

Supervised and Unsupervised Data Mining with an Evolutionary Algorithm

Robert Cattral, Franz Oppacher, Dwight Deugo

Intelligent Systems Research Unit
School of Computer Science
Carleton University
Ottawa, On K1S 5B6
Canada

{rcattral,oppacher,deugo}@scs.carleton.ca

Abstract- This paper describes our current research with RAGA (Rule Acquisition with a Genetic Algorithm). RAGA is a genetic algorithm and genetic programming hybrid that is designed for the tasks of supervised and certain types of unsupervised data mining. Since its initial release we have improved its predictive accuracy and data coverage, as well as its ability to generate more scalable rule hierarchies. These enhancements and several experiments are described.

1 Introduction

Data mining is defined as extracting structured information, such as patterns and regularities, from databases. Also known as KDD, or *knowledge discovery in databases*, the process is important because it provides means for understanding data, including the generation of predictive rules.

RAGA [Cattral, Oppacher, Deugo 1999a] was developed as a *data mining* system that uses a genetic algorithm (GA) based engine to extract knowledge in the form of predictive rules. The system was designed for the tasks of both *supervised* and certain types of *unsupervised learning*.

1.1 Supervised versus Unsupervised Learning

Supervised learning, also referred to as *learning by example*, is a process where the system attempts to find concept descriptions for classes that are, together with pre-classified examples, supplied to it by a teacher. The usual characterization of unsupervised learning as learning without pre-classified examples conflates a variety of increasingly difficult learning tasks. These tasks range from detecting potentially useful regularities among the data couched in the provided description language to the discovery of concepts through conceptual clustering and constructive induction, and to the further discovery of empirical laws relating concepts constructed by the system.

The task of *unsupervised learning* is much more demanding because here the system is only directed to search the data for interesting associations, and attempts to group elements by postulating class descriptions for sufficiently many classes to cover all items in the database.

The supervised learning experiments described in this paper are for the purpose of data classification, while the unsupervised tests perform the task of association rule discovery.

1.2 Confidence and Support Factors

To determine the quality of a rule there are a combination of statistical and subjective factors to consider. These include the rule accuracy, coverage, how useful and how interesting the rule is within the given domain. Because the latter two features are subjective in nature there is no widely accepted method available for determining them, and for the same reason there is no common scale to rate or compare them. The measures for accuracy and coverage of rules are known in other areas of information theory as *confidence* and *support factors* [Berry, Linoff 1997].

The confidence factor is a measure of the rule accuracy. It is defined to be the percentage of times that the consequent is true given that the antecedent is true. If the consequent is false while the antecedent is true, then the confidence fails for the given rule. If the antecedent is not matched by the given data item, then this item does not contribute to the determination of the confidence of the rule.

The support factor is defined as the rule coverage, which is how often the rule is correctly applied within the entire dataset. This measure is calculated by dividing the number of elements that are correctly answered using the rule by the total number of elements in the set. The basic information conveyed is how often the rule is actually used, which is an important factor to consider when deciding its worth.

2 Representation of Data

The way that RAGA represents data is the primary difference between it and other GA based systems. In a standard GA a fixed length binary string represents each individual. A decoding function converts this string into the appropriate form when required. This representation is too restrictive for the predictive rules that are generated by RAGA.

The first problem is that all rules would have an equal fixed length, which considerably limits their structure. Although it is possible to specify an upper bound that is sufficient to hold an entire rule, there are many different sizes of rules within any given set. Accommodating this within a fixed length structure would require the use of NOP (no-operation) values, and will not be as efficient.

The next problem concerns the alphabet used by the GA. In a standard GA there are only two terminals used to represent the binary values of zero and one. Variations of GA systems, such as Genetic Programming (GP) [Koza 1990], use a much larger alphabet that is dependent on the problem being solved. In a typical GP system there are numerous terminals that represent variables within the system.

The type of data being sought by RAGA is an *If-Then rule*, which is of the form:

$$\text{If } X_1 \wedge X_2 \wedge \dots \wedge X_n \text{ Then } Y_1 \wedge Y_2 \wedge \dots \wedge Y_m$$

The symbols $X_1 \dots X_n$, and $Y_1 \dots Y_m$ each represent terms within the rule, where a term is a function that either indicates the existence of an attribute, or performs an operation on two or more variables. In classification tasks the value for M is always 1, while the value for N is potentially unbound.

Rules are represented within RAGA as dynamic structures that store the information required for both presentation and evaluation. RAGA is similar to GP in this respect because it has both a user-defined number of terminals, and the length of the rule is flexible.

A flexible set of terminals is required in RAGA because different datasets have a different number of attributes, and without accommodation for each of them it is not possible to fully explore the data. Furthermore, it is sometimes necessary to make use of derived values that are not part of the original data, but rather a function of one or more attributes.

3 Evolving a default hierarchy

A default hierarchy is a collection of rules that are executed in a particular order. When testing a particular data item against a hierarchy of rules, the rule at the top of the list is tried first. If its antecedent correctly matches the conditions in the element being tested, this top rule is used. If a rule does not apply, then the element is matched against the rule at the next lower level of the hierarchy. This continues until the element matches a rule or the bottom of the hierarchy is reached.

Rules that are incorrect by themselves can be *protected* by rules preceding them in the default hierarchy, and play a useful coverage-extending role, as in the following example:

```
If (num_sides = 4) ^ (length = width) then class = square
If (num_sides = 3) then class = triangle
If (num_sides > 2) ^ (num_sides < 5) then class = rectangle
```

If the last rule were used out of order, many instances would be improperly classified. In the current position it covers the remaining data items accurately.

The effects of the hierarchy, particularly with respect to an increased data coverage and decreased penalty for overfitting, are described in [Cattral, Oppacher, Deugo 1999b].

4 Enhanced Genetic Engine

The genetic engine used by RAGA is a hybrid of GA and GP, with several modifications and additions to the standard models. This section describes the details of the evolutionary functions, and highlights differences between RAGA and each of the standard approaches.

4.1 Plug-in Style Fitness Function

In order for a GA-based system to effectively search for solutions, an appropriate fitness function must be implemented. In a data mining system, the ultimate fitness function would act as a measure for how close the rule is to what is being sought. Because this measure is subjective, particularly in undirected mining tasks, there is no straightforward way to implement such a measure.

With only a limited amount of information available for each candidate rule, an effective fitness function has to combine as much of this data as possible. In RAGA, the raw fitness is determined by using the values for confidence and support.

$$\text{RawFitness} := 100 - \text{sqrt}[(\text{confidence} - \text{confidence target})^2 + (\text{support} - \text{supportTarget})^2 * \text{sfAdjustment}]$$

This equation converts the values for confidence and support into a single numerical value that increases as each factor approaches the respective target. Basically, the closer the values are to the user specification, the higher the raw fitness is for the rule.

Depending on the dataset and the type of search being performed there are several fitness plug-in type modifiers that can be used. These act to increase or decrease the raw fitness and help to determine the actual fitness.

The first optional adjustment is obvious in the equation, and can be applied to change the importance of the support factor with respect to the rule confidence. This extension is the last term in the equation, and is called the SF-Adjustment (support factor adjustment).

By default the SF-Adjustment is 1.0, which means that both confidence and support factors are weighed equally when calculating the raw fitness. Depending on the dataset and the domain it might be important that a higher confidence is preferred to a higher support. In these cases entering a decimal value greater than zero but less than one will lower the impact of the support factor. Conversely it is possible to raise the impact of the support factor by entering a multiplier values that is greater than one.

The next fitness plug-in is *Encourage wider rule coverage*, which awards a bonus to rules that are first in the hierarchy to successfully answer a particular data item. The goal is to reward rules based on the number of elements they correctly answer within the dataset, excluding those answered by other rules. Effectively, this is identical to creating *niches* for rules that address certain classes, and is advantageous for classification tasks because it maximizes data coverage.

The next plug-in option is *Limit expansion to good rules*, which is used to control the expansion of rules during the evolutionary process. When this option is not used, rules can grow in size without restriction using either crossover or mutation operations. Growth can occur in either the antecedent or the consequent, and is limited only by the maximum length as specified for each in the configuration options.

The problem with allowing rules to expand in size without restriction is that useless rules, which are labeled as such if they contribute nothing positive to the results, are also allowed to expand and become more complex. Even in the case of rules that have no confidence it is still possible that new terms will be added. This has the potential to impair the search in two ways, specifically an increased processing time as well as an increased number of bad rules.

When in use, this option guarantees that each rule must not expand unless it already has a positive confidence.

The goal of this fitness modifier is to encourage sound basic components within each rule. Because large rules will not exist unless they are built from valid components, processing time is not wasted on their evaluation. Also, because each rule has a solid base with potential for refinement, the system will produce smaller and more useful rules, which leads to a better hierarchy overall.

The final fitness adjustment plug-ins is *Reward shorter rules*. This can apply a bonus that is based solely on the total number of terms present in the antecedent and consequent.

The motivation behind using this option is to bias the system towards slightly shorter rules. Because of increased readability and ease of understanding, this is common in Data Mining applications. This is similar to a reward used by GP systems, where shorter programs are allotted credit for being *parsimonious*. Basically, the two rules or programs perform exactly the same within the current dataset, however the smaller one is presumed to be better.

The amount of bonus awarded is dependent on the actual length, as well as the *Reward constant*, which is an option associated with this plug-in. The reward constant is an additive value that is applied to the raw fitness in proportion to the desired length.

The reasons to bias the system towards smaller rules is the expectation that they will be easier to understand, as well as more efficient to process. This is often true because larger rules are more likely to contain redundant or unimportant information, and this can obscure the basic components that make the rule successful.

Complex rules are not only more difficult to read and analyze, but more importantly they can be unnecessarily specialized. For example, if a rule is simply a superset of terms as compared to another shorter rule, but the two rules answer exactly the same elements within the dataset, then the additional term(s) are not contributing useful information. Although it may not be significant with the current data (training set) it is possible that the rule is not general enough to handle future cases (testing set).

It is important to recognize that if the additional terms were useful in even a small way, there would be a change in the results and a shift in the fitness. Because this is not expressed there is little chance that additional specialization will be more effective in new cases. Without having any additional knowledge to help differentiate the two rules it is advisable to keep the more general (shorter) one.

4.2 Fitness Proportional Selection

For selection of rules that are copied from one generation into the next, RAGA uses *fitness proportional selection*, where the best rules are given a selection advantage. The dilemma with fitness proportion selection is that because the more fit individuals are constantly being selected, they are usually copied several times within a single generation. When this happens over several generations, a single individual with a high fitness tends to get preferred, and eventually dominates the entire population. This does not occur in RAGA because unlike standard GA, duplicate rules are not allowed to co-exist within the population.

After a single member has been selected for the next generation, there will be a probability that crossover will occur.

4.3 Crossover

RAGA performs a single-point crossover that splits and recombines rules, and also guarantees that the results are still valid. This is done by first choosing a random splitting point between terms for each of two distinct rules. After the splice point for the first rule is determined, the second is more restricted because it is forced to be on the same side of the equation. (ie: if the first rule is splitting within the antecedent, then the second rule must split within the antecedent). After the rules are broken, the halves are recombined with the other rule. The result of this operation is the creation of two new rules.

In RAGA, this group of four rules (two parents plus two children) is called a *family*. Exactly two of these rules, chosen according to the highest fitness, will be selected for the next generation. This differs from some GA implementations where crossover will indiscriminately select the child-rules and discard the parent-rules. In our experiments this was found to be quite beneficial because it protects the population during periods where the rates for crossover or mutation are increased.

One of the reasons for this difference is that standard GA systems do not consider the possibility that the child-rules are invalid. Because crossover operates at the term level, with the ability to change both size and structure of rules, it is capable of creating rules that are not allowed by the restrictions set by the user during the configuration stage. When an invalid rule is detected after a crossover operation, it is removed from the family immediately.

After the two family members are selected for the next generation, each will be copied pending the possibility of mutation.

4.4 Mutation

In RAGA there are two types of mutation, referred to as *micro-mutation* and *macro-mutation*. Micro-mutation is a single point change that occurs within the terms of terms of each rule. Macro-mutation does not affect the specifics of an individual term, but rather is used to add or remove terms from either side of the rule.

The goal of micro-mutation is to modify the terms in an attempt to add variation to the hierarchy. This type of mutation is capable of changing the term variables, constants, or the type of operation. This operation does not consider terms or rules themselves, and does not anticipate the effect of the change. If the result of the mutation is an invalid term or rule, then the entire rule is discarded immediately.

Macro-mutation is similar in that population members are modified and either moved into the next generation or discarded, however the goal is to experiment with the specialization and generalization of existing rules.

To specialize a rule is to restrict its application by specifying additional constraints. The purpose is to increase the accuracy by lowering the coverage. If it works correctly, then the coverage will be reduced only by elements that were being misclassified beforehand.

When a rule is generalized, constraints are removed in an attempt to increase coverage. If this works correctly, then the constraints are relaxed enough to answer more elements without the penalty of a reduced accuracy.

Both mutation types are used together during the evolutionary process, and can be controlled using varying probabilities.

Due to the fact that the genetic operations behave somewhat randomly it is often the case that rules are modified such that they become inefficient or invalid afterwards. RAGA handles these cases with a process named *Intergenerational Processing*, which uses a non-evolutionary approach to modify and replace rules.

4.5 Intergenerational Processing

Mechanisms of this type are not typically used in GA, however the idea is similar in some respects to a GP variant known as Typed-GP [Montana, 1995]. Typed-GP guarantees that evolved programs will not contain critical errors, thus saving processing time by eliminating worthless members from the population.

A popular argument against using this type of mechanism is that it amounts to *cheating* by allowing forces other than natural selection to influence the members. The first reason for taking this position is that in theory the selection pressure will eliminate worthless members without interference. Secondly, by eliminating individuals that are not selected against naturally, there is a risk that useful genetic material is being lost as a consequence. Regardless of these points, RAGA allows the user to specify one or more options that override the natural selection process.

The genetic engine in RAGA makes the use of this screening available because of the results from many tests that were performed during the software development stage. It was determined that by allowing the system to remove certain types of rules, not only was there an improvement in the time required to complete the search, but the rule sets were more efficient and performed better on average.

In order to prevent the loss of diversity expected with the non-evolutionary removal of rules, each deleted member is immediately replaced. This replacement is done through the use of rule modification or regeneration.

When a rule is deemed invalid the first step is to modify it such that it becomes valid. The technique used for this is the systematic deletion of terms from antecedent, consequent, or both. If this is not possible because of the rule structure or an excess of similar members in the population, a complete substitution will be made. If after replacing or modifying the rule the result is still invalid, or is a duplicate with another rule in the set, this technique is continually applied until all conditions are satisfied.

5 Problems with Classification Systems

The problem with many classification systems is that they operate using only 1-place predicates, which allows for the comparison of an attribute to a constant value. Depending on the problem domain, solutions of this type may not be scalable.

A good example of this limitation is shown later during the description of results found during classification of the polygon dataset. Of the systems used for experimenting on this data, only RAGA was capable of recognizing relationships between variables when the data was in its raw form. It is still possible for other algorithms to consider these relationships, but this requires that the data first be tailored in a pre-processing stage.

Customizing the data to handle inter-attribute relationships requires adding derived values that express the result of a known operation. For example, in order to compare the length of an object to its width, the addition of

a new attribute is required. One possible solution is to create a value for length divided by width. This new variable can be examined by any classification system, and predictions can be made based upon it. One trivial observation is that a value of 1.0 would mean that length equals width.

There are a number of disadvantages to creating new attributes during the pre-processing stage, including a trade-off between an increase in the search space and increased input by the human expert. The trade-off is necessary because the types of relationships as well as the applicable attributes must be determined in advance. For cases where human bias is acceptable, the expert can try to determine beforehand which attributes should be compared, and how. Unfortunately, given the fact that data mining techniques are in place to discover information, this is not always possible because the expert does not understand the data well enough to decide.

In cases where the expert is not able to isolate key attributes or determine what possible relationships should be tested, then a new column must be generated for each pair of attributes using each of the available operations. This increases the size of the search space dramatically, making the process much longer and less likely to succeed. Furthermore, in cases where more than two attributes can be related, the increase in search space is exponential. Many of these cases result in a search that cannot be completed in a reasonable period of time.

To summarize, adding relationship attributes during the pre-processing stage is a poor choice because the human expert must decide upon both variables and the operations. This itself is a task in data mining, and although classification systems can take advantage of them, they are still unable to discover them.

6 Supervised learning experiments

6.1 Polygon Dataset

The polygon dataset describes a group of geometric shapes, and was designed to demonstrate the limitations in using classifiers with 1-place predicates.

The dataset consists of 190 differently sized shapes including triangles and rectangles. Each of these polygons is defined by only four attributes (L1-L4), which correspond to the lengths of up to four different sides. The L1 attribute indicates the length of bottom side, while L2-L4 define the remaining sides in clockwise order from the bottom.

One additional attribute, the *type*, is non-predictive and is used to train classifiers. This type can be one of five

values as follows: equilateral triangle, isosceles triangle, other triangle, square, and rectangle.

The dataset does not contain noise or errors, and all of the polygons obey the following rules:

- Equilateral triangle:** Three equal sides, one zero length.
- Isosceles triangle:** Two sides equal, one of different length, and one of zero length.
- Other triangle:** Three non-zero and non-equal sides, one side of zero length.
- Square:** All four sides equal.
- Rectangle:** Two pairs of opposite sides equal.

All triangles follow the geometric rule: $BC < AC + AB$, which guarantees that they are valid.

Several classification systems were used to generate classifiers for the polygon dataset. The most impressive results were found using *C5.0* (successor of *C4.5* [Quinlan 1993]), and *RAGA*. Common training and testing sets were used for all of the applications in two different tests. In the first test, the training set consists of 190 elements and the same data is used for testing. The second test uses an additional set of 100000 randomly generated legal polygons, where no additional training is performed. By evaluating this data with the rules discovered in the first test, it provides some measure of scalability when addressing unseen data from the same source.

6.1.1 Classification using C5.0

When *C5.0* was used to generate a decision tree, several of the software-specific options were tested. These options include the creation of *rulesets*, and several variations of *boosting*, however the results were not as good as the standard configuration and are therefore not shown.

The predictive accuracy of the data when testing against the full training set is 84.2%, using a decision tree of 46 leaves.

At several points within the tree, comparisons are made between different attributes and the constant zero. This is important because it determines the non-existence of particular side, and is general enough to handle unseen data. The problem is that this knowledge alone is not sufficient to accurately classify all of the data, and thus is not a general solution.

The result of classifying the 100000 randomly generated legal polygons is 39.37% correct, and 60.63% misclassified.

6.1.2 Classification using RAGA

The options used within *RAGA* are the defaults, as specified in the configuration for classification tasks.

A total of 19 rules (show below) were generated in order to achieve 100% predictive accuracy over all of the elements. A default class was not used, however the individual rules address enough subsets within the data to account for 100% coverage.

```

If (L2 = L3)^(L4 = L1) then (Class = 4)
If (L3 > L1)^(L2 < L4)^(L4 = L3) then (Class = 2)
If (L3 >= L1)^(L4 = L2)^(L2 < L1) then (Class = 5)
If (L3 = L1)^(L2 > L3) then (Class = 5)
If (L4 < L2)^(L2 = L1)^(L1 > L3) then (Class = 2)
If (L3 < L1)^(L2 < L1)^(L4 < L1) then (Class = 3)
If (L4 > L3)^(L2 < L4)^(L4 > L1) then (Class = 3)
If (L1 < L2)^(L4 < L2)^(L2 > L3) then (Class = 3)
If (L3 > L1)^(L3 > L2)^(L4 < L3) then (Class = 3)
If (L3 > L1)^(L2 = L3)^(L4 < L2) then (Class = 2)
If (L2 = L3)^(L4 > L1)^(L4 = L3) then (Class = 1)
If (L2 < L4)^(L4 = L1)^(L1 <= L3) then (Class = 1)
If (L4 < L2)^(L2 <= L1)^(L2 = L3) then (Class = 1)
If (L1 >= L4)^(L4 > L3)^(L3 < L1)^(L4 = L2) then (Class = 1)
If (L2 < L4)^(L4 = L1)^(L4 > L3)^(L3 < L1) then (Class = 2)
If (L2 < L4)^(L1 = L3)^(L4 < L1)^(L2 <= L1)^(L2 < L3) then
(Class = 2)
If (L1 < L3)^(L4 >= L3)^(L3 < L2)^(L4 = L2) then (Class = 2)
If (L4 < L2)^(L2 <= L1)^(L1 = L3)^(L2 < L3) then (Class = 2)
If (L3 < L1)^(L1 < L4)^(L2 = L4) then (Class = 2)

```

As expected, *RAGA* evolved a set of rules that considers the relationships between attributes, rather than their values. In early stages of the evolution, *RAGA* developed and tested rules such as:

```

If (L2 > 1248) Then (class = 2)

```

Due to the brittleness of these rules, they were not able to achieve 100% accuracy and were discarded by *RAGA*. At the conclusion of the search, only the more general rules remained.

The result of running the 100000 randomly generated legal polygons is 93.09% correct, and 6.91% misclassified.

For this dataset, *RAGA* did not take advantage of the default hierarchy. Additionally, the test for the constant zero was not explicit, but rather a comparison to other variables. This may not be as efficient, however it is equally accurate in both testing and training sets.

6.1.3 Manually Preprocessing Data for C5.0

Although *C5.0* is incapable of determining what inter-attribute relationships would be useful, an experiment was done to see how well the classification would perform if the

functionality did exist. To achieve this, six derived variables were added to the list of useable attributes, as follows:

R_L1L2 := L1 / L2.	Relationship between 1 & 2
R_L1L3 := L1 / L3.	Relationship between 1 & 3
R_L1L4 := L1 / L4.	Relationship between 1 & 4
R_L2L3 := L2 / L3.	Relationship between 2 & 3
R_L2L4 := L2 / L4.	Relationship between 2 & 4
R_L3L4 := L3 / L4.	Relationship between 3 & 4

The value of these derived fields will be a real number that indicates the relationship between the two specified sides, in terms of length. The general rule is:

Value = 1.0: The two sides are the same length
Value < 1.0: The first side is shorter than the second.
Value > 1.0: The first side is longer than the second.

A predictive accuracy of 95.3% was achieved using a total of 25 leaves in the decision tree. This tree is smaller and more accurate than the last one created using the same application, which lacked the derived relationship variables.

Also unlike the first experiment, this decision tree is more scalable because it considers relationships rather than constant values. If the program were capable of determining and working with these relationships automatically, then it would have achieved this improved accuracy during the first experiment.

The result of running the 100000 randomly generated legal polygons is 78.5% correct, and 21.5% misclassified. The reason for the high error rate is that C5.0 was unable to recognize the importance of a relationship that is exactly 1.0, and grouped together those with similar values.

7 Unsupervised Learning Experiments

Although several undirected data mining tasks were performed using RAGA, verifying the results is a non-trivial task in itself. Particularly for non-experts in the domain being examined, it is difficult to judge the value of any given rule, and may require the opinions of several knowledgeable individuals. Because of this additional effort, it is often difficult to test undirected data mining algorithms on real datasets.

7.1 Generating Synthetic Datasets

One way to address the problem of evaluating results from an unsupervised learner is to generate a set of random data that conforms to a pre-specified set of known rules. The data can optionally contain noise, incorrect and missing values, as well as non-essential attributes.

Example synthetic rule set

If $(A > 5) \wedge (B < 20)$ Then $(C > 8) \wedge (C < 12)$
If $(C < 4)$ Then $(B > 15)$
If $(C > 10)$ Then $(A > 4)$

Based on these rules a large number of elements can be generated, and this set can be used to test undirected data mining algorithms.

The ultimate goal for an undirected search algorithm is to find the exact set of rules as shown above, however this is only one of many possible outcomes. Assuming that several rules are generated with reasonable accuracy they do not necessarily have to match those used to define the set. Several factors that can influence the search results are erroneous or inaccurate values, an uneven distribution with respect to the defining rules, and the number of elements generated. This last factor is quite important because too little data often means less coverage of the rule space is considered. Results from this type of search often include rules that are too specific and do not have the proper level of support when applied to a superset of the data.

7.2 Results

Preliminary results indicated that experimentation with undirected data mining was not as straightforward as it was for simple classification. In most experiments there were several rules being sought, however only one was usually found by the conclusion of the search. Further analysis of this revealed the source of this problem, as well as one potential solution.

The reason for the single-rule solution is that the dominant rule is preferred at an early point in the process. This occurs because it has a high fitness value with respect to the others being considered. Unlike classification tasks, rules are not rewarded for uniquely addressing data elements, and therefore a very similar rule will often have a very similar fitness. As the generation evolves, variations of the good rule are created in an attempt to improve it, however the original copies and unwanted variations are not automatically discarded. Because these rules are similar in content the fitness is similar, and as with the primary rule the fitness is higher than other rules being considered. The result is a single rule that essentially dominates the entire population with variations of itself, and does not allow for the exploration of other niches.

To control this problem, a new fitness plug-in was created to penalize rules based on similarity. The goal is to force rules apart for being syntactically similar, without considering the elements they cover. This allows for the co-

existence of rule niches that answer overlapping subsets of elements.

Preliminary testing of this new fitness modifier has shown some success in unsupervised mining tasks. For example, a dataset was generated using the following five rules:

If $(B \leq 14) \wedge (D \geq 35) \wedge (D \leq 36)$ Then $E=1$
If $(B \leq 13) \wedge (D \geq 33) \wedge (D \leq 34)$ Then $E=2$
If $(A \geq 6.734049) \wedge (B \geq 18)$ Then $E=3$
If $(A \leq 4.562865) \wedge (D \geq 38) \wedge (D \leq 39)$ Then $E=4$
If $(A \leq 2.871963) \wedge (D \geq 31) \wedge (D \leq 32)$ Then $E=5$

During testing, RAGA was able to quickly generate several sets of rules that were semantically equivalent to those above. For example, after a one experiment the following rules were members of the rule hierarchy:

If $(b < 14) \wedge (d = 34)$ then $(e = 2)$
If $(d < 39) \wedge (b > 18) \wedge (a > 6.8179)$ then $(e = 3)$
If $(a \leq 5.5046) \wedge (d \geq 38)$ then $(e = 4)$
If $(a < 3.6621) \wedge (d \leq 32)$ then $(e = 5)$

It is obvious that the set is not complete, however 100% data coverage was not a goal during these experiments. Rather than trying to classify the data, RAGA was simply searching for *interesting* associations. Considering this, the incomplete coverage is not an issue, but the quality of the rules generated must be closely examined.

The main difference between the discovered rules and the actual rules is that the attribute *D* is only bounded on one side. (ie: an upper or lower bound is indicated, but not both). After analyzing the results and the dataset it was discovered that (in all of the cases addressed) the second bound was simply not required once the second attribute in the equation was isolated. With this understood it appears that RAGA has in fact simplified some of the original rules, although only for a subset of the data.

8 Conclusion

We have described our current research with RAGA, including the recent enhancements and modifications. As shown by several experiments, RAGA is capable of handling both supervised and unsupervised learning tasks, and in some cases has distinct advantages as compared to other systems.

In supervised data mining tasks, RAGA demonstrates the ability to generate a scalable rule hierarchy that is unattainable by algorithms that are limited to 1-place predicates. Although this advantage may depend on the

problem domain, the success of RAGA does not depend on this feature alone.

In unsupervised data mining tasks the results are not compared to those of other algorithms, however the preliminary testing generated rules that semantically match or closely approximate the actual underlying rules of synthetic datasets.

Bibliography

Catral, R., Oppacher, F., Deugo, D. (1999a). "Rule Acquisition with a Genetic Algorithm". Proceedings of the 1999 Congress on Evolutionary Computation, p. 125-129.

Catral, R., Oppacher, F., Deugo, D. (1999b). "Using Genetic Algorithms to Evolve a Rule Hierarchy". Lecture notes in Computer Science, Vol. 1704, p. 289-294. 125-129.

[Berry, Linoff, 1997] Michael J.A. Berry, Gordon Linoff, 1997. Data Mining Techniques: For Marketing, Sales, and Customer Support. Wiley computer publishing.

John R. Koza, (1992). "Genetic Programming: On the programming of computers by means of natural selection". MIT Press, Cambridge, Mass.

David J. Montana, (1995). "Strongly typed genetic programming". Evolutionary computation 3(2).

R.J. Quinlan, (1993). C4.5 is a licensed product that can be acquired from Morgan Kaufmann Publishers, Ann Arbor, Michigan.

I. W. Flockhart, N. J. Radcliffe, (1996). "A Genetic Algorithm-Based Approach to Data Mining". Proceedings of the Second International Conference on Knowledge Discovery and Data Mining.

John H. Holland, (1975). "Adaptation in Natural and Artificial Systems". University of Michigan Press (Ann Arbor).