GA and secondly it allows stronger rules to take a bigger part in decision making than weaker ones.

The authors demonstrate that the fuzzy classifier system is capable of successfully approximating one-input, one-output function mappings. As with Valenzuela-Rendón's system, a knowledge of the correct system output is used as the basis for credit assignment. Using this information, a "Measure of Goodness" is calculated for each rule which reflects not only the quality of the rule's conclusion fuzzy set, but also how well the rule's condition intervals are set for a particular input.

5. A Pittsburgh-style fuzzy classifier system (P-FCS1)

In this section, we describe in detail a Pittsburgh-style fuzzy classifier system called P-FCS1 and justify the choice of representation and genetic operators used. In summary, the main features of P-FCS1 are:

- it is based on the Pittsburgh classifier system model;
- it is rule-based as opposed to domain-based;
- the number of rules in each rule-set is allowed to vary, under the action of crossover, rule creation and deletion operators, and a cover operator;
- the system learns both fuzzy rules and fuzzy set membership functions;
- rule and fuzzy set membership encodings are real-numbered rather than using bit strings;
- a crossover operator is employed which respects the fact that genes representing rules with overlapping fuzzy sets are epistatically linked.

5.1. Rule-set encoding on genome

We employ the same rule representation used by Parodi and Bonelli [19] in their Michigan-style fuzzy classifier system, with each rule, $\boldsymbol{R}_k$, for an n-input, m-output system, expressed as:

$$\boldsymbol{R}_k\text{:}(x_{c1k}, x_{w1k}); \ldots (x_{cnk}, x_{wnk}) \Rightarrow (y_{c1k}, y_{w1k}); \ldots (y_{cmk}, y_{wmk})$$

The bracketed terms represent the centres and widths of fuzzy set membership functions over the range of input and output variables. The condition part of each rule comprises one fuzzy set membership function for each input variable and the action part comprises one fuzzy set membership function for each output variable. The genome representing a complete rule-set is a variable length concatenated string of such fuzzy rules. This representation allows genetic operators to work on both fuzzy rule-sets and membership functions. In addition since fuzzy set membership functions are encoded together with each rule (as opposed to using a global collection of fuzzy sets used by all rules), rules are permitted to evolve with different degrees of vagueness in the fuzzy sets they relate.

5.2. Choice of GA recombination operators

In this section we introduce a new crossover operator for use in Pittsburgh-style fuzzy classifier systems. As background to the motivation behind the employment of a modified crossover operator, we first restate Holland's schema theorem and the building block hypothesis for GAs, and outline the operators used by Smith in the LS-1 discrete valued learning classifier system.

5.2.1. Holland's schema theorem, the building-block hypothesis and LS-1's genetic operators

Holland's schema theorem provides a theoretical underpinning to the workings of the genetic algorithm. The theorem places a bound on the expected growth or decay of the number of schemata contained within a generation as follows:

$$m(h, t+1) \geqslant m(h, t)\frac{f(h)}{\bar{f}}\left[1 - p_c\frac{\delta(h)}{l-1} - p_m o(h)\right],$$

where $m(h, t)$ is the number of schemata h contained within a population at generation t, $f(h)$ is the average fitness of strings containing the schema h in the population, $\bar{f}$ is the average fitness of the population, l is the string length, p_c is the probability of crossover, $\delta(h)$ is the defining length of the schema h, $o(h)$ is the order of the schemata and p_m is the probability of mutation.

From this starting point, most GA researchers view good schemata as basic building blocks whose juxtaposition under action of GA operators produces good problem solutions. Goldberg [4] points out that three conditions are necessary for a schema to be a successful building block for the GA:

1. It should have above average fitness.
2. It should be low order (to survive mutation).
3. It should have a short length (to survive crossover).

As Smith states in his Ph.D. Thesis [25], there are two implicit assumptions in the schema theorem and the hyperplane analysis which supports it, namely, position dependence in the structures being manipulated, and fixed structure length. In a Pittsburgh classifier system, the external behaviour of a particular classifier set is independent of the order of individual rules within the encoded genome. However, the order of rules in classifier sets undergoing recombination is crucially important since rules which cooperate to good effect and which are far apart in the genome (effectively a long high-fitness schema) are likely to be split apart by crossover. In addition, in Smith's LS-1 system and the Pittsburgh-style fuzzy classifier system we describe here, rule-set length is allowed to vary, implying variable length genomes.

To address these issues, Smith designed modified genetic operators for dealing with position-independent variable length genomes, and performed a detailed analysis that demonstrates that these operators are capable of finding and exploiting good combinations of classifiers. The LS-1 crossover operator [25] aligns two parent structures randomly, then exchanges attributes on either side of a randomly chosen crosspoint to yield child structures. An inversion operator provides the system with an ability to cluster good combinations of rules in neighbouring loci on the genome, thus forming tight building blocks which are more resistant to being disrupted by crossover. A similar approach is expounded by Goldberg et al. [4] in their Messy GA.

5.2.2. Genetic operators used in P-FCS1

In moving from the discrete to the fuzzy case, a further level of interaction between rules encoded on the genome is introduced. Any two or more rules whose input fuzzy set membership functions overlap are epistatically linked on the genome since the crisp output value over the range of overlap of the inputs is determined by the combined action of all matched rules. Indeed, the identification of arbitrary input/output functions using fixed shape membership functions relies on this overlap. A crossover operator which preserves rather than destroying these linkages is likely to be a good one. In the initial implementation of P-FCS1 we take this observation to its extreme conclusion and order rules on the genome according to input membership function centres. A detailed description of the new crossover operator, first for single-input rule-sets and then for multiple-input rule-sets follows.

For single-input rule-sets, the rules are sorted according to the centres of their input membership functions prior to crossover. This introduces an association between a rule's input membership function centre and its position on the genome. In the case of one-point crossover, a random number is selected within the range of the input variable (we assume this is known) with uniform probability density. This number forms the crosspoint. For example, consider two single-input rule-sets each containing five rules with the following centres of input membership functions:

Parent 1: 0.10 0.30 0.40 0.90 1.50

Parent 2: 0.00 0.05 0.15 0.70 1.00

Suppose the number 0.35 (in the input space) was randomly chosen as the basis for crossover. The resulting offspring produced by applying the crossover operator would be:

Child 1: 0.10 0.30 0.70 1.00

Child 2: 0.00 0.05 0.15 0.40 0.9 1.50

In the case of n-dimensional input spaces, a crosspoint vector $C_i, i \in \{1, \ldots, n\}$ is selected. After crossover, Child 1 contains rules from Parent 1 such that $\forall i$, $x_{cik} < C_i$, and rules from Parent 2 such that $\forall i$, $x_{cik} > C_i$. Child 2 contains the remaining rules from both rule-sets. Care must be taken in the probability distribution used in selecting the components of the n-dimensional crosspoint. For example,

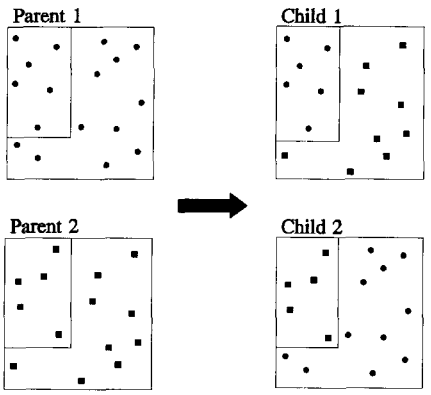

Fig. 1. Operation of one-point ordered crossover operator for 2-inputs.

choosing each crosspoint component independently using a uniform probability density function between some known limits is not suitable when $n > 1$ since the average "hypervolume" containing rules being crossed over decreases as dimensionality increases. In this initial implementation of the crossover operator, we choose a random number, R_c in the range [0, 1] to represent the normalised "hypervolume" of rules being crossed over. The crosspoint vector is calculated as

$$C_i = MIN_i + (MAX_i - MIN_i).(R_c)^{1/n},$$

where $[MIN_i, MAX_i]$ is the range of the input variable x_i. This one-point ordered crossover operator with two system inputs is depicted in Fig. 1. In this figure, the centres of fuzzy rule membership functions are denoted as filled circles (Parent 1) and filled square (Parent 2) in the plane of the two input variables. The right-hand side of the diagram shows the rules inherited by each of the two offspring (Child 1 and Child 2) from each of the parents for the chosen crosspoint.

The two-point version of the crossover operator involves the generation of two crosspoints C_{1i} and C_{2i} as follows:

$$C_{1i} = MIN_i + (MAX_i - MIN_i).(R_{1c})^{1/n}$$

$$C_{2i} = C_{1i} + (MAX_i - MIN_i).(R_{2c})^{1/n}$$

where R_{1c} and R_{2c} are selected randomly in the range [0, 1] with uniform probability density. After crossover, Child 1 contains rules from Parent 1 such that

$$\forall_i, \quad ((x_{cik} > C_{1i})\, AND\, (x_{cik} < C_{2i})) \quad or$$

$$((x_{cik} + MAX_i - MIN_i) < C_{2i})$$

together with rules from Parent 2 which do not satisfy this condition. Child 2 contains the remaining rules from both rule-sets.

The mutation operator used in P-FCS1 is considerably simpler and applies real-number creep to fuzzy set membership function centres and widths. Mutation is therefore used for fine tuning rather than for introducing radically different individuals into the population.

Table 1
Comparison of P-FCS1 with other Pittsburgh-style GA/Fuzzy approaches

Attribute	Reference							P-FCS1
	[1]	[10]	[11]	[14]	[15]	[20]	[28]	
Learns fuzzy sets	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Learns fuzzy rules	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Real (R), binary (B) or multi-valued (M) elements	B	B	M	B	B	B	M	R
Modified crossover	Yes	No	Yes	No	Yes	No	No	Yes
Variable number of rules	Yes	No	No	Yes	Yes	No	No	Yes
Global (G) or local (L) fuzzy sets	L	G	G	L	G	L	G	L

5.3. Comparison of P-FCS1 with other Pittsburgh-style GA/Fuzzy approaches

It is worthwhile comparing the representation and operators used in P-FCS1 with those of the Pittsburgh-style GA/Fuzzy approaches reviewed in Section 3. Table 1 summarises some of the key attributes of P-FCS1 and of each work described and referenced in Section 3. These attributes include the representation employed (binary, real-valued or discrete multi-valued genome elements; fixed or variable size rule-sets; global or local fuzzy sets) and the learning approach and GA operators used (learning fuzzy sets; learning fuzzy rules; using modified crossover operator).

Fixing fuzzy sets and learning rules [28] or fixing rules and learning fuzzy sets [10] can only be done when expert or heuristic knowledge is at hand. Of course when this knowledge is available from experts or can be generated by engineers, this is the wisest approach. However, for control of complex systems with unknown characteristics and where no training data are available, it is often not possible to decide on a simultaneously effective and efficient partitioning of the input/output spaces. Ascertaining fuzzy set membership functions is therefore difficult in such situations. Homogeneous partitioning of the input space when the input/output mapping varies in complexity within the space is inefficient and does not scale to high-dimensional spaces. P-FCS1 attempts to overcome this problem by allowing the GA to evolve fuzzy sets and rules simultaneously. While this feature offers a more powerful representation, it increases the size of the search space in that there are more parameters to adapt. This makes learning potentially more difficult.

In much previous GA/Fuzzy work [1, 10, 14, 15] a binary representation is employed. The original GA and associated schema analysis rely on a binary string representation. However, an extensive body of evidence is accumulating which suggests that the floating-point representation is often faster, more consistent from run to run, and provides higher precision. At the same time, the floating-point representation permits special operators and is intuitively closer to the problem space of a fuzzy controller.

P-FCS1 and the systems described in [1, 14, 15] employ variable-length genomes encoding a variable number of fuzzy rules. By contrast, systems using a fixed number of rules require at least some knowledge of the underlying problem complexity which may not be known a priori. Allowing the number of rules in the rule-base to vary, under the action of suitably modified versions of genetic operators such as crossover and mutation, and other operators such as cover, rule creation and rule deletion, permits the learning system to automatically adapt the number of rules according to the complexity of the problem.

P-FCS1 encodes fuzzy set membership functions locally within individual rules as opposed to using a global collection of fuzzy sets used by all rules. This approach is also proposed in [1, 14, 20]. The advantage of this representation, once more, is expressive power for the learning of rules which possess their own specificity in terms of the fuzzy sets they relate. This is likely to be of benefit in tackling the "curse of dimensionality" when scaling to multi-dimensional systems. However, this advantage is at the cost of, once again, an increase of the search space for the GA. Whether or not the additional degree of freedom introduced by the use of local fuzzy sets complicates linguistic interpretation is open to debate. Certainly, rules no longer employ easily understood fuzzy sets such as "NEGATIVE-LARGE", but then it is a fairly simple task to inspect the centres and widths of learned fuzzy set membership functions pertaining to a particular rule and perform a linguistic interpretation of that rule's operation.

To summarise, P-FCS1 combines many of the best features of recent research applying the GA to learn fuzzy controllers in a new way, using an appropriate representation and novel genetic operators. As stated earlier, the simultaneous learning of fuzzy rules, localised fuzzy sets and the number of rules is a difficult task and the search space is potentially large due to the number of parameters to adapt. In the next section, we describe experimental results that demonstrate that the GA has the potential to rise to this challenge.

6. Experiments, results and interpretation

The main objectives of the experiments conducted with P-FCS1 were fourfold:

1. To investigate the performance of different crossover operators in P-FCS1.
2. To compare the performance of P-FCS1 with that of reported Michigan-style fuzzy classifiers systems.
3. To compare the performance of P-FCS1 with the Pittsburgh approach to learning fuzzy rule-sets using fixed membership functions.
4. To evaluate P-FCS1 in a complex distributed control problem.

To satisfy objectives 2 and 3 we chose two learning tasks. The first is one-input, one-output function identification, as tackled by Valenzuela-Rendón [29], and Parodi and Bonelli [19] using Michigan-style fuzzy classifier systems. The second learning task chosen is the cart-centring control problem addressed by Thrift [28], using a Pittsburgh-style GA designed controller. For both tasks, detailed experimental results have been reported which facilitate comparison. For each task, we experimented with a variety of crossover operators, thus accomplishing objective 1. To achieve the fourth objective, we applied P-FCS1 to distributed routing control in a packet-switched network. Experimental details and results are presented below, together with a discussion of results.

6.1. Function identification

In this experiment P-FCS1 was given the task of learning one-input, one-output function identification for a number of different functions. The functions chosen were:

F1: $y = \sin(20x^2)$,

F2: $y = x$,

F3: $y = 4(x - 0.5)^2$,

F4: $y = \sin(20x)$,

with x in the range [0, 1]. Function F1 was chosen because the local input–output mapping becomes more complex with increasing x and we were interested in how the fuzzy membership functions and rules evolved by P-FCS1 would cover the input and output spaces. Functions F2, F3, F4 were chosen since they were dealt with by Valenzuela-Rendón, and Parodi and Bonelli using Michigan-style fuzzy classifier systems.

6.1.1. Population initialisation

A fixed population size of 100 rule-sets was used. Each rule-set in the population was initialised to contain 20 random rules (although under the action of crossover the number of rules in a rule-set is allowed to vary). Input membership function centres were selected randomly with uniform probability density in the range $[-0.2, +1.2]$. Output membership function centres were selected randomly from the range $[-0.2, +1.2]$ for F2 and F3 and the range $[-1.4, +1.4]$ for F1 and F4. Extending the range of initial membership function centres beyond the range of the input and output spaces is used to improve the performance of the system at the extremes of these spaces. Initial membership function widths were selected randomly with uniform probability density in the range

$$\left[0, \frac{2(X_{\max} - X_{\min})}{N_{\text{init}}}\right],$$

where $[X_{\min}, X_{\max}]$ is the range of the input or output variable and N_{init} is the initial number of rules in each rule-set. The range was chosen to encourage adequate coverage of input and output spaces for each initial rule-set.

6.1.2. Evaluation function

Each rule-set was presented with 50 equally spaced input values in the range [0, 1]. Using the same input values for each rule-set simplifies the learning process by removing noise from the evaluation function. The rule-set fitness was calculated as the inverse of the mean square error of the predicted outputs.

6.1.3. Genetic algorithm details

A generational strategy was adopted in which two new rule-sets were produced by the GA at each generation. Selection for reproduction was rank-based with $s = 1.8$. The replacement strategy used

was to replace the two weakest members of the population with the newly generated rules. A mutation rate of 0.1 was selected for each offspring rule-set. The mutation operator picks a single rule at random from the rule-set. Next, either the centre or width of one membership function within that rule is multiplied by a random number in the range [0,9, 1.1]. Crossover operates at individual rule boundaries (i.e. complete rules are crossed over). The crossover rate was set to 0.8 and the following crossover operators were experimented with: straight one-point, straight two-point, ordered one-point, ordered two-point and Smith's crossover operator. In the latter case, an inversion operator was also applied with a probability of 0.5. The inversion operator selects two points at random within the rule-set genome (at rule boundaries) and inverts the order of those rules lying between the two chosen points.

6.1.4. Results and interpretation

For each crossover operator listed in the previous paragraph P-FCS1 was run 10 times, using different initial random seeds, to learn the function F1. Fig. 2 shows the results of this experiment. This graph plots the error of the best population member (averaged over 10 runs) against generation number. The error is calculated as:

$$\frac{1}{1-(-1)}\int_0^1 |y - y_c|\,dx,$$

where y is the output predicted by the fuzzy classifier system and y_c is the correct output. The results displayed in Fig. 2 seem to indicate that, for this simple one-input, one-output function identification task, the two-point version of the new crossover operator is able to learn good solutions faster and to produce better end solutions (given the run time allowed) than the other operators.

The best solution discovered by P-FCS1 is shown in Fig. 3, which plots predicted and correct output over the input range. The error of this solution, calculated as described above, is 2.3%. Fig. 4 depicts the rules contained in this solution rule-set. Each rule is represented by a "diamond" in the figure with the centre of each diamond set by the centres of the input and output membership functions of the corresponding rule. The width and height of each diamond are determined by the width of the input membership function and the width of the output membership function, respectively.

We were interested in why Smith's crossover operator performed so poorly in this simple one-input, one-output fuzzy case when its good

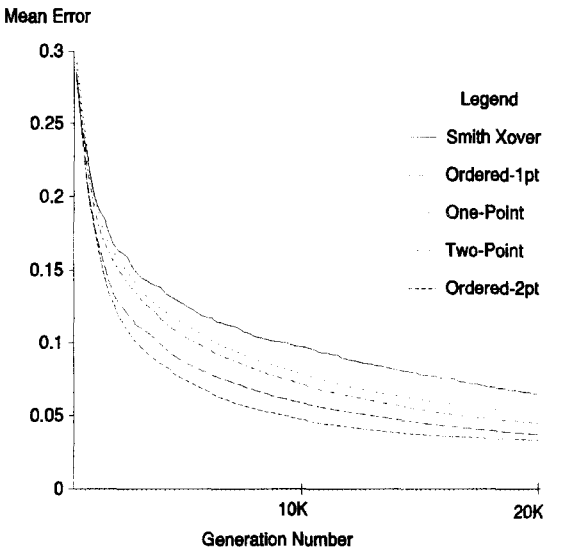

Fig. 2. Mean error for function F1 for various crossover operators.

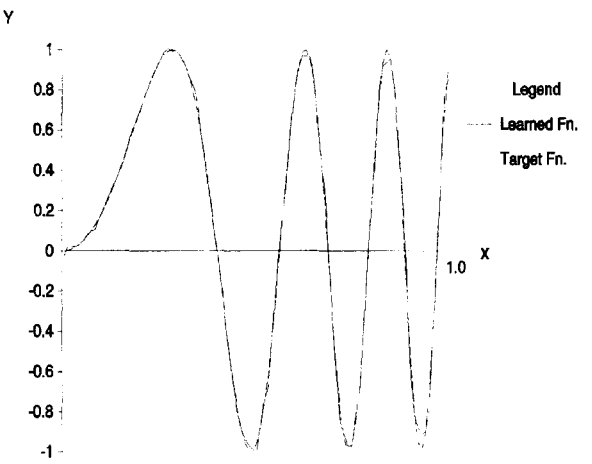

Fig. 3. Best solution found for function F1.

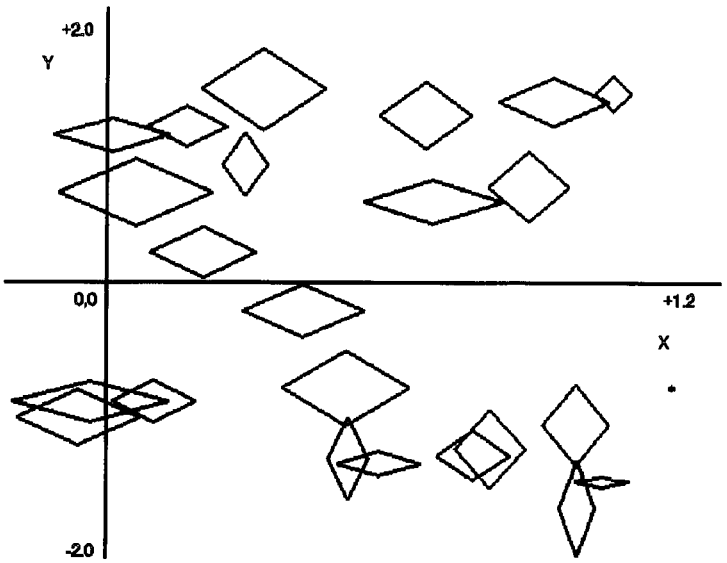

Fig. 4. Coverage of rules for a learned mapping for function F1.

performance for discrete-valued Pittsburgh-style classifier systems is undisputed. When rule-sets discovered using this operator were inspected they were found to contain multiple copies of identical rules. Smith points out this possibility in his thesis [28], and warns against using the genetic operators out of context in situations where the frequency of occurrence of a rule has an influence on performance. In LS-1 such redundancy does not affect the behaviour of the classifier system. However, in a fuzzy classifier system, clearly, duplication of rules does affect system behaviour. We have not experimented with a modified version of Smith's crossover operator which deletes duplicate rules but suspect this modification would improve performance. Also, the simple one-input, one-output function identification task does not involve any performance linkage between rules whose fuzzy set membership functions do not overlap. We envisage situations in more complex problems where such linkage would exist. In some environments, taking an action for a particular set of inputs might only be a "good" action if an appropriate action is taken for a different set of inputs and vice versa. Also, if fuzzy rule chaining is required, Smith's crossover operator is likely to be more successful. A hybrid crossover operator which combines the ordered crossover operator introduced here and Smith's crossover operator might well prove to be worthwhile investigating.

To compare the performance of P-FCS1 with reported results using Michigan-style classifier systems, we experimented with learning the functions F2, F3 and F4. In the case of F2 and F3, the initial number of rules of all population rule-sets was set to 10; for F4 this was set to 20. Table 2 compares the results obtained using P-FCS1 over ten runs using different initial random seeds with those obtained by Parodi and Bonelli [19] using a Michigan-style fuzzy classifier system (M-FCS). For functions F2 and F3, the error is calculated as

$$\frac{1}{1-0}\int_0^1 |y - y_c|\,dx$$

where y is the predicted output and y_c is the correct output. In the case of P-FCS1 we show the range of the number of rules in the best rule-sets which were generated in the ten independent runs, and the mean and best error achieved by the system over these ten runs.

The results shown in Table 2 indicate that a Pittsburgh-style classifier system using reinforcement learning can be at least as good as

Table 2
Performance comparison of P-FCS1 and a Michigan-style FCS [19]

System	Function	Number of rules	Error	Number of cycles
M-FCS	F2	500 (fixed)	1.19%	85 800
P-FCS1	F2	6–11 (variable)	Mean 0.96% Best 0.15%	80 000
M-FCS	F3	100 (fixed)	3.48%	103 700
P-FCS1	F3	7–14 (variable)	Mean 2.50% Best 1.50%	100 000
M-FCS	F4	1000 (fixed)	4.67%	197 500
P-FCS1	F4	14–30 (variable)	Mean 4.55% Best 3.23%	200 000

a Michigan-style classifier system using supervised learning in attempting to solve this simple task. The granularity of rule coverage in the two systems is very different: P-FCS1 uses a large population of small but variable size rule-sets compared to Parodi and Bonelli's system which uses a single population comprising a large fixed size rule-set with attendant rule strengths. This raises a key question: how will the two systems scale up to higher dimensional spaces? The Michigan-style classifier system outlined by Parodi and Bonelli to learn function F4 uses 1000 rules for one-input, one-output function mapping; clearly there will be problems in scaling to higher dimensions. P-FCS1 uses fewer rules per rule-base but as a result the optimisation problem is likely to become much harder in high-dimensional spaces. Further work is required to resolve this issue.

6.2. Cart-centring control task

In this task, a cart with mass m moves on a one-dimensional frictionless track (see Fig. 5). The objective is, given an initial position and velocity, to move the cart to zero position and velocity in the minimum time. Inputs to the controller are position, x, and velocity, v. The controller output is the applied force, F. The cart is simulated by the following equations of motion:

$$x(t+\tau) = x(t) + \tau v(t),$$

$$v(t+\tau) = v(t) + \tau \frac{F(t)}{m}.$$

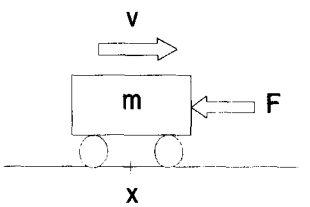

Fig. 5. The cart centering problem.

To enable comparison with Thrift's [28] experimental results, we chose $\tau = 0.02$ s and $m = 2.0$ kg. The maximum magnitude of the applied force was restricted to 2.0 N.

6.2.1. Population initialisation

A population of 100 rule-sets was used. Each rule-set in the population was initialised to contain 25 random rules. Input membership function centres (position and velocity) were selected randomly with uniform probability density in the range $[-5.0, +5.0]$. Output membership function centres (force) were chosen in the range $[-2.5, +2.5]$. Initial membership function widths were selected randomly with uniform probability density in the range

$$\left[0, \frac{2(X_{\max} - X_{\min})}{\sqrt{N_{\text{init}}}}\right],$$

where $[X_{\min}, X_{\max}]$ is the range of the input or output variable and N_{init} is the initial number of rules in each rule-set.

6.2.2. Evaluation function

Fitness evaluation is identical to that used by Thrift [28]. A simulation of the cart is run for 500 classifier steps (10 s with $\tau = 0.02$) with starting points $(x0, v0)$ selected from 25 equally spaced positions in the range $(-2.5, -2.5)$ to $(+2.5, +2.5)$. The fitness of each rule-set is measured as $(10 - T)$ where T is the average time for the cart to reach a position in state-space (x, v) such that $\max(|x|, |v|) < 0.5$. If, for a given starting position $(x0, v0)$ more than 10 s of simulated time are required, the fitness returned is zero. P-FCS1 is therefore, in this case, using pure reinforcement learning.

6.2.3. Genetic algorithm details

A generational strategy was adopted in which ten new rule-sets were produced by the GA at each generation. Selection for reproduction was rank-based with $s = 1.8$. The replacement strategy used was to replace the ten weakest members of the population with the newly generated rule-sets. The following three crossover operators were experimented with: ordered one-point, ordered two-point and Smith's crossover (with inversion).

6.2.4. Results and interpretation

The cart simulation was run ten times with different crossover operators. Fig. 6 shows how the average time to centre the cart for the best population member varied with generation number from 0 to 100 for different crossover operators. This diagram also shows the average time taken to centre the cart using an optimal bang-bang controller using the same 25 initial starting points [20]. The optimal bang-bang control rule is defined as follows. The force applied to the cart, $F(t)$, is chosen to be either $\boldsymbol{F}$ or $-\boldsymbol{F}$, where $\boldsymbol{F}$ is some positive constant. $F(t)$ is set equal to $\boldsymbol{F}$ if

$$\frac{v^2 \operatorname{sgn}(v)}{2|\boldsymbol{F}/m|} < -x$$

and $-\boldsymbol{F}$ otherwise. $\boldsymbol{F}$ is set to 2.0 N for comparison purposes. Fig. 7 shows a good input/output mapping learned by P-FCS1 using the ordered two-point crossover operator.

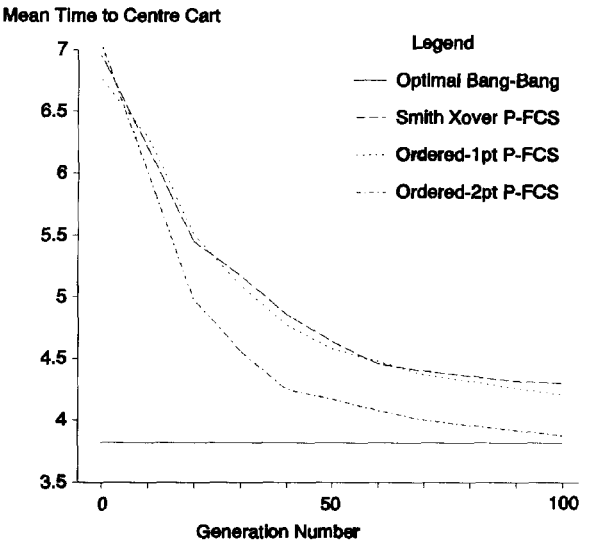

Fig. 6. Graph of mean time to centre the cart with $x, v < 0.5$ using Smith's, ordered 1-pt, and ordered 2-pt crossover operators.

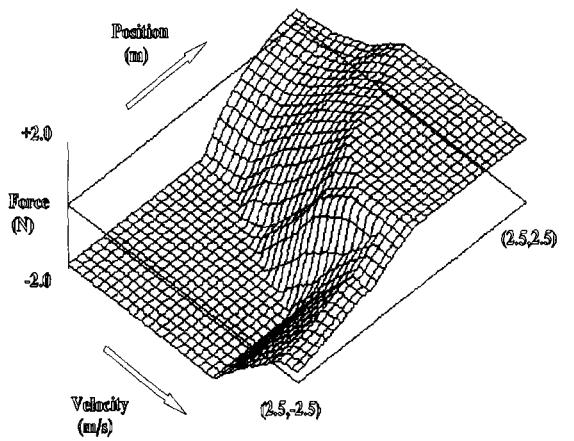

Fig. 7. Learned (position, velocity)- > force mapping for fuzzy controller in cart-centering problem.

To compare the results using P-FCS1 with Thrift's results [28], we evaluated the ten learned fuzzy controllers using the ordered two-point crossover by running the cart simulation over 100 runs with random starting points in $-2.5 < x < +2.5$, $-2.5 < v < +2.5$. The cart was centred successfully every time with a mean of 2.90 s (standard deviation 0.11 s). Thrift's system achieved 3.28 s,

and the optimal bang-bang controller achieved 2.86 s. The number of rule-set evaluations used for P-FCS1 was 1100 (100 initial rule-et evaluations plus 10 evaluations per new generation for 100 generations), compared with 3000 rule-set evaluations by Thrift (population size of 31, elitist replacement strategy, 100 generations). The results using P-FCS1 therefore compare well.

We repeated this simulation, using the same learned 10 rule-sets, but this time reduced the tolerance on x and v to 0.2 s. Six of the rule-sets centred the cart every time within a mean time of 4.20 s (standard deviation 0.31). However, the remaining four rule-sets did not manage to centre the cart in around half of the simulations. When these unsuccessful simulations were displayed, it was observed that the cart was quickly centred to within the range $-0.5 < x, v < +0.5$, on which the system was trained, but once inside this region of the state space the cart showed small amplitude oscillations around a position offset from the centre of the track. A controller evolved by Thrift's system generalised better to the new tolerance with a mean time to centre the cart of 4.8 s. The optimal bang-bang controller achieves a mean of 3.2 s.

In a final experiment, we ran P-FCS1 again 10 times using two-point ordered crossover but this time using the smaller tolerance $-0.2 < x, v < +0.2$ during the learning period. In 100 simulations using random starting points and velocities, the resultant 10 rule-sets successfully managed to centre the cart every single time within an average time of 3.50 s (standard deviation 0.27 s).

6.3. *A network routing control task*

In this task a fuzzy controller is required to perform distributed routing control [23] in a simulated 3-node datagram packet switched network (see Fig. 8). The network is fully connected with bidirectional full duplex links between each node pair. Packets requiring transmission over a particular link are queued using a first-come first-served discipline. Packets arrive from outside the network at network source node i ($i = A, B, C$), to be delivered to destination node j ($j = A, B, C$), $j \neq i$, at an average rate of λ_{ij}.

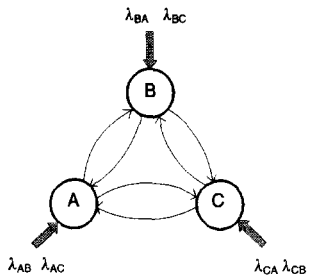

Fig. 8. Three-node packet switched network used in simulation.

A controller situated at each node must decide whether to route each packet directly to its destination or via an intermediate node. Controller decisions are based on packet delay measurements over the different paths. The goal is to minimise average global packet delay (i.e. the average delay between packet arrival at the source node and packet delivery to the destination node for all packets which arrive during the period of simulation irrespective of source and destination). Since the network topology chosen is symmetrical, an identical routing controller is placed at each node. The learning system is therefore required to determine a routing controller, copies of which are deployed at each switching node and operate in parallel, which minimises global packet delay.

Each routing controller is implemented as a variable size fuzzy classifier system with four inputs and two outputs. At each node the controller inputs are:

DelayLeftDirect: The measured packet delay from the source node for packets destined for the node to the left of the source node and which are routed directly.

DelayLeftIndirect: The measured packet delay from the source node for packets destined for the node to the left of the source node and which are routed indirectly (i.e. via the node to the right of the source node).

DelayRightDirect: The measured packet delay from the source node for packets destined for the node to the right of the source node and which are routed directly.

DelayRightIndirect: The measured packet delay from the source node for packets destined for the node to the right of the source node and which are

routed indirectly (i.e. via the node to the left of the source node).

Packet delays are measured at the destination node (each packet is time-stamped on arrival in the system) and averaged over the last N_{Measure} packets for each route taken for each source node. In the simulation, we assume this information is transmitted without delay to source nodes once the averages have been taken and transmission of control information does not consume network bandwidth. In a real network such information would be sent as control packets which would incur a finite delay and utilise network bandwidth. N_{Measure} is a parameter we vary by hand in the experiments described later and determines the granularity of measurements. Also, in a real network, a trade-off would have to be made in choosing the value of N_{Measure}. If too small a value is chosen, the network becomes swamped with control packets which compete with user data packets for use of the shared bandwidth. If too large a value is chosen, measurements become out of date and meaningless. The actual input variables presented to the fuzzy controller are the logarithm of measured delays muliplied by a constant scaling factor. This heuristic affords the controller greater sensitivity at low and medium delays. For simplicity, we neglect signal propagation delays.

At each node, the controller outputs are:

PLeftDirect: The probability that a packet arriving at the source node which is destined for the node to the left of the source node is routed directly. Hence $(1 - PLeftDirect)$ is the probability that the packet takes the indirect route.

PRightDirect: The probability that a packet arriving at the source node which is destined for the node to the right of the source node is routed directly. Hence $(1 - PRightDirect)$ is the probability that the packet takes the indirect route.

By dynamically adjusting local *PLeftDirect* and *PRightDirect* control outputs based on network delay measurements, the distributed assembly of controllers should attempt, in a cooperative fashion, to spread the network load to minimise global mean packet delay in response to changing traffic conditions in the network.

6.3.1. Population initialisation

A population of 80 rule-sets was used. Each rule-set in the population was initialised to contain 40 random rules although this was allowed to vary during learning up to a maximum of 80 rules under the action of a cover operator and crossover. Output membership function centres were chosen in the range [0, 1]. Initial membership function widths were selected with uniform probability density in the range

$$\left[0, \frac{2(X_{\max} - X_{\min})}{\sqrt{N_{\text{init}}}}\right]$$

where $[X_{\min}, X_{\max}]$ is the range of the input or output variable and N_{init} is the initial number of rules in the rule-set.

6.3.2. Evaluation function

Each network simulation is run for a simulation time of 500 s. The data rates of all network links are set to 10 000 bits per second. The variations in mean packet arrival rates with time for each source destination pair are shown in Fig. 9. These traffic patterns were chosen to exercise the dynamic capabilities of the routing controller in moving from relatively light network load, when direct routing is optimal, to heavy load when controllers must balance the offered load between direct and indirect network paths. In the simulation, packets arriving at an intermediate node are always forwarded to the destination node to avoid a "ping-pong" effect. The evaluation function for each rule-set returns the inverse of the mean measured packet delay for all packets delivered during the simulation.

Experiments were carried out with two types of traffic arrival process and packet size distribution. In the first, deterministic process, packet arrivals are equally spaced in time and packet sizes are fixed at 1000 bits. In the second, packet arrivals follow a Poisson distribution (with the same mean arrival rate as in the deterministic case), and packet sizes are exponentially distributed with the same mean size. In the latter case the system model becomes a distributed M/M/1 queuing system.

6.3.3. Genetic algorithm details

A generational strategy was used in which ten new rule-sets were produced by the GA at each

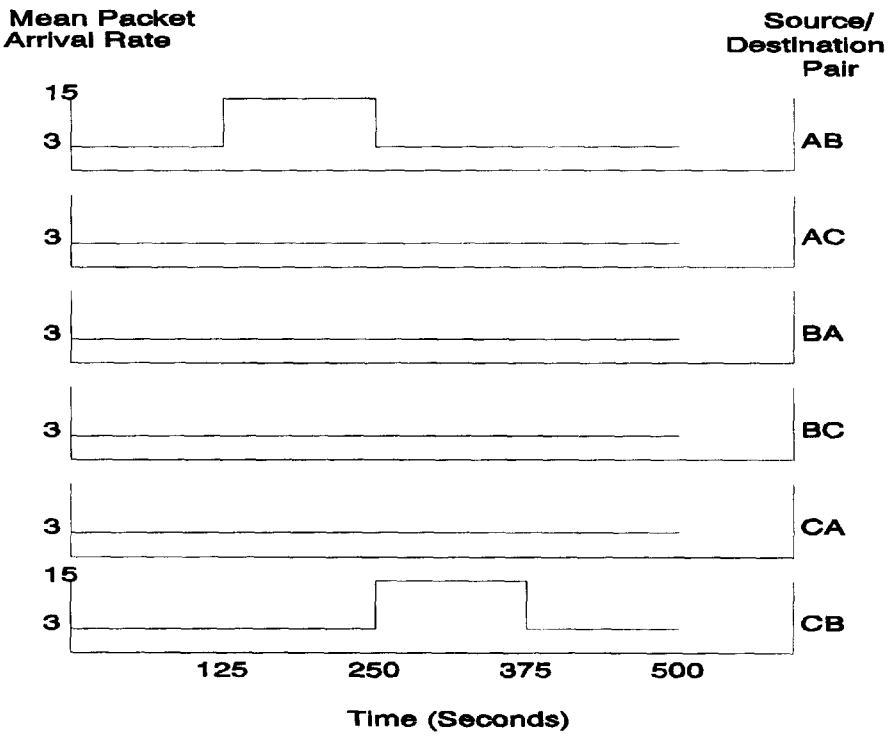

Fig. 9. Mean packet arrival rates for each source/destination pair versus time.

generation, using rank-based selection ($s = 1.8$) and replacing the ten weakest population members at each generation. The ordered two-point crossover operator was employed. Mutation rate and crossover rate were 0.1 and 0.8, respectively. In addition, a cover operator was implemented as follows: if a set of inputs is encountered which does not match any rules in the rule-base, a new rule is created with input fuzzy set membership function centres set equal to the unmatched input vector; output membership function centres are set randomly in the allowed range; and all membership function widths set as described in Section 6.3.1. a bove.

6.3.4. Results and interpretation

To evaluate the best controllers evolved by P-FCS1, we compared their performance with a shortest-path routing algorithm [23] which routes all packets along the route whose measured delay is least between a particular source/destination pair. A range of measurement intervals, N_{Measure} from 2 packets to 100 packets were used. Experiments were conducted using both deterministic and probabilistic packet arrival processes and packet size distributions. In each case, 10 independent runs of P-FCS1 were conducted with different initial random seeds. In addition, different initial random seeds were also used for each of the network simulations used in evaluating a particular individual. The latter introduces noise in the evaluation function and we were interested in whether the system could learn in the face of this potential difficulty. Each of the 10 learned fuzzy controllers using P-FCS1 were evaluated in 20 subsequent simulations and the result are presented in Table 3 where they are compared with the shortest path routing algorithm. This table shows mean packet delay over the complete simulation interval with standard deviations shown in brackets.

The results shown in Table 3 indicate that, when the measurement interval is small, the shortest-path algorithm outperforms the learned fuzzy

Table 3
Mean packet delay (in seconds) of fuzzy routing controllers learned by P-FCS1 compared with mean packet delay using shortest-path (SP) routing algorithm (standard deviations shown in brackets)

Measurement interval ($N_{Measure}$)	SP-routing (deterministic arrivals)	SP-routing (probabilistic arrivals)	Fuzzy control (deterministic arrivals)	Fuzzy control (probabilistic arrivals)
2	0.58 (0.09)	1.14 (0.32)	0.73 (0.12)	1.06 (0.42)
5	1.11 (0.22)	1.61 (0.29)	1.16 (0.16)	2.78 (0.41)
10	1.96 (0.23)	2.46 (0.42)	1.31 (0.32)	2.98 (0.48)
20	3.52 (0.39)	4.27 (0.75)	1.29 (0.28)	3.21 (0.45)
50	6.14 (0.42)	6.48 (1.29)	1.38 (0.60)	3.66 (0.51)
100	8.53 (3.00)	10.85 (1.49)	1.50 (0.38)	4.10 (0.65)

controllers, although not by that large a margin. As the measurement interval increases, the learned fuzzy controllers begin to outperform the shortest path algorithm significantly. As mentioned earlier, an important characteristic of a routing algorithm is that routing control information should not consume excessive network bandwidth. A value of $N_{Measure}$ greater than 20 is realistic for a real network and the results using a GA-derived fuzzy controller appear to be better than the simple shortest-path algorithm in this region of rate of feedback.

7. Conclusions and further work

We have presented an overview of research into GA based machine learning applied to fuzzy control and have described a fuzzy classifier system based on the Pittsburgh model. We conclude that the fusion of genetic algorithms and fuzzy logic, being a relatively new field of research, is still very much of an art. While this is so at the present time, it appears that the combination has the potential to become a powerful tool in the control of complex, multi-dimensional, non-linear systems. To achieve this potential, many problems have to be solved, perhaps the greatest being the problem of scalability to multi-dimensional spaces. Based on the work of others and our own experience, we suggest the following ways forward:

1. Rule-based approaches appear to be more likely to scale up than domain-based approaches, particularly if the latter use homogeneous partitioning of the input space.

2. Variable length rule-set representations (together with attendant operators for creating and deleting rules, perhaps implemented as cover operators or mutation operators) have the desirable property of being able to grow or shrink according to the complexity of the problem space.

3. The ability of the learning system to de-activate one or more conditions in a fuzzy rule can also effectively reduce the size of the search space by permitting evolution of general rules, if such rules are appropriate, and perhaps a fuzzy form of default hierarchy.

4. The genome representation and genetic operators should exploit the implicit cooperation between fuzzy rules with overlapping input fuzzy set membership functions.

5. Learning both fuzzy rules and fuzzy set membership functions is likely to be necessary for control of complex systems with unknown dynamics and with high-dimensional input/output spaces. Whether it is better to learn these in stages or simultaneously requires further investigation. A number of authors argue that the complex non-linear interactions between changes in fuzzy sets and fuzzy rules render the problem of simultaneously learning these extremely difficult. This is certainly true if an attempt is made to apply local gradient descent techniques as the basis of learning. Whether or not the genetic algorithm, with its global search characteristic, is powerful enough to overcome this problem remains an open question.

References

[1] M.G. Cooper and J.J. Vidal, Genetic design of fuzzy controllers: the cart and jointed pole problem, in: *Proc. Third IEEE Internat. Conf. on Fuzzy Systems*, IEEE Piscataway, NJ (1994) 1332–1337.

[2] S. Forrest and J.H. Miller, Emergent behaviour in classifier systems, in: S. Forrest, Ed., *Emergent Computation* (MIT Press, Cambridge, MA, 1991) 213–227.

[3] T. Furuhashi, K. Nakaoka and Y. Uchikawa, Suppression of excessive fuzziness using multiple fuzzy classifier systems, in: *Proc. Third IEEE Internat. Conf. on Fuzzy Systems*, IEEE Piscataway, NJ (1994) 411–414.

[4] D.E. Goldberg, B. Korb and K. Deb, Messy genetic algorithms: motivation, analysis and first results, *Complex Systems* **3** (1989) 493–530.

[5] J.J. Grefenstette, Multilevel credit assignment in a genetic learning system, in: *Genetic Algorithms and their Applications: Proc. Second Internat. Conf. on Genetic Algorithms* (Lawrence Erlbaum, Hillsdale, NJ 1987) 202–209.

[6] J.H. Holland, Properties of the bucket brigade algorithm, in: *Proc. First Internat. Conf. on Genetic Algorithms and their Applications.* (Lawrence Erlbaum, Hillsdale, NJ 1985) 1–7.

[7] J.H. Holland and J.S. Reitman, Cognitive systems based on adaptive algorithms, in: D.A. Waterman and F. Hayes-Roth, Eds., *Pattern-directed Inference Systems* (Academic Press, New York, 1978).

[8] J.J. Holmblad and L.P. Ostergaad, Control of a cement kiln by fuzzy logic, in: *Fuzzy Information and Decision Processes* (North-Holland, Amsterdam, 1982) 389–399.

[9] L.J. Huang and M. Tomizuka, A self-paced fuzzy tracking controller for two-dimensional motion control, *IEEE Trans. Systems Man Cybernet.* **20**(5) (1990) 1115–1124.

[10] C. Karr, Design of an adaptive fuzzy logic controller using a genetic algorithm, in: R. Belew and L. Booker, Eds., *Proc. Fourth Internat Conf. on Genetic Algorithms* (Morgan Kaufmann, Los Altos, CA, 1991) 450–457.

[11] J. Kinzel, F. Klawonn and R. Kruse, Modifications of genetic algorithms for designing and optimising fuzzy controllers, in: *Proc. First IEEE Internat. Conf. on Evolutionary Computation* (IEEE Piscataway, NJ, 1994) 28–33.

[12] B. Kosko, *Neural Networks and Fuzzy Systems* (Prentice-Hall, Englewood Cliffs, NJ, 1991).

[13] B. Kosko and S. Isaka, Fuzzy Logic, *Sci. Amer.* **269**(1) (July 1993) 62–67.

[14] M. Lee and H. Takagi, Integrating design stages of fuzzy systems using genetic algorithms, in: *Proc. Second IEEE Internat. Conf. on Fuzzy Systems* (IEEE, San Francisco, 1993) 612–617.

[15] J. Liska and S.S. Melsheimer, Complete design of fuzzy logic systems using genetic algorithms, in: D. Schaffer, Ed., *Proc. Third IEEE Internat. Conf. on Fuzzy Systems* (IEEE Piscataway, NJ, 1994) 1377–1382.

[16] E.H. Mamdani, Applications of fuzzy algorithms for control of a simple dynamic plant, in: *Proc. IEE* **121**(12) (1974) 1585–1588.

[17] K. Nakaoka, T. Furuhashi and Y. Uchikawa, A study on apportionment of credits of fuzzy classifier system for knowledge acquisition of large scale systems, in: *Proc. Third IEEE Internat. Conf. on Fuzzy Systems* (IEEE Piscataway, NJ, 1994) 1797–1800.

[18] A. Ollero and A.J. García-Cerezo, Direct digital control, auto-tuning and supervision using fuzzy logic, *Fuzzy Sets and Systems* **30** (1988) 135–153.

[19] A. Parodi and P. Bonelli, A new approach of fuzzy classifier systems, in: S. Forrest, Ed., *Proc. Fifth Internat. Conf. on Genetic Algorithms* (Morgan Kaufmann, Los Altos, CA, 1993) 223–230.

[20] D.T. Pham and D. Karaboga, Optimum design of fuzzy logic controllers using genetic algorithms, *J. Systems Engrg.* **1** (1991) 114–118.

[21] G. Roberts, Dynamic planning for classifier systems, in: S. Forrest, Ed., *Proc. Fifth Internat. Conf. on Genetic Algorithms* (Morgan Kaufmann, Los Altos, CA, 1989) 244–255.

[22] T. Saski and T. Akiyama, Traffic control process of expressway by fuzzy logic, *Fuzzy Sets and Systems* **26** (1988) 165–178.

[23] M. Schwartz, *Telecommunication Networks: Protocols, Modelling and Analysis* (Addison-Wesley, Reading, MA, 1987).

[24] L. Shu and J. Schaeffer, HCS: Adding hierarchies to classifier systems, in R.K. Belew and L.B. Booker, Eds., *Proc. Fourth Internat. Conf. on Genetic Algorithms* (Morgan Kaufmann, Los Altos, CA, 1991) 339–345.

[25] S.F. Smith, A learning system based on genetic adaptive algorithms, Ph.D. Thesis, University of Pittsburgh (1980).

[26] R. Sutton, Reinforcement learning architecture for animats, in: *From Animals to Animats: Proc. First Internat. Conf. on Simulation of Adaptive Behaviour* (MIT Press, Cambridge MA, 1991) 188–296.

[27] H. Takagi and M. Sugeno, Fuzzy identification of systems and its application to modelling and control, *IEEE Trans. Systems Man and Cybernet.* **15** (1985) 116–132.

[28] P. Thrift, Fuzzy logic synthesis with genetic algorithms, in: R. Belew and L. Booker, Eds., *Proc. Fourth Internat. Conf. on Genetic Algorithms* (Morgan Kaufmann, Los Altos, CA, 1991) 509–513.

[29] M. Valenzuela-Rendón, The fuzzy classifier system: a classifier system for continuously varying variables, in: R. Belew and L. Booker, Eds., *Proc. Fourth Internat. Conf. on Genetic Algorithms* (Morgan Kaufmann, Los Altos, CA, 1991) 346–353.

[30] S.W. Wilson, ZCS: A zeroth level classifier system, *Evolutionary Computation* **2**(1) (1994) 1–18.

[31] S.W. Wilson and D.E. Goldberg, A critical review of classifier systems, in: D. Schaffer, Ed., *Proc. Third Internat. Conf. on Genetic Algorithms* (Morgan Kaufmann, Los Altos, CA, 1989) 244–255.

[32] L. Zadeh, Outline of a new approach to the analysis of complex systems and design processes, in: *IEEE Trans. Systems Man Cybernet.* **SMC-3** (1973) 28–44.