Encouraging Complementary Fuzzy Rules within Iterative Rule Learning

Encouraging Complementary Fuzzy Rules within Iterative Rule Learning

Michelle Galea
School of Informatics
University of Edinburgh
m.galea@sms.ed.ac.uk

Qiang Shen
Department of Computer Science
University of Wales, Aberystwyth
qqs@aber.ac.uk

Vishal Singh
Larsen & Toubro Ltd, EmSyS
Bangalore, India
singhv@myw.ltindia.com

Abstract

Iterative rule learning is a common strategy for fuzzy rule induction using stochastic population-based algorithms (SPBAs) such as Ant Colony Optimisation and genetic algorithms. Several SPBAs are run in succession with the result of each being a rule added to an emerging final ruleset. Between SPBA runs, cases in the training set that are covered by the newly evolved rule are generally removed, so as to encourage the next SPBA to find good rules describing the remaining cases. This paper compares this IRL variant with another variant that instead weights cases between iterations. The latter approach results in improved classification accuracy and an increased robustness to parameter value changes.

1 Introduction

The use of stochastic population-based algorithms (SPBAs) for fuzzy rule induction has proved both popular and successful, with one of the most common strategies being that of Iterative Rule Learning (IRL), Figure 1 (see [1] for a review on the use of evolutionary algorithms such as genetic algorithms (GAs) [2] and genetic programming (GP) [3] for this purpose).

There are two main IRL variants—iteration by class where in each iteration rules describing a specific class are learnt (Figure 2), or independent of class where in each iteration good rule antecedents are first found and the class is determined afterwards (Figure 3). In either case, an SPBA is run several times in succession, with the result of each—the best fuzzy rule generated by the current algorithm—being considered a partial solution. Between runs of SPBAs, the training set is generally reduced by removing from it the cases that are covered by the newly evolved best rule. This is done so as to encourage the next algorithm to find good rules that describe the remaining cases in the training set.

Work on fuzzy rule induction using this strategy suggests a potential shortcoming. This arises out of the fact that fuzzy rules, due to their very nature, will match or cover all cases within a training set, but to varying degrees. Having a final set of complementary fuzzy rules is therefore essential to the inference process—i.e. it is necessary to avoid a situation where a case requiring classification is closely matched by two or more rules that have different conclusions. Some authors have stressed the necessity for additional mechanisms within the IRL strategy that encourage co-operation between the induced fuzzy rules. However, there is no direct evidence to support these claims, or to indicate that one mechanism may be better than another.

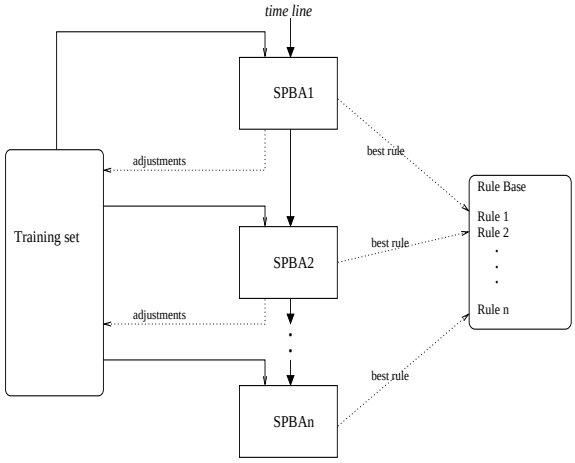

Figure 1: Iterative rule learning strategy

This paper therefore provides some clarification by directly comparing two different mechanisms from the literature for encouraging co-operative fuzzy rules. Furthermore, it introduces a relatively new approach to the discovery of individual fuzzy rules based on Ant Colony Optimisation.

Section 2 elaborates on the IRL strategy and variations on training set adjustments between iterations. Since the application of Ant Colony Optimisation to fuzzy rule induction is still a relatively unexplored area (compared to GAs and GP, for instance), this topic is introduced in Section 3. Section 4 then describes the implemented IRL algorithm with its two methods for

```
(1)    for each class
(2)        reinitialise training set
(3)        while class cases uncovered
(4)            run SPBA to generate rules
(5)            add best rule to final ruleset
(6)            remove covered class cases
(7)    output final ruleset
```

Figure 2: Iterative rule learning—iteration by class

```
(1)    while termination condition false
(2)        run SPBA to generate rules
(3)        add best rule to final ruleset
(4)        adjust training set as necessary
(5)    output final ruleset
```

Figure 3: Iterative rule learning—iteration independent of class

updating the training set. Section 5 presents an analysis of the results obtained, while Section 6 highlights the need for future work.

2 Encouraging Complementary Fuzzy Rules

Within IRL, the most common way for introducing some degree of co-operation between fuzzy rules in the final ruleset is by removing cases covered by the newly evolved rule between iterations (e.g. [4, 5]). This ensures that successive iterations are forced to find rules describing the remaining cases. There are a few known alternatives, however.

In [6] for instance, the authors implement an enhancement on their earlier work involving the *SLAVE* system [4]. *SLAVE* follows a class-dependent IRL approach, with the SPBA used being a GA. In their later work, in order to enhance co-operation between the induced rules during inference, the authors retain the same iterative approach but do not eliminate training cases between GA runs. Instead, they attach to each case various indicators that are used in the evaluation of a rule. These indicators are based on the rules already present in the final ruleset and are updated whenever a new rule is added. Hence, the rules in subsequent GA runs are evaluated taking into consideration a degree of interaction with already existing rules.

This new version of *SLAVE* results in improved accuracy and a reduction in both the number of rules in the final ruleset and the execution time. However, in this new version the authors also amend the fitness function, adding a term to encourage rules with fewer conditions in the rule antecedent. It is therefore difficult to judge whether the reported improvement to *SLAVE* is due to the new method for adjusting the training set between iterations, or to the new fitness function.

Another alternative is presented by Hoffmann [7]. An evolution strategy (ES) [8] algorithm is repeatedly invoked and each time identifies the fuzzy rule that best classifies the current distribution of training cases. Each training case has an attached weight and the author employs a mechanism to change the distribution of the cases from one iteration to the next. Examples that have been correctly classified by fuzzy rules generated in earlier iterations have a lower wieight, while those that have been misclassified have a higher weight. In each iteration the ES is guided to concentrate on generating rules that are best adapted at dealing with the previously misclassified cases.

In the following sections, removal of cases between IRL iterations is compared against a weighting method based on Hoffmann's work [7].

3 Ant Colony Optimisation

Ant Colony Optimisation (ACO) is a population-based heuristic motivated by the foraging strategies of real ants, who are generally capable of finding the shortest path between their nest and a food source. This is attributed to the fact that ants lay a chemical substance, called a pheromone, along the paths they take, and when presented with a choice between alternative paths, they tend to choose the one with the greatest amount of pheromone. Pheromone, however, evaporates so that over time the shortest path accrues more pheromone as it is traversed more quickly.

The appeal of ACO lies in several factors: it provides a simple effective mechanism for conducting global search by simultaneously constructing multiple solutions that investigate diverse areas of the solution space; a simplicity of implementation that requires minimum understanding of the problem domain; the problem-specific elements—such as the fitness function and heuristic—which may be readily borrowed from existing literature on rule induction; and, an explicit heuristic embedded in the solution construction mechanism that makes for easy insertion of domain knowledge.

In ACO, each artificial ant is considered a simple agent, communicating with other ants indirectly by effecting changes to a common environment. A high-level description of an ACO-based algorithm is given in Figure 4.

```
(1)    while termination condition false
(2)        each ant constructs a new solution
(3)        evaluate new solutions
(4)        update pheromone levels
(5)    output best solution
```

Figure 4: Basic ACO algorithm

Following is a brief introduction of the main elements necessary for an implementation of an ACO algorithm [9], set in the context of rule induction. The first four elements relate to Figure 4 line (2), the fifth relates to line (3), and the sixth to line (4):

1. An appropriate *problem representation* is required that allows an artificial ant to incrementally build a solution using a *probabilistic transition rule*. The problem is modelled as a search for a best path through a graph. In the context of rule induction a solution is a rule antecedent and each node of the graph represents a condition that may form part of it, such as OUTLOOK=Sunny, or OUTLOOK=Cloudy.

2. The *probabilistic transition rule* determines which node an ant should visit next. The transition rule is dependent on the *heuristic* value and the *pheromone* level associated with a node.

3. A local *heuristic* provides guidance to an ant in choosing the next node for the path (solution) it is building. Possible examples may be based on fuzzy subsethood values, or a measure of the vagueness in a fuzzy set.

4. A *constraint satisfaction method* forces the construction of feasible rules. For instance, if simple propositional IF-THEN rule antecedents are being constructed, then only one fuzzy linguistic term from each fuzzy variable may be selected.

5. A *fitness function* determines the quality of the solution built by an ant.

6. The *pheromone update rule* specifies how to modify the pheromone levels of each node in the graph. For instance, between iterations of an ACO algorithm, the nodes (conditions) contained in the best rule antecedent created get their pheromone levels increased.

A first attempt to apply ACO to fuzzy modelling is found in [10], though in this work the ACO algorithm is not used for constructing fuzzy rule antecedents, but for assigning rule conclusions. In a graphical representation of the problem, the fixed number of graph nodes are fuzzy rule antecedents found by a deterministic method from the training set. An ant traverses the graph, visiting each and every node and probabilistically assigns a rule conclusion to each.

In [11] a class-dependent IRL strategy is adopted with each iteration running an ACO algorithm to produce fuzzy rule antecedents. The problem graph for the ACO consists of nodes that represent conditions that may be selected by an ant when building its fuzzy rule.

```
(1)    while casesRemaining<maxCasesUncovered
(2)       while all termination criteria false
(3)          each ant constructs rule
(4)          evaluate all rules
(5)          update pheromone levels
(6)       add best rule to final ruleset
(7)       adjust training set
(8)    output final ruleset
```

Figure 5: Implemented algorithm

4 The Implemented System

This section details the system implemented, using class-independent IRL as the overall strategy, and ACOs as the rule discovery procedure. Figure 5 provides an overview. Note that lines (2)–(6) are equivalent to the ACO algorithm in Figure 4.

ACOs are run until the termination criterion in Figure 5 line (1) is met, with the outcome of each being a rule that is added to the final ruleset. This criterion is based on a user-defined parameter called `maxCasesUncovered` and when the number of 'remaining cases' in the training set falls below this value, the IRL algorithm will terminate. Since one of the mechanisms in this work for adjusting the training set between IRL iteration does *not* remove cases, this is clarified in section 4.3.

There are two different termination criteria for each individual ACO run, line (2). If a predefined number of successive iterations of the ACO each produce the same 'best' rule, then this is interpreted as the artificial ants having converged to an optimal path, and the ACO will terminate. If this does not occur then the ACO will terminate after have carried out a predefined maximum number of iterations.

4.1 Rule Construction

When creating a rule antecedent an ant traverses a problem graph where each node represents a term that may be added e.g OUTLOOK=Sunny. The choice of the next node to visit depends on both a heuristic value and the pheromone level associated with the node. It is made probabilistically but is biased towards terms that have relatively higher heuristic and pheromone values.

However, after selection and before a term is added to a rule antecedent, a check is made—this ensures that the resultant rule antecedent covers a minimum number of cases from the training set (set by a parameter called `minCasesPerRule`), and is a way of avoiding over-fitting to the training data. With fuzzy sets all fuzzy rules cover all training instances, but to varying degrees, and so what constitutes coverage of an instance by a fuzzy rule is clarified in section 4.2.1.

Once an ant has stopped building a rule antecedent

a rule consequent is chosen. This is done by assigning to the rule consequent the class label of the majority class among the cases covered by the built rule antecedent.

4.1.1 Heuristic

The heuristic used to guide ants when selecting terms is based on fuzzy entropy [13], and gives a measure of the vagueness or fuzziness of a fuzzy set:

$$\eta_j = -\frac{1}{N}\sum_{k=1}^{N} \mu_j(k)\ln(\mu_j(k)) + (1-\mu_j(k))\ln(1-\mu_j(k))$$

where η_j is the heuristic value of term j, N is the number of cases in the training set, and $\mu_j(k)$ is the membership value of case k for term j. The heuristic values of all terms are calculated once at the beginning of IRL, and do not change.

4.1.2 Pheromone Updating

At the start of an ACO run, Figure 5 line (2), all nodes in the graph have an equal amount of pheromone which is set to the inverse of the number of nodes. The pheromone level of individual nodes, however, changes between iterations of the ACO run, line (5). Towards the end of each iteration, line (4), rules created by all ants are evaluated and their fitness is determined (section 4.2). The terms that have been used in the best rule, say *R*, then get their pheromone levels increased:

$$\tau_j(t+1) = \tau_j(t) + \tau_j(t)\cdot f_R, \quad \forall j \in R$$

i.e. at time *t+1* each term *j* in rule *R* gets its pheromone level increased in proportion to the fitness, f_R, of the rule. A normalisation of pheromone levels of *all* terms further results in a decrease of the pheromone levels of terms *not* in *R*.

The pheromone updating process is a reinforcement mechanism—both positive and negative—for ants constructing new rules in successive iterations: terms that have had their pheromone levels increased have a higher chance of being selected, while those that have had their levels decreased have a lower chance.

4.1.3 Transition Rule

Ants select terms while constructing a rule antecedent according to a transition rule that is probabilistic but biased towards terms that have higher heuristic and pheromone levels. The probability that ant *m* selects term *j* when building its rule during iteration *t* is given by:

$$P_j^m(t) = \frac{[\eta_j]\cdot[\tau_j(t)]}{\sum_{i\in I_m}[\eta_i]\cdot[\tau_i(t)]}$$

where I_m is the set of terms that may still be considered for inclusion in the rule antecedent being built by ant *m*, i.e. *excluding* terms that are already present in the current partial rule antecedent, and terms that have already been considered but found to decrease coverage of the training set below the required number of instances (as set by `minCasesPerRule`).

The probabilistic nature of the rule is a way of introducing exploration into the search for a solution, in the expectation that a more optimal solution may well be found rather than by following a greedy approach and adhering strictly to terms with the highest values.

4.2 Rule Evaluation

The concept of fuzzy rule matching is used both during the construction of rule by an ant to ensure that the rule being built covers a minimum number of cases in the training set, and during the evaluation of a rule to determine how well the rule describes cases in the training set. What constitutes coverage or matching of a fuzzy instance by a fuzzy rule needs to be defined, as does the fitness function used to evaluate the constructed rules.

4.2.1 Fuzzy Rule Matching

A fuzzy rule is said to cover or match a fuzzy case if their degree of match is equal to or greater than a predefined value, here defined by a parameter called `fuzzyThreshold`.

When rule *R* is applied to a case *k* it is necessary to determine how well the attributes of *k* match the condition part of *R*, and how the class of *k* matches the conclusion of *R*. An example follows.

Consider a rule R that describes the conditions leading to a decision to do *Weightlifting* (this is from one of the datasets used in this work and is described further in section 5.1):

$$IF \text{ TEMPERATURE } is\ Mild\ AND \text{ WIND } is\ Windy$$
$$THEN\ Weightlifting$$

For the purpose of illustrating how a condition match may be determined, a more convenient representation of the rule is used: *R*=(0,0,0; 0,0,1; 0,0; 1,0; 0,0,1). This means that there are five attributes, the first four being condition attributes with three or two values (terms) in the domains, and the last representing the class attribute with three possible values (*Volleyball*, *Swimming* and *Weightlifting* respectively). Terms that are present in the rule are denoted by 1, others by 0. These rules may only classify instances into one class.

Consider now a fuzzy case *k*=(0.9,0.1,0.0; 0.0,0.3,0.7; 0.0,1.0; 0.9,1.0; 0.0,0.3,0.7). The representation is similar as for rule *R*, though the value for each term represents the degree of membership and lies in the range [0,1]. Note that the conclusion attribute values may be greater than 0 for more than one class, that an instance is considered to belong to

the class with the highest degree of membership, and in this case, the class is *Weightlifting*.

The degree of match between *R* and *k* is given by

$$mRule(R,k) = Min(mCond(R,k), mConc(R,k))$$

where the degree of condition match between *R* and *k* is

$$mCond(R,k) = Min_a(mAtt(R_a, k_a))$$

and the degree of conclusion match is

$$mConc(R,k) = Max_{1 \leq L}(Min(\mu_{Class_j}(R), \mu_{Class_j}(u)))$$

with *L* being the number of class labels, i.e. the number of terms for the class attribute. In the above definitions, $mAtt(R_a, k_a)$ measures the degree of match between an attribute *a* in *R* and the corresponding attribute in *k*:

$$mAtt(R_a, k_a) = \begin{cases} 1 : R_a \; empty \\ Max_j(Min(\mu_j(R_a), \mu_j(k_a))) : otherwise \end{cases}$$

where R_a *empty* indicates that no term from the domain of attribute *a* is present in rule *R*, and *j* is a specific term within the domain of attribute *a*. If the attribute is not represented at all in the rule, the interpretation is that it is irrelevant in making a particular classification.

From the rule and case examples above the attribute matches are: $mAtt(R_1, k_1) = 1.0$, $mAtt(R_2, k_2) = 0.7$, $mAtt(R_3, k_3) = 1.0$ and $mAtt(R_4, k_4) = 0.9$, with the condition match therefore $mCond(R,k) = 0.7$. The conclusion match is $mConc(R,k) = 0.7$ and the resulting rule match is $mRule(R,k) = 0.7$. If the `fuzzyThreshold` is set at 0.7 or below, then *R* is considered to cover *u*. If `fuzzyThreshold` is set above 0.7, then *R* is considered to not sufficiently match *k*.

4.2.2 Fitness Function

After a rule R has been constructed it needs to be evaluated. This is done by assessing how well R is at matching cases in the training set that have the same class as itself (f_{1_R}), and how well it is at avoiding matching those cases that belong to different classes (f_{2_R}):

$$f_{1\_R} = \frac{\sum_{k|c^k=c^R}(w^k \cdot mRule(R,k))}{\sum_{k|c^k=c^R}(w^k)}$$

$$f_{2\_R} = \frac{\sum_{k|c^k \neq c^R}(w^k \cdot mRule(R,k))}{\sum_k(w^k \cdot mRule(R,k))}$$

where k is a case in the training set, $k|c^k = c^R$ is a case that has the same class as rule R, w^k is the weight attached to the case (defined in 4.3), and $mRule(R,k)$ is the degree of match between R and k.

The resulting fitness for R is given by:

$$f_R = \begin{cases} 0 & f_{2\_R} > k_{max} \\ f_{1\_R} \cdot (1 - \frac{f_{2\_R}}{k_{max}}) & f_{2\_R} \leq k_{max} \end{cases}$$

where k_{max} represents the maximal rule inconsistency that can be tolerated as defined in [12].

4.3 Training Set Adjustment

The adjustment of the training set between IRL runs has been implemented in two different ways. It is the impact of these two different mechanisms on the resulting fuzzy rulesets that is being investigated in this work, as measured by how well they perform on classification tasks.

4.3.1 Removal of Training Examples

The first mechanism is the common one of removing cases from the training set that are covered by the newly evolved best rule from the previous ACO run. This encourages the next ACO to find rules that describe the remaining cases in the training set.

Note that when using this mechanism for adjusting the training set between iterations, for the purpose of evaluating rules created by an ACO run, a case that is still in the training set has an attached weight $w^k = 1$, while one that has been removed from the training set has weight $w^k = 0$.

With regards to the termination criterion for the IRL strategy, Figure 5 line (1), `casesRemaining` is the number of cases in the training set with $w^k = 1$.

4.3.2 Weighting of Training Examples

This second mechanism is based on [7] and changes the distribution of the training set between ACO runs.

Initially, all cases in the training set have the same weight value of 1. After the best rule R of an ACO run has been determined, the weights of all cases are recomputed. The new weight $w^k(t+1)$, of case k is given by:

$$w^k(t+1) = \begin{cases} w^k(t) & c^k \neq c^R \\ w^k(t) \cdot \beta^k & c^k = c^R \end{cases}$$

where $w^k(t)$ is the previous weight of case k, c^k is the class of k and c^R the class of rule R. β^k depends on the degree of match between R and k, and how good R is at *not* matching with cases that have different classes:

$$\beta^k = \left(\frac{f_{2\_R}}{1 - f_{2\_R}}\right)^{mRule(R,k)}$$

Cases in the training set that are correctly classified have their weights decreased, while those that are misclassified maintain their original weight. Note, however, that though no cases have actually been removed

Dataset	#Cases	#Attributes	#Classes
Saturday Morning	16	4	3
Image	210	19	7
Iris	150	4	3
Water Treatment	521	38	3

Table 2: The Datasets

Parameter	SM	Image	Iris	WT
`noIterations`	100	25	1000	25
`noAnts`	25	5	1	2
`fuzzyThreshold`	0.5	0.5	0.5	0.5
`weightThreshold`	0.5	0.5	0.5	0.5

Table 3: Parameter Values

from the training set, in order to give more direction to the next ACO for finding descriptive rules for cases that are continually being misclassified, only cases with an attached weight greater than a pre-defined value (as set by the `weightThreshold` parameter), are considered during rule construction.

This `weightThreshold` is also used as part of the termination criterion for the IRL strategy, Figure 5 line (1)—`casesRemaining` is the number of cases in the training set with w^k greater than `weightThreshold`.

5 Experiments and Discussion

5.1 The Datasets

The two different mechanisms for adjusting the training set between IRL iterations were tested on a fuzzified version of the Saturday Morning Problem (SM) [14], and on three other datasets obtained from the Machine Learning Repository of the University of California at Irvine (UCI) [15]. Dataset details are provided in Table 2. The Iris, Image and Water Treatment datasets (WT) were fuzzified using overlapping trapezoidal functions as used in [16].

The Water Treatment dataset contains the daily observations of 38 sensors monitoring the operation of an urban waste water treatment plant at various stages throughout the process, with the objective being to predict faults in the process. There are 13 possible classifications in the original dataset, but with most classifications being assigned to only a few records in the database. The 13 classifications have therefore been collapsed to just 3, as in [17]. The dataset had missing values for some attributes and these were replaced by the average value of the corresponding attribute.

5.2 System Parameters

The different parameter values used for the various datasets are listed in Table 3. It stipulates the number of iterations for an ACO, and the number of ants per iteration for an ACO. `fuzzyThreshold` is the parameter used during rule building and rule evaluation to determine whether a fuzzy rule sufficiently covers or matches a fuzzy instance. `weightThreshold` is applicable only when cases in the training set are weighted instead of removed outright. Little parameter tuning has been done and these values are based on a few exploratory runs of the system that indicated reasonable results would be obtained. Various values were used for each of the different datasets for the parameter `minCasesPerRule`, and these are presented as appropriate. `maxCasesUncovered` which is used by the termination criterion for IRL was kept at the value of `(minCasesPerRule-1)`, but this value could possibly be varied to better effect and is discussed in Section 5.3.2.

	% Predictive Accuracy	
	Case Removal	Case Weighting
Saturday Morning (2)	47.50 (4.4)	59.38 (3.3)
Image (5)	63.96 (4.0)	66.05 (4.1)
Iris (10)	92.27 (1.9)	93.00 (2.7)
Water Treatment (6)	66.44 (3.7)	70.84 (2.0)

Table 4: Predictive Accuracy

5.3 Results and Analyses

5.3.1 Predictive Accuracy

Table 4 presents the accuracies obtained using the two different mechanisms for adjusting the training set between IRL iterations. The figure in brackets next to each dataset name is the value used for `minCasesPerRule`.

For the Saturday Morning problem each result is obtained by averaging the results of ten leave-one-out cross-validations. Ten 10-fold cross-validations were performed on the other three datasets. The figure in brackets next to the accuracy value is the standard deviation of the 10 average predictive accuracies of the ten leave-one-out/10-fold cross-validations. It gives a measure of the variance in the accuracy arising from the cases in the different folds of the training set, and from the stochastic nature of the rule construction algorithm itself.

On these datasets it may be observed that IRL with case weighting generally performs better than IRL with removal of covered cases. A possible explanation may be that because all cases are still present during each iteration, at least to a certain degree, then they provide some feedback during rule evaluation as to how the rules currently being evaluated might interact with rules already in the final ruleset.

R1	IF PETALLENGTH is Low AND SEPALLENGTH is Low AND PETALWIDTH is Low AND SEPALWIDTH is High THEN CLASS is *Iris-setosa*
R2	IF PETALLENGTH is Low AND SEPALLENGTH is Low AND PETALWIDTH is Low AND SEPALWIDTH is Medium THEN CLASS is *Iris-setosa*
R3	IF PETALLENGTH is Medium AND SEPALLENGTH is Medium AND PETALWIDTH is Medium AND SEPALWIDTH is Medium THEN CLASS is *Iris-versicolor*
R4	IF PETALLENGTH is Medium AND SEPALLENGTH is Medium AND PETALWIDTH is Medium AND SEPALWIDTH is Low THEN CLASS is *Iris-versicolor*
R5	IF PETALLENGTH is Medium AND SEPALLENGTH is High AND PETALWIDTH is Medium THEN CLASS is *Iris-versicolor*
R6	IF PETALLENGTH is High AND SEPALLENGTH is High AND PETALWIDTH is High AND SEPALWIDTH is Medium THEN CLASS is *Iris-virginica*
R7	IF PETALLENGTH is High AND PETALWIDTH is High AND SEPALWIDTH is Low THEN CLASS is *Iris-virginica*

Table 1: Example Ruleset for the Iris Dataset

	Number of Rules	
	Case Removal	Case Weighting
Saturday Morning (2)	5.9	6.0
Image (5)	29.5	32.4
Iris (10)	9.0	9.3
Water Treatment (6)	30.0	37.0

Table 5: Ruleset Comprehensibility

5.3.2 Ruleset Comprehensibility

Table 5 lists the average number of rules resulting from the experiments discussed in the preceeding subsection.

In general, IRL with case weighting produces more rules, which might detract from the comprehensibility of the ruleset. This is because all cases are present in the training set for a longer period, i.e. until their attached weight value falls below the user-set `weightThreshold` value. The consequence is that it takes this IRL variant longer to meet the termination criterion `casesRemaining<maxCasesUncovered`.

However, there are several possibilities that may be explored in future work to resolve this issue. Some of these center around investigating several parameters in more detail, in order to understand their interdependencies, and the impact they may have on the ruleset in terms of accuracy and ruleset comprehensibility. These include:

- `weightThreshold` and `maxCasesUncovered`, which between them control the number of ACOs run;
- `fuzzyThreshold`, which controls how specific a rule is—the higher the value the fewer the conditions that may be added to the rule, resulting in more general rules that cover more cases in the training set; and
- `minCasesPerRule`, which also controls how specific or general a rule is—the higher the value the more cases a rule is required to cover from the training set during construction.

Another possibility is to conduct some form of post-processing on the ruleset produced. Consider the example ruleset provided in Table 1, one of several generated by the implemented algorithm. Note that rules R1 and R2, both describing the class *Iris-setosa*, differ only in the last condition of their respective antecedent. The same is true of rules R3 and R4 for the class *Iris-versicolor*. A post-processing step could consolidate these rules.

An alternative is to amend the rule discovery mechanism so that more expressive rules may be constructed. For instance, rules with internal disjunction between attribute values (e.g. SEPALWIDTH is Medium OR High), or rules that may include negated terms (e.g. SEPALWIDTH is NOT_Low). In [11] the ACO was amended so that such rules could be constructed, and the results suggest that a more expressive knowledge representation language has a beneficial impact on both the size of the ruleset and its predictive accuracy.

5.3.3 Parameter Value Robustness

Some experiments were run varying the parameter controlling the minimum number of cases that each rule must cover during construction—`minCasesPerRule`. It may be seen from Table 6 that, in general, IRL with case weighting provides not only increased accuracy but that there is also an indication of increased robustness to variations in the values of this parameter—the range of average predictive values for IRL with case weighting is lower than those for IRL with case removal.

This raises the prospect of future work investigating whether this variant of IRL also produces increased robustness to value changes of other parameters, such as the number of iterations of an ACO, or the number of ants within an ACO iteration.

Saturday Morning		
`minCasesPerRule`	Case Removal	Case Weighting
5	37.50	56.25
4	25.00	56.25
2	47.50	59.38
Range	*22.50*	*3.13*
Iris		
`minCasesPerRule`	Case Removal	Case Weighting
12	85.73	90.67
10	92.27	93.00
8	89.93	92.67
5	85.13	93.80
Range	*7.13*	*3.13*
Image		
`minCasesPerRule`	Case Removal	Case Weighting
10	35.95	41.47
7	58.15	61.47
5	63.96	66.05
Range	*28.01*	*24.58*

Table 6: Predictive Accuracy while varying `minCasesPerRule`

6 Conclusions

This preliminary work has only just begun to explore how a very common strategy for rule induction may be refined specifically for the induction of *fuzzy* rules. IRL with removal of covered cases between iterations works well for crisp rule induction, but it has often been implemented for fuzzy rule induction without taking into account the fundamental difference between crisp and fuzzy rules, and how they interact together during the inference process.

Other ways for adjusting the training set between iterations may also be considered. One simple adaptation of the basic removal of covered cases between iterations may be to change the rule selection process slightly. Instead of evaluating each rule individually and separately on the training set, it may be possible to first combine each rule to be evaluated with existing rules in the final ruleset, and then use this new ruleset to classify the training set. This means that any new rule added to the final ruleset has been selected based on how well it interacts with already existing rules in the final ruleset.

The early results presented here suggest that exploring and comparing mechanisms that encourage cooperation between fuzzy rules during the inference process is worth investigating. A related avenue of investigation is to explore whether similar results are possible even if different rule discovery mechanisms are used—in this work rules were constructed using ACO, but IRL for fuzzy rule induction has often been implemented using GAs, GP and ES, for instance. As well as providing useful knowledge in its own right, this future work will provide additional evidence to support or disprove the findings reported here.

References

[1] M. Galea, Q. Shen and J. Levine, "Evolutionary approaches to fuzzy modelling for classification" *The Knowledge Engineering Review* **19**(1), pp. 27–59, 2004.

[2] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press, 1975.

[3] J. R. Koza, *Genetic programming: On the programming of computers by means of natural selection*, A Bradford Book, The MIT Press, 1992.

[4] A. Gonzales, R. Perez and J. Verdegay, "Learning the structure of a fuzzy rule: a genetic approach" *Fuzzy Systems and Artificial Intelligence* **3**(1), pp. 57–70, 1994.

[5] W. Romao, A. Freitas and R. Pacheco, "A genetic algorithm for discovering interesting fuzzy prediction rules: Applications to science and technology data" *Proc. Genetic and Evolutionary Computation Conference* pp. 343–350, 2002.

[6] A. Gonzales and R. Perez, "SLAVE: A genetic learning system based on an iterative approach" *IEEE Transactions on Fuzzy Systems* **7**(2), pp. 176–191, 1999.

[7] F. Hoffmann, "Combining boosting and evolutionary algorithms for learning of fuzzy classification rules" *Fuzzy Sets and Systems* **141**(1), pp. 47–58, 2004.

[8] I. Rechenberg, *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien der Biologishen Evolution*, Stuttgart: Fromman-Holzboog Verlag, 1973.

[9] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, 1999.

[10] J. Casillas, O. Cordon, and F. Herrera, "Learning Fuzzy Rules using Ant Colony Optimization Algorithms" *Proc. 2nd International Workshop on Ant Algorithms* pp. 13–21, 2000.

[11] M. Galea and Q. Shen, "Fuzzy Rules from Ant-Inspired Computation" *Proc. IEEE International Conf. on Fuzzy Systems*, Budapest, July 2004.

[12] A. Gonzales and R. Perez, "Completeness and consistency conditions for learning fuzzy rules" *Fuzzy Sets and Systems* **96**, pp. 37–51, 1998.

[13] A. De Luca and S. Termin, "A definition of a nonprobabilistic entropy in the setting of fuzzy sets theory" *Information and Control* **20**(1), pp. 301–312, 1972.

[14] Y. Yuan and M. Shaw, "Induction of fuzzy decision trees" *Fuzzy Sets and Systems* **69**, pp. 125–139, 1995.

[15] C.L. Blake and C.J. Merz, *UCI Repository of Machine Learning Data* Deparatment of Computer Science, University of California, Irvine CA, 1998. `http://www.ics.uci.edu/~mlearn/MLRepositary.html`

[16] P. J. Bentley, "Evolving fuzzy detectives: An investigation into the evolution of fuzzy rules" late breaking papers at the *Genetic and Evolutionary Computation Conference* pp. 38–47, 1999.

[17] Q. Shen and A. Chouchoulas, "A rough-fuzzy approach for generating classification rules" *Pattern Recognition* **35**, pp. 2425–2438, 2002.