A Genetic Algorithm for Generalized Rule Induction

Alex A. Freitas

CEFET-PR, DAINF
Av. Sete de Setembro, 3165
Curitiba - PR, Brazil, 80230-901
Tel: ++55 +41 322-4544 ext. 648
alex@dainf.cefetpr.br

Keywords: genetic algorithms, data mining, knowledge discovery, generalized rule induction.

Abstract.

Data mining consists of the efficient discovery of knowledge from databases. This paper presents a new genetic algorithm designed for discovering a few interesting, high-level prediction rules from databases, rather than discovering classification knowledge (often a large rule set) as usual in the literature. Three important data mining issues addressed by our algorithm are the interestingness of the discovered knowledge, the computational efficiency of the algorithm, and the trade-off between representation expressiveness and efficiency.

1 Introduction.

In essence, data mining consists of the efficient discovery of knowledge from large databases. The knowledge discovery process consists of several phases [1], but the core of this process is usually the application of machine learning-based algorithms to databases. However, in “conventional” machine learning, the emphasis is usually on finding accurate knowledge, as measured e.g. by the classification error rate, in the case of classification tasks. In contrast, in data mining, knowledge accuracy is often a secondary concern, and the emphasis is on finding novel, interesting, comprehensible knowledge, which will be ultimately used by a human being (e.g. a senior business analyst) for high-level decision making. In general it is not very desirable to return to the user all the discovered rules. They would be simply too many in the case of large databases, and this would hinder the user’s decision-making process. In data mining it is usually better to return to the user only a small set of high-quality rules, the “knowledge nuggets”, rather than all rules.

In this paper we present a genetic algorithm (GA) designed to discover a few high-quality rules from a given database. The data mining task addressed by our GA consists of predicting the values of a small number of goal attributes given the values of other attributes, called predicting attributes. Note that this is an extension of the conventional data mining task of classification, where there is only one goal attribute to be predicted. The data mining task addressed in this paper is also often called generalized rule induction.

It should be noted that there are post-processing, rule set-refinement algorithms that extract a small set of high-quality rules from a large set of discovered rules, so that conventional machine-learning algorithms can be used, together with these post-processing algorithms, for extracting knowledge nuggets [2], [3]. However, this approach has two drawbacks. First, the discovery of a large set of rules for a given problem is usually a very time-consuming activity, and probably a not very cost-effective one, since many of the discovered rules will eventually be discarded by the post-processing algorithm anyway. Second, the post-processing algorithms can be quite complex and computationally expensive themselves. Hence, the approach followed in this paper (discovering only a small set of high-quality rules) tends to be significantly more efficient.

In addition, it should be emphasized that most data mining-related GAs proposed in the literature – e.g. [4], [5], [6], [7], [8] – address the task of classification only, while our GA addresses the task of generalized rule induction, as mentioned above.

This paper is organized as follows. Section 2 discusses the design of our GA. Section 3 presents the results of computational experiments. Finally, Section 4 presents the conclusions.

2 Design of the Genetic Algorithm (GA).

This section presents our genetic algorithm (GA) designed for the task of generalized rule induction. Since this algorithm was designed to discover a few high-quality rules ("knowledge nuggets"), we will call it GA-Nuggets. The next subsections discuss several aspects of the design of GA-Nuggets.

2.1 Knowledge Representation.

There is a trade-off between a knowledge representation's expressiveness power and its associated computational cost. A propositional logic ("0-th order") representation allows efficient execution of a data mining algorithm, but it is rather limited in expressiveness. For instance, a propositional miner cannot discover a rule with the condition "Income > Expenditure", since this condition involves a relation (">") between two distinct attributes. On the other hand, a first-order logic representation has a much greater expressive power - allowing the discovery of conditions such as "Income > Expenditure" - at the expense of a much larger rule space to be searched, which significantly increases the processing time of the data mining algorithm.

Our solution for the above trade-off is to use data dictionary information (commonplace in real-world databases systems) as a kind of background knowledge that guides the algorithm to consider only a small number of first-order relationships between attributes, rather than all possible relationships. The basic idea is to use an attribute-typing system significantly stronger than the one normally used in data mining algorithms. In general, data mining algorithms handle attributes by classifying them into one out of very few attribute types, often just two types: categorical and continuous attributes. However, database systems offer much richer information about attribute types, which can be used to filter out non-sensible first-order relationships. For instance, suppose an employee database contains the attributes Salary and Age. Both attributes would be simply considered as "continuous" by most data mining algorithms. However, in principle it does not make sense to let the algorithm consider first-order relationships such as "Salary > Age". These two attributes have very different types and domains, they are not "compatible", and the data mining algorithm can be aware of that by consulting the data dictionary. On the other hand, from the data dictionary the algorithm can find out which attributes are "compatible" to, say, attribute Salary. In this example, Salary and its compatible attributes would belong to the same data type, namely "money".

	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7
A_0	yes	yes	yes	no	no	no	no	no
A_1	yes	yes	yes	no	no	no	no	no
A_2	yes	yes	yes	no	no	no	no	no
A_3	no	no	no	yes	yes	no	no	no
A_4	no	no	no	yes	yes	no	no	no
A_5	no	no	no	no	no	yes	no	no
A_6	no	no	no	no	no	no	yes	no
A_7	no	no	no	no	no	no	no	yes

Figure 1: Example of an Attribute-Compatibility Table for 8 attributes.

Consulting information in the data dictionary of the database system, GA-Nuggets computes an Attribute-Compatibility Table (ACT), as shown in Figure 1. The number of rows and columns of this table equals the number of attributes being mined. In the example shown in Figure 1, there are 8 attributes, named A_0...A_7. If attributes A_i and A_j - i,j in [0..7] - are compatible, the corresponding entry i,j of the table has the value "yes", otherwise that entry has the value "no". An ACT is constantly used by GA-Nuggets, not only to generate individuals of the initial population but also to apply genetic operators during evolution (see below). We believe that, by using an ACT, GA-Nuggets achieves a good trade-off between knowledge representation expressiveness and computational cost for searching the rule space.

2.2 An Individual's Encoding.

In GA-Nuggets an individual corresponds to a single rule of the form "if A then C", where A is the rule antecedent, consisting of a conjunction of conditions on the values of the predicting attributes, and C is the rule consequent, specifying the value predicted for a given goal attribute. The rule antecedent and the rule consequent are encoded separately in an individual, as follows.

An individual's rule antecedent is encoded as a variable-length list of rule conditions. The rule antecedent can contain two types of rule conditions: (a) propositional logic (hereafter propositional, for short) rule conditions; (b) first-order logic (hereafter first-order, for short) rule conditions, in which the pair of operands is compatible, as defined in the Attribute-Compatibility Table (ACT) for the data being mined (see above).

More precisely, propositional conditions have the form A_i *op* V_{ij}, where A_i is the i-th predicting attribute, *op* is a relational operator (=, ≥, ≤) and V_{ij} is the j-th value of the i-th predicting attribute's domain. First-order conditions have the form A_i *op* A_j, where A_i and A_j ($i \neq j$) are two compatible attributes, as defined in the corresponding ACT, and *op* is a relational operator (=, ≥, ≤). For instance, the first-order condition $A_3 \geq A_4$ is satisfied by a database tuple iff the value of attribute A_3 in the tuple is greater than or equal to the value of attribute A_4 in the tuple.

To avoid the production of individuals with many conditions in the rule antecedent (which would reduce the degree of comprehensibility of the corresponding rule, going against the philosophy of data mining), the user specifies a maximum number of antecedent rule conditions for an individual.

An individual's rule consequent is encoded as a single condition of the form $G_i = V_{ij}$, where G_i is the i-th goal attribute and V_{ij} is the j-th value of the i-th goal attribute's domain. Note that although each rule predicts the value of a single goal attribute, in a given population different rules can predict the value of different goal attributes.

The genetic material of an individual's rule antecedent is randomly generated when the initial population is created and is thereafter subject to the action of crossover and mutation, as will be seen below. In contrast, the production of the genetic material of an individual's rule consequent is treated in a special way, due to its strategic importance in determining the quality of the rule represented by the individual. The basic idea is to delay the generation of the rule consequent until fitness-computation time, when the rule consequent is generated in such a way that the rule's predictive accuracy is maximized. A more elaborated description of this idea is as follows.

First of all, crossover and mutation operators are applied only to an individual's rule antecedent. Once these genetic operators have been applied, the just-produced rule antecedent of an offspring individual is matched against all the tuples of the database, to compute statistics to be used for calculating the individual's fitness. These statistics contain the frequency of tuples satisfying the rule antecedent, distributed per goal attribute and per goal attribute value, as illustrated by the table shown in Figure 2. In this Figure there are two binary goal attributes, denoted G_1 and G_2, each of them with two attribute values, denoted V_1 and V_2 . Each entry of the table shown in Figure 2, denoted F_{ij} (i=1,2; j=1,2,), contains the frequency of database tuples that satisfy the rule antecedent and have value V_j for goal attribute G_i.

	goal attribute values V_1	V_2
goal attribute G_1	F_{11}	F_{12}
goal attribute G_2	F_{21}	F_{22}

Figure 2: Distribution of the frequency of tuples satisfying a rule antecedent, per goal attribute and per goal attribute value.

Once a table like the one shown in Figure 2 is computed, GA-Nuggets will select, to make up the rule consequent, the pair of <goal attribute, goal attribute value> whose frequency F_{ij} maximizes the predictive accuracy of the rule. As an example, suppose that predictive accuracy maximization is achieved by selecting the maximum value F_{ij} in the table. Suppose also that the maximum frequency value is F_{12}. Then the GA will choose the goal attribute G_1 and its value V_2 to make up the rule consequent "$G_1 = V_2$".

It is interesting to note that this delayed generation of an individual's rule consequent has two advantages. First, it maximizes the individual's predictive accuracy for a given rule antecedent, by design. Second, it leads to a gain in computational efficiency, since by matching a single rule antecedent against the database just once we are computing, in a *single database pass*, the predictive accuracy associated with several possible rules. This is significantly more efficient than matching the entire rule (antecedent and consequent) against the database for each possible rule. Note also that this is important because predictive accuracy computation usually by far dominates the processing time of a GA mining a large database.

A simple example of a rule encoded by an individual of GA-Nuggets is illustrated in Figure 3. In this Figure, A_i denotes the value of the i-th predicting attribute, while G_i denotes the value of the i-th goal attribute. In this example the rule antecedent has two conditions. The first one is a first-order condition, while the second one is a propositional condition. The rule consequent predicts a given value ("0") for the goal attribute G_1.

(rule antecedent) IF ($A_0 = A_1$) AND ($A_2 \geq 0$)
(rule consequent) THEN ($G_1 = 0$)

Figure 3: A simple example of a rule encoded by an individual.

2.3 Fitness Function.

The fitness function of GA-Nuggets consists of two terms. One term measures the predictive accuracy of the rule associated with the individual, while the other term measures the degree of interestingness of the rule consequent. These terms are elaborated in the following.

One of the simplest, most intuitive estimates of a rule's predictive accuracy is its "confidence factor" on the training data, calculated as the ratio X / Y, where X is the number of tuples that satisfy both the rule antecedent and the rule consequent and Y is the total number of tuples that satisfy the rule antecedent. However, in the early experiments with GA-Nuggets we observed that this term had a strong bias towards the discovery of "small disjuncts", i.e. rules that cover a very small number (say about 5) of tuples. It is relatively easy to find small disjuncts that have a confidence factor of 100% on the training data. However, these rules are likely to reflect idiosyncrasies of the data which may not constitute the knowledge nuggets we are looking for, and are likely to have a significantly smaller confidence factor on the test set. (For a more comprehensive discussion about the problem of small disjuncts, see e.g. [9], [10].)

Therefore, we decided to modify this term of the fitness function in such a way that its bias towards small disjuncts were reduced, but still keeping as much as possible of the simplicity of the confidence factor definition. Hence, we have borrowed the idea of subtracting 1/2 from the numerator X in the above formula for confidence factor, i.e. we use the term

$$(X - 1/2) / Y. \qquad (1)$$

This idea was proposed by Quinlan [11] in the context of decision tree pruning. To see why this approach reduces the bias towards small disjuncts, note that the smaller the size of the disjunct associated with a rule, the greater will be the reduction of the numerator X in comparison with the denominator Y in the above formula, and the smaller will be the value of the above expression.

The second term of GA-Nuggets' fitness function is a measure of the degree of interestingness of the consequent ("THEN part") of the rule associated with the individual. The rationale behind the introduction of this term in the fitness function is the following. Let G_i denote a given goal attribute and g_i denote a given value in the domain of this attribute. The higher the "prior probability" (relative frequency) of the value g_i in the training set, the easier is to discover a rule predicting the value g_i (i.e. having consequent $G_i = g_i$), and, in some sense, the less interesting the discovered rule is. The challenge is to discover highly accurate and general rules predicting rarer goal attribute values. These rules tend to be more interesting for the user.

Based on the above argument, the second term of GA-Nuggets' fitness function is defined as

$$(1 - \text{Prob}(g_i))^{1/k}, \qquad (2)$$

where $\text{Prob}(g_i)$ is the prior probability (relative frequency) of the goal attribute value g_i, and k is a user-defined parameter, empirically set to 2 in our experiments.

The exponent 1/k in the above formula can be regarded as a way of reducing the influence of the rule consequent interestingness in the value of the fitness function. The higher the value of k, the closer the value of the rule consequent interestingness term gets to one, and so the smaller the influence of this term in the fitness function (formula 3 below). The value of k was empirically set to 2 in the experiments reported in this paper.

Finally, the fitness function of an individual i is defined as the product of the above two terms, that is:

$$\text{Fitness}_i = ((X - 1/2) / Y) * (1 - \text{Prob}(g_i))^{1/k} \qquad (3).$$

Note that each of the terms takes on values between 0 and 1, so that the fitness of an individual also takes on a value between 0 and 1.

2.4 Genetic Operators.

We have designed a crossover operator that takes into account our problem-dependent individual encoding. In essence, we use a one-point crossover. However, a straightforward application of the conventional crossover operator would lead to the production of invalid genetic material. In particular, a predicting attribute might occur in two or more conditions of the rule antecedent, with inconsistent values between the conditions. To consider a concrete example, suppose that the first rule condition of one of the mating individuals is "$A_1 = 0$", while the last rule condition of the other mating individual is "$A_1 = 1$". A crossover operation would produce an offspring individual whose conjunctive rule antecedent would include the two rule conditions "$A_1 = 0$" and "$A_1 = 1$" - which obviously characterizes an invalid genetic material, since attribute A_1 cannot take on values 0 and 1 at the same time.

To avoid this kind of problem, our crossover operator includes a method to repair invalid genetic material. After crossover, GA-Nuggets checks all the rule conditions of each of the two just-produced offspring individuals. Invalid conditions are replaced with a new rule condition. For simplicity and uniformity, this replacement is performed by calling the mutation operator (see below), which produces only valid rule conditions.

The mutation operator produces only rule conditions that are valid genetic material, avoiding inconsistency with other rule conditions of the same individual. The mutation operator is applied to an entire rule condition, rather than to the rule condition's components only (an operand or operator), and it has access to all the rule conditions of the corresponding individual. When a rule condition is to be mutated, its operands and operator are discarded and a triple of operand-operator-operand is randomly generated, subject to the restriction that the newly-generated condition is not inconsistent with other rule conditions.

The mutation operator can produce either propositional or first-order conditions, so that it also has access to the Attribute-Compatibility Table for the data being mined. Note that the new condition produced by the mutation operator can have either one or two attributes, depending on the new condition having a propositional or first-order representation (see section 2.2). Hence, the mutation operator replaces a rule condition with a new condition where the operands and operator are randomly generated, subject to the restriction that the attribute(s) occurring in the new condition do not occur already in other rule conditions of the same individual. We have also devised an adaptive mutation rate mechanism, to be explained in section 2.5.

It is important to clarify that in GA-Nuggets the mutation operator is used in two situations: (a) After crossover, in order to introduce new genetic material in an individual. In this case mutation is applied only with a mutation probability, which is dynamically adapted during the search (again, see section 2.5). (b) *Within* crossover, in order to repair invalid genetic material produced by crossover. In this case mutation is applied whenever necessary, regardless of any mutation probability.

Finally, both crossover and mutation act on the rule antecedent only. Recall that the rule consequent is automatically re-generated to maximize the rule's predictive accuracy, whenever the individual undergoes any modification in the genetic material of the rule antecedent.

2.5 Adaptive Mutation Probability and Result designation.

In many GA applications, only the best individual of the last generation is designated as the GA's result. In our case, however, each individual corresponds to a single rule, and it is desirable to designate more than one rule (but still a *small* number of rules) as the result. Hence, it is desirable to avoid that the population be dominated by a single highly fit individual.

Our approach to cope with this problem is to use an adaptive mutation probability, where the value of the mutation probability is increased as the population becomes dominated by an individual. More precisely, the higher the number of copies of the fittest individual in the current population, the higher the mutation probability used in that generation. In the current version of GA-Nuggets, the value of the mutation probability varies from 5% (when the fittest individual has a single copy in the current population) to 90% (when all individuals of the current the population are copies of the fittest individual).

Although these mutation probability values seem to be high at first glance, we should bear in mind two points. First, we use an elitist strategy (see below), so that the high mutation probabilities cannot cause the loss of the fittest individual of a generation. Second, even though mutation introduces a seemingly unfavorable (with respect to predictive accuracy) change in the rule antecedent, recall that after mutation the rule will have its consequent re-generated in such a way to maximize the rule's predictive accuracy. Hence, in GA-Nuggets mutation promotes diversity with a reduced risk of significantly degrading the quality of an individual. Note that GA-Nugget's adaptive mutation approach is significantly different from the one used by some "conventional" adaptive mutation probability GAs, where the mutation probability is often reduced in the final generations.

The above method allows that the last generation contains several different, high-quality individuals, as will be shown in section 3. In this sense, the above adaptive mutation probability method achieves an effect similar to niching, with one advantage: the former tends to be more computationally efficient than the latter. In any case, in future versions of GA-Nuggets, we might implement some niching mechanism and compare its effectiveness against the one of the above adaptive mutation probability method.

GA-Nuggets returns to the user not only the best individual of the last generation but also other individuals of the last generation that have fitness greater than an user-specified value.

3 Preliminary Computational Results and Discussion.

We have done a preliminary, “proof-of-concept” experiment to evaluate the GA proposed in this paper. (We intend to do more extensive experiments in the near future.) In this experiment the parameters of GA-Nuggets were set as follows. The population size was 100, and each run consisted of 200 generations. The maximum number of conditions in the rule antecedent was set to 4 (half the number of predicting attributes). For reproduction we used tournament selection, where the number of individuals in the tournament was 2, a common default in the literature. An elitist reproduction strategy was also used, where the best individual of a generation was copied unaltered to the next generation. We used problem-specific crossover and mutation operators, as discussed in section 2.4. The crossover probability was 0.8 and the mutation probability was varied from 0.05 to 0.9, as explained in section 2.5. The next two subsections discuss the data set and the computational results of our experiment.

3.1 The Data Set.

We have used a synthetic database, consisting of 1,000 tuples and 10 attributes. The first 8 attributes, denoted $A_0...A_7$, are predicting attributes, while the last 2 attributes, denoted G_1 and G_2, are goal attributes. The data mining task consisted of predicting the values of the goal attributes, given the values of the predicting attributes. The values of the predicting attributes were randomly generated according to a uniform probability distribution. More precisely, predicting attributes A_0, A_1, A_2, A_3, A_4, A_5 were defined as ordinal categorical attributes, each of them with a domain size of 5. Since their values are ordinal, the operators applied to them in rule conditions can be not only “=“ but also ”≥“ and “≤“. Predicting attributes A_6 and A_7 are categorical, the former with a domain size of 2 and the latter with a domain size of 4. The only operator that can be applied to these attributes in rule conditions is “=“.

To implement the idea of using strong attribute typing to focus the search for first-order attribute relationships (see section 2), the above 8 predicting attributes were classified into 5 different attribute sets, each associated with a distinct attribute type. Within each of these attribute sets, the attributes are compatible with each other, and first-order rule conditions can be formed only when the condition’s attributes belong to the same attribute set. The attribute sets were defined according to the Attribute-Compatibility Table shown in Figure 1. As can be deduced from that table, the first set of compatible attributes is composed by attributes A_0, A_1 and A_2; the second set by attributes A_3 and A_4, and the last three sets are unary sets, composed respectively by attributes A_5, A_6 and A_7 (i.e. each of these 3 attributes can be used only in propositional rule conditions, not in first-order conditions, since they are not compatible to any other attribute).

The two goal attributes G_1 and G_2 had a binary domain, taking on the value 0 or 1. For each tuple of the database, once its predicting attribute values were generated, two classification functions were applied to those values, to determine the values of the goal attributes G_1 and G_2. To introduce some noise (commonplace in the real-world), the wrong value of a goal attribute was assigned to a tuple with a probability of 5%. The two classification functions used are shown in Figure 4, where A_i denotes the value of the i-th predicting attribute and G_i denotes the value of the i-th goal attribute.

IF ((($A_0 < 3$) AND ($A_2 > 1$)) OR
(($A_0 = A_1$) AND ($A_3 < A_4$)) OR
(($A_2 > A_0$) AND ($A_1 > 2$) AND ($A_6 = 1$)))
THEN ($G_1 = 1$)
ELSE ($G_1 = 0$)

(a)Function used to compute value of goal attribute G_1

IF ((($A_4 = 3$) AND ($A_6 = 0$) AND ($A_7 > 1$)) OR
(($A_0 < 2$) AND ($A_2 > 2$) AND ($A_3 = A_4$)) OR
(($A_0 < A_1$) AND ($A_5 > 1$) AND ($A_6 = 1$) AND ($A_7 > 0$)))
THEN ($G_2 = 1$)
ELSE ($G_2 = 0$)

(b)Function used to compute value of goal attribute G_2

Figure 4: Classification functions used to compute the values of the goal attributes in the synthetic database.

3.2 Computational Results and Discussion.

The discussion presented in this section focuses mainly in three points. First, we analyze whether or not the adaptive mutation probability mechanism was able to avoid convergence of the population to a single highly fit individual and allow the evolution of other good individuals. Second, we analyze the predictive accuracy of the discovered rules on a separate test set (unseen by the algorithm during the search for rules). Third, we analyze the generality of the discovered rules (recall that we have introduced in the fitness function a certain bias against small disjuncts).

Let us start with the adaptive mutation probability mechanism proposed in the paper. This mechanism was able to avoid the convergence of the population to a single highly fit individual. The best individual of the run was created in generation 21. A few generations later this individual started to dominate the population. For instance, in generation 28 there were 27 copies of this individual. However, as the number of copies of this individual increased, the adaptive mutation probability was correspondingly increased. As a result, the number of copies of the best-of-the-run individual in any generation was almost always less than 50 (half the population size), and the number of copies of this individual in the last generation was 38.

This allowed other good individuals to evolve and to be present (sometimes with a few copies of themselves) in the last generation too. The result returned to the user consisted of all individuals (rules) of the last generation with fitness greater than or equal to 0.7. In our case, this result consisted of seven individuals.

Let us turn now to fitness and predictive accuracy issues. On the average, the seven discovered rules had a fitness value of 0.7595, resulting from an estimated predictive accuracy on the training set - computed by formula (1) - of 0.9573 and a rule consequent interestingness measure of 0.7935.

It should be noted that, although the value of the rule consequent interestingness measure is approximately the same in both the training and the test sets (with the conventional assumption that these two data sets follow the same probability distribution), the value of the predictive accuracy in the test set (unseen data) can be quite different from the one in the training set. Hence, we have measured the actual prediction accuracy rate of a rule on unseen data (the test set), a new database with 1,000 synthetically-generated tuples. The test set data has the same attribute definitions and probability distributions as the training set, but the two data sets have of course different data (this is easily achieved by changing the random seed used in the synthetic data generator). The prediction accuracy rate was calculated, on the test set, as the ratio X / Y, where X is the number of tuples that satisfy the rule antecedent and have the goal attribute value predicted by the rule consequent and Y is the total number of tuples that satisfies the rule antecedent. In our experiment, the seven rules returned to the user had an average predictive accuracy on the test set of 0.8534. At first glance, this value is significantly smaller than the above-mentioned estimated predictive accuracy (on the training set) of 0.9573. However, we should bear in mind two points.

First, predictive accuracy is just one term of the fitness function, and GA-Nuggets is maximizing fitness, i.e. the product of predictive accuracy and rule consequent interestingness. Actually, rule consequents with higher degrees of interestingness are more difficult to find, and so their predictive accuracy tends to be somewhat lower. Second, the value of 0.8534 was averaged over the seven discovered rules. It turns out that two of these rules had a relatively small predictive accuracy (0.667 and 0.571) on the test set. If we ignore these two "outliers", the remaining five rules had an average predictive accuracy of 0.947 on the test set. This is a very high value, considering that, in theory, the best predictive accuracy on the test set would be 0.95, due to the existence of 5% noise in the data. (For a general discussion on how noise limits prediction accuracy, see e.g. [12]).

To summarize, one can say that GA-Nuggets discovered five high-quality rules (concerning both predictive accuracy on the test set and rule consequent interestingness) and two not so good rules (with relatively small predictive accuracy on the test set).

It is also interesting to mention that the five high-quality discovered rules had a predictive accuracy on the test set higher than the best-of-the-run individual, which confirms that it is important to avoid the convergence of the population to a single highly fit individual.

Turning to the generality of the discovered rules, on the average the seven rules returned to the user covered 19.6 tuples. Two of the discovered rules could be considered as small disjuncts, covering 7 and 9 tuples. (One of these small disjuncts had a good predictive accuracy of 83,3% on the test set, but the other one had a relatively bad predictive accuracy of 57,1% on the test set.) Out of the seven discovered rules, the largest disjunct was a rule covering 56 tuples, which had a very good predictive accuracy of 97,9% on the test set. Overall, these results show that GA-Nuggets is still discovering relatively small disjuncts. Hence, although the use of formula (1) has the effect of somewhat biasing the search towards larger disjuncts, this formula alone seems not to be enough, and it seems desirable to include another term in the fitness function with an explicit bias towards rules with higher generality (larger number of tuples covered). This will be the object of future research.

4 Conclusion.

We have presented a new genetic algorithm, called GA-Nuggets, designed for the efficient discovery of a few interesting, high-level prediction rules ("knowledge nuggets") from databases. We have focused on the task of generalized rule induction, which can be regarded as an extension of the well-known task of classification.

GA-Nuggets searches for rules that have both a high predictive accuracy and a high degree of rule-consequent interestingness (defined in the algorithm as a small prior probability of the value predicted by the rule consequent).

GA-Nuggets uses a problem-dependent individual representation which represents a good trade-off between the expressiveness power and the computational efficiency of knowledge representations. On one hand it allows the discovery of first-order logic rules, with significantly greater expressiveness power than the propositional representation used by the vast majority of data mining systems. On the other hand, to reduce the computational complexity associated with the larger rule space of first-order systems, GA-Nuggets uses strong attribute typing information found in data dictionaries (commonplace in real-world database systems). This kind of information allows the algorithm to limit the search for first-order relationships to small groups of compatible attributes, rather than to all attributes.

Two other efficiency-related aspects of GA-Nuggets are as follows. First, as discussed in Section 2.2, in a *single* database pass we compute the information necessary to calculate the predictive accuracy of several potential individuals, whose corresponding rules predict different values of different goal attributes. This is clearly much more efficient than performing one database pass for computing the fitness of each of those potential individuals.

Second, since GA-Nuggets addresses the task of generalized rule induction, a single run of GA-Nuggets can discover rules predicting the value of different goal attributes. If we were to use classification algorithms in this task, we would need two run a classification algorithm n times, once for each of the n goal attributes to be predicted. While the required processing time can still be reasonable for two goal attributes, the situation quickly gets worse as the number of goal attributes is increased. GA-Nuggets avoids the excessively long time associated with running one classification algorithm for each goal attribute, since it searches for rules for predicting any of the goal attributes at the same time.

Of course, the advantages of GA-Nuggets do not come for free. Recall that classification algorithms usually discover a “complete classification knowledge”, in the sense that every unseen tuple can be classified by the discovered knowledge. In contrast, GA-Nuggets discovers only a few interesting, high-level rules, without bothering for discovering a complete classification knowledge. In some sense, GA-Nuggets trades knowledge completeness for knowledge comprehensibility and computational efficiency. This can be undesirable in some applications, but we believe that overall, in the context of data mining and knowledge discovery applications, knowledge comprehensibility and computational efficiency are more important than knowledge completeness.

Finally, recall that the experiments reported in this paper are somewhat preliminary, since they have used only a synthetic database and that GA-Nuggets is still evolving. Data mining is mainly an empirical area, and for a validation of the promising results reported in this paper it is necessary to apply GA-Nuggets to real-world databases. We also intend to extend our algorithm in several ways, including a more comprehensible evaluation of different fitness functions and their effects on the kind of knowledge nuggets that are discovered by GA-Nuggets, as well as investigating some niching scheme to improve the control of the variety of rules produced by the algorithm.

References.

[1] U.M. Fayyad, G. Piatetsky-Shapiro and P. Smyth. From data mining to knowledge discovery: an overview. In: U.M. Fayyad, et al. (Eds.) *Advances in Knowledge Discovery and Data Mining*, 1-34. AAAI/MIT Press. 1996.

[2] J.A. Major and J.J. Mangano. Selecting among rules induced from a hurricane database. *J. Intelligent Information Systems* 4(1), Jan./95, 39-52.

[3] F. Gebhardt. Choosing among competing generalizations. *Knowledge Acquisition*, 3, 1991, 361-380.

[4] S. Augier, G. Venturini and Y. Kodratoff. Learning first order logic rules with a genetic algorithm. *Proc. 1st Int. Conf. Knowledge Discovery & Data Mining*, 21-26. AAAI Press, 1995.

[5] F. Neri and A. Giordana. A parallel genetic algorithm for concept learning. *Proc. ICGA-95*, 436-443.

[6] C. Anglano, A. Giordana, G. Lo Bello and L. Saitta. A network genetic algorithm for concept learning. *Proc. 7th Int. Conf. on Genetic Algorithms*, 434-441. Morgan Kaufmann, 1997.

[7] C.Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13, 1993, 189-228.

[8] B. de la Iglesia, J.C.W. Debuse and V.J. Rayward-Smith. Discovering knowledge in commercial databases using modern heuristic techniques. *Proc. 2nd Int. Conf. Knowledge Discovery & Data Mining*. AAAI, 1996.

[9] R.C. Holte, L.E. Acker and B.W. Porter. Concept learning and the problem of small disjuncts. *Proc. Int. Joint Conf. AI (IJCAI-89),* 813-818.

[10] G.M. Weiss. Learning with rare cases and small disjuncts. *12th Int. Conf. Mach. Learning*, 558-565. 1995.

[11] J.R. Quinlan. Generating production rules from decision trees. *Proc. IJCAI-87*, 304-307.

[12] C. Cortes, L.D. Jackel and W.-P. Chiang. Limits on learning machine accuracy imposed by data quality. *Proc. 1st Int. Conf. Knowledge Discovery & Data Mining*, 57-62. 1995.