# Breeding Decision Trees Using Evolutionary Techniques

**Athanasios Papagelis** PAPAGEL@AHEADRM.COM
**Dimitris Kalles** KALLES@AHEADRM.COM
Computer Technology Institute, Patras, Greece, PO BOX 1122, 261 10
AHEAD Relationship Mediators, Patras, Greece, GR 26221

## Abstract

We explore the use of genetic algorithms to directly evolve classification decision trees. We argue on the suitability of such a concept learner due to its ability to efficiently search complex hypotheses spaces and discover conditionally dependent as well as irrelevant attributes. The performance of the system is measured on a set of artificial and standard discretized concept-learning problems and compared with the performance of two known algorithms (C4.5, OneR). We demonstrate that the derived hypotheses of standard algorithms can substantially deviate from the optimum. This deviation is partly because of their non-universal *procedural bias* and it can be reduced using global metrics of tree quality like the one proposed.

## 1 INTRODUCTION

Decision tree induction is a very popular and practical method for pattern classification. The construction of optimal decision trees has been proven to be NP-complete, under several aspects of optimality and even for simple concepts (Murthy, 1998). Current inductive learning algorithms use variants of impurity functions like information gain, gain ratio (Quinlan, 1986), gini-index (Breiman *et al.*, 1984), distance measure (de Mantaras, 1989) to guide the search. Fayyad (1991) discusses several deficiencies of impurity measures. He pointed out that impurity measures are insensitive to inter-class separation and intra-class fragmentation, as well as insensitive to permutations of the class probability distribution. Other authors (Kononenko *et al.*,1997) (Ragavan & Rendell, 1993) indicated that those measures assume that attributes are conditionally independent and therefore they have poor chances of revealing a good hypothesis in domains with strong conditional dependencies between attributes. Furthermore, several authors have provided evidence that the presence of irrelevant attributes can mislead the impurity functions towards producing bigger, less comprehensible, more error-prone classifiers.

This work is an attempt to overcome the use of greedy heuristics and search the decision tree space in a more natural way. More specifically, we make use of genetic algorithms to directly evolve binary decision trees in the conquest for the one that most closely matches the target concept. On doing so we adopt a natural representation of the search space using actual decision trees and not binary strings. We couple our objective with a simplification motivation. We use GAs to robustly evolve *accurate* as well as *simple* decision trees.

Although GAs have been used for classification and concept learning tasks (Wilson, 1986) (Goldberg, 1989) (Booker *et al.*, 1990) (De Jong et al, 1993) (Janikow, 1993) (Congdon, 1995), there is rather little work on their utility as a tool to evolve decision trees[1]. Close but distinct relatives of this work comes from Koza (1991) who points out the suitability of the tree genome for decision tree building, Bot and Longdon (2000) who used GP techniques to evolve linear classification trees and Nikolaev and Slavov (1998) who analyzed a global fitness landscape structure and its application on decision tree building.

Since Schaffer (1993) introduced the concept of different levels of suitability for *learner biases* (the fact that no algorithm biases can be suitable for every target concept) the idea that there is no universally better algorithm is fast maturing on the machine learning community. We might do better to map different algorithms to different groups of problems with practical importance.

There are several types of biases but here we distinguish between *preference* and *procedural* bias. A preference bias is based on the learner's behavior while a procedural bias is based on the learner's design. For example, C4.5 is biased towards accurate, small trees (preference bias) and uses the gain-ratio metric and minimum-error pruning (different procedural biases). A preference bias is most often desirable since it determines the characteristics of the produced tree. On the other hand, an inadequate procedural bias may severely affect the quality of the output. The proposed search imposes a new *weak* procedural bias, one that allows the concept learner to consider a relative large

[1] However, the tree genome has been extensively used in GP to represent program parse trees.

number of hypotheses, in a relative efficient manner. The proposed weak bias employs global metrics of tree quality. We thus shift from "how to induce a tree" (standard, impurity-based induction) to "what criteria an induced tree must satisfy". We view setting a policy direction, as opposed to how a policy should be implemented, as a de facto decrease in bias with significant advantages over other highly used procedural biases in complex search spaces.

There is an active debate on whether less greedy heuristics can improve the quality of the produced trees. Garey and Graham (1974) showed that greedy algorithms can be made to perform arbitrarily worse than the optimal. Norton (1989) showed that exhaustive lookahead applied to ID3 reduced tree sizes on average and produced small gains in accuracy. Ragavan and Rendell (1993) showed that their LFC algorithm that performed both lookahead and constructive induction can perform well on tasks involving feature interaction. On the other hand, Murthy and Salzberg (1995) found that one-level lookahead yield larger, less accurate trees on many tasks. Quinlan and Cameron-Jones (1995) reported similar findings and hypothesized that lookahead can yield "fluke theories" that fit the training data but have poor predictive accuracy.

Genetic algorithms are neither hill-climbing systems nor exhaustive searchers. Rather, they are a type of beam search. When tuned properly GAs can aggregate desired characteristics of both hill-climbing and exhaustive search algorithms.

The rest of this paper is organized in three sections. In the next section we elaborate on the construction of the proposed system (GATree) and the modifications to the standard mutation-crossover operators. We then demonstrate via an experimental session that the proposed search procedure indeed works and point out some of its benefits. Finally, we put all the details together identifying good points or possible pitfalls and discussing lines of research that have been deemed worthy of following.

## 2 THE GATree SYSTEM

In order to apply GAs to a particular problem, we need to select an internal representation of the space to be searched combined with an external evaluation function, which assigns scores to candidate solutions. Traditionally, GAs use binary strings to represent points in search space. However, such representations do not appear well suited for representing the space of concept descriptions that are generally symbolic in nature and with varying length and complexity.

There are two different approaches one might take to resolve this issue. The first involves changing the fundamental GA operators so as to work well with the complex non-string objects, while the second attempts to construct string representations of solutions that minimize any changes to the basic GA philosophy.

We stuck with the first approach for two fundamental reasons. First, it is natural to use a tree structure to represent decision trees and the mutation-crossover operators can be efficiently altered to match this structure. Second, it is not trivial to alter the basic mutation-crossover operators so as to be used with string representatives of decision trees and at the same time preserve trees structures.

For this work we have used GALIB (Wall, 1996), a robust C++ library of Genetic Algorithm Components. GALIB offers a wide range of internal representations combined with easily adjusted parameters so as to optimally tune its behavior.

### 2.1 DATA PREPROCESSING AND GENETIC OPERATORS

We use GALIB's tree representation to build a population of *minimal* binary decision trees (trees that consist from one node and two leaves). Every decision node has a random chosen value as its installed test. This is done in two steps. First we choose a random attribute. Then, if that attribute is nominal we randomly choose one of its possible values; if it is continuous we randomly pick an integer value belonging to its min-max range. This approach reduces the size of the search space and it is straightforward. Still, it has problems with real-valued attributes; for this work we concentrated on nominal attributes. Leaves are populated using the same line of thought; we just pick a random class from the ones available.

The basic form of the proposed algorithm introduces minimum changes to the mutation-crossover operators. Mutation chooses a random node of a desired tree and it replaces that node's test-value with a new random chosen value. When the random node is a leaf, it replaces the installed class with a new random chosen class (Figure 1).

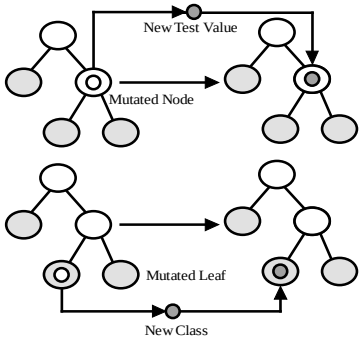


*Figure 1.* Mutation Examples

The crossover operator chooses two random nodes and just swaps those nodes' sub-trees. Since predicted values rest

only on leaves, the crossover operator does not affect tree's coherence (Figure 2).

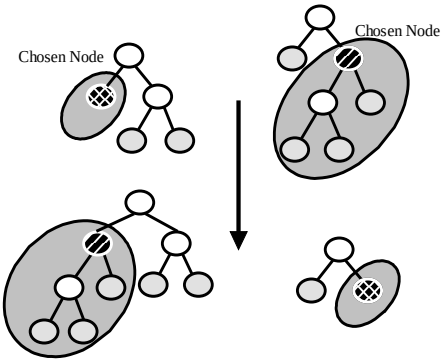


*Figure 2.* Crossover Examples

## 2.2 PAYOFF FUNCTION

A natural way to assign utility to a decision tree is by using it to classify the known instance-set. Each tree is granted a scaled payoff according to its performance. We chose to grant higher payoffs to smaller trees (assuming that they perform almost equally with bigger ones). This is a way to avoid unnecessary test-values replications along a specific path (an unavoidable side-effect since we do not exclude any already used attribute-value from being used again) while at the same time we derive comprehensible decision trees. Thus, the fitness function is balanced between accuracy and size:

$$payoff(tree\ i) = CorrectClassified_i^{\ 2} * \frac{x}{size_i^{\ 2} + x} \qquad (Eq.1)$$

The second part of the product (the size factor) includes a factor $x$ which has to be set to an arbitrary big number. Thus, when the size of the tree is small the size factor is near one, while it decreases when the tree grows big. This way, the payoff is greater for smaller trees.

The size factor can be altered to match individual needs. For example, if we had set $x$ to 1,000,000 then the GA would search inside a bigger search space (more trees). However, bigger search spaces inevitably mean less optimized trees for a fixed number of generations. Alternative size factors can be used that would prefer trees with sizes inside some range (assuming that we know that the target concept can be represented with a decision tree of a specific size). This could lead to more efficient search and thus less time for the GA to converge.

## 2.3 ADVANCED SYSTEM CHARACTERISTICS

To reduce the overcrowding problem (Goldberg, 1989) we used a scaled payoff function, which aimed at reducing the similarity of decision trees on the population. When there were many decision trees with similar characteristics[2] we reduced their payoff function.

Furthermore, we implemented several alternative crossover and mutation functions. An interesting alternative crossover used a bias evolution towards more fit subtrees. We implemented a data structure that kept for every node the correct/incorrect classified instances passing from it. That information was used to alter the probability with which a node was chosen for mutation or crossover. More accurate subtrees had less chance to be used for crossover or mutation.

To speed up evolution we also implemented an altered version of Limited Error Fitness (LEF) (Gathercole & Ross, 1997). This technique introduces an error limit. If the number of errors of an individual, during the process of evolution, is higher than the error limit, all remaining cases are treated as errors. This means that poor individuals will not be evaluated on the entire training set, saving CPU time. With moderate usage of the error limit we were able to produce significant CPU timesavings and insignificant accuracy loses.

To test the effectiveness of all those components we further implemented a second layer genetic algorithm. The genomes of this algorithm included coded information about the mutation/crossover rates and different heuristics as well as a number of other optimizing parameters. The second layer was tested using several datasets to ensure result robustness. Some of the most recurring results were a mutation rate of 0.005, a crossover rate of 0.93, the need to use a crowding avoidance technique and the fact that alternative mutations/crossovers did not produce significant improvements compared to the basic mutation/crossover operators.

## 2.4 SEARCH SPACE AND INDUCTION COSTS

The size of search space depends on tree size. Let $D(n)$ be the number of topologically different binary decision trees of $n$ leaves. It has been proven (Fayyad, 1991) that:

$$D(n) = \begin{cases} 0 & n = 0 \\ \frac{1}{n}\binom{2n-2}{n-1} & n > 0 \end{cases} \qquad (Eq.\ 2)$$

The search space depends also on the amount of different attribute-values and classes of the underlying concept. Suppose that $\alpha$ is the sum of the distinct values[3] of all features and that $c$ is the distinct problem classes. Since we use binary decision trees the number of internal nodes is $n-1$. An internal node can use any one of the $\alpha$ distinct values and that holds for every node. Since we allow values to be

[2] Similarity of decision trees was estimated using the formula: tree_diff = |(levels_tree1-levels_tree2)+(nodes_tree1-nodes_tree2)|.

[3] We assume only nominal attributes. For continuous ones the search space is enormously bigger since the possible test values inside a min-max range are infinite.

reused, a binary decision tree of n leaves has $\alpha^{n-1}$ syntactically different trees regarding the attribute values. This has to be multiplied with the $c^n$ syntactically different decision trees regarding the problem classes. Therefore, the total number of syntactically different binary decision trees of *n* leaves is:

$$T(n,\alpha,c) = D(n) * \alpha^{n-1} * c^n \qquad (Eq.3)$$

When we search for a specific tree we do not stick to trees with specific number of leaves; instead we search on a space containing a wide range of tree sizes. Assuming that the number of training instances is *k*, the maximum number of leaves is also *k* (one instance at every leaf). Thus, the size of the search space is:

$$S(k,\alpha,c) = \sum_{n=1}^{k} T(n,\alpha,c) \qquad (Eq.4)$$

A serial search for the best tree is prohibitive even under very restrictive situations. Suppose that we set *k* to a small number (e.g., 10) and that we have a rather simple concept to learn (2 attributes with 3 different values for each and 2 problem classes). We further reduce the space size by considering only the possible decision trees for $n$=10 (even though we should consider all the trees for $n \in [1,10]$). This gives, $T(10,6,2) = 4862 \cdot 6^9 \cdot 2^{10} = 50{,}173{,}704{,}142{,}848$. Any search algorithm has to do better then successively test every possible tree.

It can be proven (Quinlan, 1986) that feature selection at a node of greedily induced trees, has complexity $O(ak)$ for *a* features and *k* instances. In contrast, one-level lookahead's complexity is $O(a^2k^2)$ (Murthy and Salzberg, 1995), or more generally $O(a^dk^d)$ for *d-1* levels of lookahead. Those factors are the dominant ones during decision tree induction since subsequent future selection are based on a partitioned dataset and the number of nodes cannot be greater than the number of instances.

The cost of the proposed heuristic is based on four different factors: the number of generations (*gen*), the number of genomes that are evaluated in the population (*pop*), the number of instances (*k*) and the average path an instance has to follow from the root to a leaf (*avPath*). Then the cost of the algorithm is: $gen * pop * k * avPath$. Quite safely, the *pop* parameter can be set to a constant multiplier of the number of dataset features *a* ( $pop = c_1 a$ ) with $c_1 << a$. Furthermore, under a very pessimistic higher boundary we can set *avPath* to *k*. With those assumption, the complexity of the algorithm is $O(gen * k^2 * a)$. Appendix A presents an extension over the basic algorithm that caches previous classifications and can lower the basic complexity to $O(gen * k * a)$ especially when the derived trees become large. One cannot precisely express the generations needed for convergence since they depend on the complexity of the underlying concept. However, since the GA evolves complete solutions, the algorithm can be terminated whenever necessary. One should also not forget that GAs are highly parallel procedures, and thus, even lower absolute time requirements are possible using a parallel evolution. Another advantage of this procedure is that the output is not just a decision tree but also a collection of decision trees that can be used alternatively or collectively.

## 3 EXPERIMENTS

Our first aim was to examine the rate with which GATree produces fit hypotheses for target concepts. To ensure complexity variety we used several artificial datasets that were constructed using *DataGen*; a program that uses random rules to generate artificial instance-sets (Melli, 1999). The goal was then to use those sets to reconstruct the underlying knowledge.

For the cross validation experiments we used WEKA; a library of Machine Learning Algorithms in Java (Witten & Frank, 2000). We made use of WEKA's implementations for two known classifiers; the C4.5 implementation (Quinlan, 1993) with binary decision trees and the OneR implementation (Holte, 1993). The parameters for those classifiers were chosen to be the default ones used by WEKA (Version 3.1.6).

Cross-validation was first performed on a number of artificial datasets explicitly designed to demonstrate some of GATree's benefits over greedy heuristics. Then, we compared its performance against C4.5 and OneR over several discretized datasets. C4.5 and OneR have different representational bias: C4.5 is biased towards accuracy (and secondarily size) while OneR is biased towards extremely simple classification rules (and secondarily accuracy). We demonstrate that their derived hypotheses can unnecessarily deviate from the dual goal (under straightforward assumptions). We argue that this deviation is partly because of their inappropriate procedural bias and thus, can be reduced using global metrics of tree quality. For all comparisons, we adopted a standard 5-fold cross-validation.

A problem with GAs is the diversity of the obtained results due to factors like the initial random seed, the initial population and number of generations. The diversity may be surprisingly high for complex search spaces given that we have limited resources (limited number of genomes and generations). Instead of using a big number of generations and an equally big number of genomes, we adopted a diverse strategy that uses relatively few generations and a small number of genomes but repeats the learner several times. For every output of the cross-validation experiments we repeated the algorithm 10 times and then picked the highest fit genome (based on training set). For every experiment we present the standard deviation between cross-validation runs.

Experiments showed that the dominant cost factors are the number of genomes, the number of generations, the factor *x* and the number of instances. An evolution consisting of 200 generations with 200 genomes on a 1000 instances

dataset (20 attributes, 100 attribute-value pairs) with factor *x* set to 10000 takes approximately 30 seconds on a Pentium III 600Mhz. The same dataset takes about 3 seconds to be evaluated using Weka's C4.5. With a constant *x* factor we may expect trees of similar sizes throughout evolution. Thus, if we double generations, population or instances the needed time for evolution will be doubled too. Although it is clear that the current form of the algorithm cannot be scaled easily to really large datasets, it can be used with most UCI like datasets without processing power being a real problem.

The algorithm's parameters during the experiments are presented in Table 1. We have chosen to use overlapping populations; every generation replaces 25% of the worst individuals of the previous one. The initial population was set to 200 even though it can vary depending on the complexity of the target concept. The number of generations was fixed to 200 for all cross validation experiments. The mutation and crossover rates were set to 0.005 and 0.93 accordingly, based on the second layer feedback. We used the basic form of mutation and crossover operators. In order to allow reproducibility we initialized the random generator using the value 123456789.

The factor *x* was set to 1000 for the experiments with standard datasets. A small factor *x* means a bias towards small trees. However this bias is flexible since the algorithm may deviate from it (only as much is needed) to produce an acceptable hypothesis. For all other experiments we set the factor *x* to 10000 (emphasis on accuracy).

Table 1. Experiments Parameters

| | |
|---|---|
| Evolution Type | Generational |
| Initial Population | 200 |
| Generations | 200 – 800 |
| Generation Gap | 25 % |
| Mutation Probability | 0.005 |
| Crossover Probability | 0.93 |
| Size Factor | 1000-10000 |
| Random Seed | 123456789 |

### 3.1 HYPOTHESES FITNESS

To ensure that the GA produces fit hypotheses we tested its performance with two synthetic datasets. Both datasets had 3 attributes (A, B, C), that could take up to 26 distinct values (a…z) and 3 problem classes (c1, c2, c3). For those experiments we set the number of generations to 800.

The exact activation rules of the first synthetic dataset are presented below:

(31.0%) **c1** ← B=(**f** or **g** or **j**) & C=(**a** or **g** or **j**)
(28.0%) **c2** ← C=(**b** or **e**)
(41.0%) **c3** ← B=(**b** or **i**) & C=(**d** or **i**)

Attribute A is not used by any activation rule and, thus, its main influence is as noise.
Although the target concept is not very complicated, the search space is huge. Figure 3 presents the results obtained using GATree with 100 random instances of the abovementioned concept.

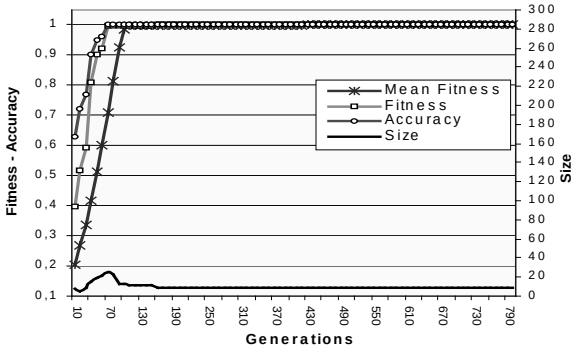


*Figure 3.* Results for the simple concept

*Mean fitness* refers to the average fitness score of all genomes inside the current population. *Fitness* is the fitness score of the best individual. *Accuracy* is the obtained classification accuracy using the best genome and *Size* is the number of nodes of the best individual.

The algorithm quickly (in less than 100 generations) finds a maximum fit hypothesis and then (for about 80 generations) makes minor adjustments adopting smaller trees that guarantee the obtained accuracy.

More complex problems may not converge to maximum fit hypotheses. Often the misclassified instances would be those that create an exception to the underlying concept characteristics and thus, by not creating a rule to classify them, we produce a more fit hypothesis for test data (this can be viewed as a form of flexible pruning). However, on noisy datasets, oversearching may produce overfitted trees. In such situations we could either use alternate size fitness functions (which somehow avoid the incorporation of noise) or post-process the derived trees with a pruning technique.

Figure 4 presents the results for a more complex artificial dataset. The dataset was created using eight rules (in contrast with the three rules of the first dataset). Furthermore, the rules had more complex structures, adopting more disjunctions per rule. The first two activation-rules were as below:

(15.0 %) **c1** ← A=(**a** or **b** or **t**) & B=(**a** or **h** or **q** or **x**)
(14.0%) **c1** ← B=(**f** or **l** or **s** or **w**) & C=(**c** or **e** or **f** or **k**)

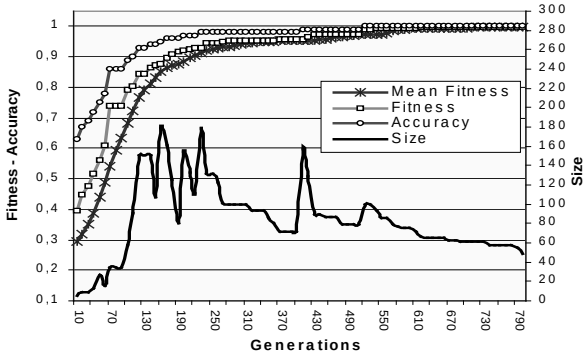


*Figure 4.* Results for the complex concept

Evidently GATree had a harder time to find a fit hypothesis. More search had to be done, inside bigger and more complex trees. An interesting part of the graph is the size peaks that appeared during searching. For example, between the 370th and 430th generation the size of the tree was overly expanded and then reduced. Such peaks identify an upper limit in the accuracy of the produced tree that needed a hypothesis jump in order for the evolution to continue. Such regions may also indicate problematic points for greedy heuristics, since they specify local maximums.

A *diminishing returns effect* is evident on those graphs. GATree was quick to produce relatively fit hypotheses but subsequent generations showed a slowly attained progress. This also indicates that, even though GAs get very close to the global optimum it is very expensive to exactly reach it. Probably it would be wiser to use an alternate strategy to fine-tune the result.

### 3.2 CONDITIONALLY DEPENDENT AND IRRELEVANT ATTRIBUTES

Consider the example data set given in Table 2.

Table 2: Example dataset

| A1 | A2 | A3 | Class |
|---|---|---|---|
| T | F | T | T |
| T | F | F | T |
| F | T | F | T |
| F | T | T | T |
| F | F | F | F |
| F | F | F | F |
| T | T | T | F |
| T | T | F | F |

The class value is determined with XOR function on attributes A1 and A2, while the third attribute A3 is randomly generated. Although such a concept seams rather easy, the greedy heuristic of C4.5 falsely estimates that the contribution of A3 is the highest among the three attributes. Moreover, C4.5 estimates that the contribution of the A1, A2 is very low. Therefore, C4.5 derives a decision tree with only one decision node (after pruning) that has the attribute A3 installed in it. Of course such a decision tree is unacceptable.

On the other hand, the less greedy strategy of GATree (which tries to minimize a tree's size while at the same time maximize accuracy) easily discovers the desired decision tree (Figure 5)

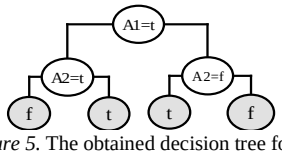


*Figure 5.* The obtained decision tree for the conditionally dependent attributes

Even if we had prevented C4.5 from pruning the tree, it would create two replicated, identical to figure 5, subtrees under the initial A3 node; a substantially bigger, less comprehensible tree.

In order to further empirically evidence the previous mentioned deficiency of greedy heuristics, we created several artificial datasets with strong dependent and irrelevant attributes. The characteristics of those datasets are presented in the following table:

Table 3: Artificial datasets characteristics

| Name | Attrib. | Class Function | Noise | Instanc. | Random Attributes |
|---|---|---|---|---|---|
| Xor1 | 10 | (A1 **xor** A2) **or** (A3 **xor** A4) | No | 100 | 6 |
| Xor2 | 10 | (A1 **xor** A2) **xor** (A3 **xor** A4) | No | 100 | 6 |
| Xor3 | 10 | (A1 **xor** A2) **or** (A3 **and** A4) **or** (A5 **and** A6) | 10% class error | 100 | 4 |
| Par1 | 10 | Three attributes parity problem | No | 100 | 7 |
| Par2 | 10 | Four attributes parity problem | No | 100 | 6 |

For the experiments we used C4.5 as a typical representative of greedy induction. The mean accuracy results of standard 5-fold cross validation are presented in table 4.

Almost all experiments showed that greedy heuristics could not efficiently deal with conditionally dependent attributes. GATree outperformed them in a more than significant level. However, one of the datasets (Xor3) showed that the presence of class noise could make GATree deviate from good predictors.

Table 4: Classification accuracy

| | C4.5 | GATree |
|---|---|---|
| Xor1 | 67±12.04 | 100±0 |
| Xor2 | 53±18.57 | 90±17.32 |
| Xor3 | 79±6.52 | 78±8.37 |
| Par1 | 70±24.49 | 100±0 |
| Par2 | 63±6.71 | 85±7.91 |

### 3.3 EXPERIMENTS WITH STANDARD DATASETS

Experiments were conducted using several datasets from the UCI Repository (Blake *et al.*, 2000). Every continuous attribute was discretized using WEKA's unsupervised equal-frequency binning method. The number of bins was optimized using the entropy minimization criterion. We decided not to use a supervised (class based) discretization to artificially produce erroneous, complex search spaces with irrelevant as well as somewhat mutually dependent attributes. Table 5 presents the classification accuracy and the derived decision trees size (with pruning).

GATree was able to produce the most accurate results (on average) even though the difference with C4.5 is not significant. However, those results were accompanied by

extremely small decision trees (C4.5 produced six times bigger trees on average).

It is clear that under such noisy datasets OneR can exceed C4.5 in accuracy, in several datasets. However, in the general case it performs substantially worse. We attribute that behaviour to its procedural bias. OneR picks only one attribute and then branch on its values. However, this overlooks the fact that there can be several other informative attributes while, equally crucially, there can be branches based on irrelevant values.

On the other hand, C4.5 produces good accurate results but with unnecessarily big trees. Pruning consistently under-prunes the resulted trees. However, the overly sized trees cannot be attributed only to the inadequacy of pruning to predict the optimal pruning level. When a decision tree induction method prunes away a subtree, it applies a statistical test that decides whether that subtree is justified by the data. But that decision has only been applied locally, in the pruned subtree. Its effect has not been allowed to percolate further up the tree, perhaps resulting in different choices being made on attributes to branch on. This is the dual process of greedy induction; pruning is another hill-climbing technique, which can quickly guide to a good result, or on the other hand, can substantially deviate from the optimum.

Contrary to greedy induction, GATree produces a dynamic, small-biased, accuracy/size based tree optimisation. This procedure is potentially superior to the (treated as uncorrelated) build-prune procedure of greedy heuristics. Nevertheless, GATree's "pruning capabilities" is just a side effect of its design. It is still open whether there are better ways to achieve its effect using more precise global metrics of tree quality.

Table 5

| | Accuracy | | | Size | |
|---|---|---|---|---|---|
| | **C4.5** | **OneR** | **GATree** | **C4.5** | **GATree** |
| Colic | 83.84±3.41 | 81.37±5.36 | 85.01±4.55 | 27.4 | 5.84 |
| Heart-Statlog | 74.44±3.56 | 76.3±3.04 | 77.48±3.07 | 39.4 | 8.28 |
| Diabetes | 66.27±3.71 | 63.27±2.59 | 63.97±3.71 | 140.6 | 6.6 |
| Credit | 83.77±2.93 | 86.81±4.45 | 86.81±4 | 57.8 | 3 |
| Hepatitis | 77.42±6.84 | 84.52±6.2 | 80.46±5.39 | 19.8 | 5.56 |
| Iris | 92±2.98 | 94.67±3.8 | 93.8±4.02 | 9.6 | 7.48 |
| Labor | 85.26±7.98 | 72.73±14.37 | 87.27±7.24 | 8.6 | 8.72 |
| Lymph | 65.52±14.63 | 74.14±7.18 | 75.24±10.69 | 28.2 | 7.96 |
| Breast-Cancer | 71.93±5.11 | 68.17±7.93 | 71.03±8.34 | 35.4 | 6.68 |
| Zoo | 90±7.91 | 43.8±10.47 | 82.4±4.02 | 17 | 10.12 |
| Vote | 96.09±3.86 | 95.63±4.33 | 95.63±4.33 | 11 | 3 |
| Glass | 55.24±7.49 | 43.19±4.33 | 53.48±4.33 | 60.2 | 8.98 |
| Balance-Scale | 78.24±4.4 | 59.68±4.4 | 71.15±6.47 | 106.6 | 8.92 |
| **AVERAGES** | **78.46** | **72.64** | **78.75** | **43.2** | **7.01** |

## 4 DISCUSSION

GATree can be easily extended to make use of sets of independent decision tree classifiers. Recall that the building blocks that (mainly) comprise the final tree are created during the first step of the algorithm (where it produces a set of minimal random binary trees) and thus, those building blocks are different every time we use a different seed to initialize the random generator. Even when distinct populations of building blocks cannot substantially differ between them (when for instance there are not many attribute-values and/or classes), there is the payoff function that can be altered to prefer classifiers with different characteristics. Now, whenever an unknown instance has to be classified one can decide about its class by using a majority vote over every decision tree inside the classifier set.

A basic drawback of GAs, compared with greedy heuristics, is speed. In order to evolve 500 decision trees for 500 generations with 25% generation gap we have to create and test 62875 decision trees. Although those trees are cheap to create and use, the time burden is substantially bigger than that of other heuristics. A potentially fruitful idea is in the making for the near future regarding this issue.

This idea is based on the fact that the crossover/mutation operators change the tree from a node downwards. Instead of classifying every instance using the changed tree we can classify only the instances that belong to the changed-node's subtree. That can result in substantial timesavings when the crossover is near the tree's fringe. The extra burden is additional structures that keep track of every instance passing from some node, together with node statistics (how many instances pass from it, how many of them were correctly classified). Appendix A presents an estimation on the average percent of instances that may have to be re-classified using this technique compared to 100% of the original algorithm; it shows a more than significant decrease in the expected number of added classifications.

## 5 CONCLUSION

In this work we have explored how GAs can be used to directly evolve decision trees. The whole approach is based on conceptual simplicity, adopting only necessary extensions to basic GAs and small *a priori* bias. The experiments have indicated potential advantages of GAs over other greedy heuristics especially when there are irrelevant or strongly dependent attributes. Furthermore, experiments demonstrated the implications of adopting greedy procedural biases.

## References


Blake, C., Keogh, E., & Merz, J. (2000) UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, Department of Information and Computer Science.

Booker L.B., D.E. Goldberg & J.H. Holland (1989). Classifier Systems and Genetic Algorithms, *Artificial Intelligence*, 40, 2, 235-282.

Bot, M., Langdon, W., (2000) Application of Genetic Programming to Induction of Linear Classification Trees, In proceedings of GECCO 2000.

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984) *Classification and Regression Trees*. Wadsworth International Group.
Congdon, C.,B.. (1995). *A comparison of genetic algorithms and other machine learning systems o a complex classification task from common disease research*, Doctoral dissertation, Department of Electrical Engineering and Computer Science, Univercity of Michigan.
DeJong, K.A., Spears, W. M., & Gordon, D.F. (1993). Using genetic algorithms for concept learning. *Machine Learning*, 13, 161-188.
Fayyad, M.U. (1991). *On the Induction of Decision Trees for Multiple Concept Learning*, Doctoral dissertation, Department of Electrical Engineering and Computer Science, Univercity of Michigan.
Garey R., M, and Graham L.,R (1974) Performance bounds on the splitting algorithm for binary testing. *Acta Informatica*, 3:347--355.
Gathercole, C., Ross, P., (1997) Tackling the Boolean even N parity problem with genetic programming and limited-error fitness. *Genetic Program'97: Proceedings of the Second Annual Conference*, 119-127.
Goldberg D. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley.
Holte, R.C. (1993). Very simple classification rules perform well on most commonly used datasets, *Machine Learning* 11,63-91.
Janikow, C., Z. (1993) A knowledge-intensive genetic algorithm for supervised learning, *Machine Learning*, 13,189-228.
Kononenko, I., E. Simec, and M. Robnik-Sikonja (1997). Overcoming the myopia of inductive learning algorithms with RELIEFF. Applied Intelligence 7, 39—55
Koza,J.R (1991) *Concept formation and decision tree induction using the genetic programming paradigm*. Parallel problem solving from nature. Springer Verlag, Berlin.
Mantaras., R.S. (1989). ID3 Revisited: A distance based criterion for attribute selection, Proceedings of Int. Symp. Methodologies for Intelligent Systems, Charlotte, North Carolina, USA.
Melli, G. (1999). Data Set Generator Program, www.datasetgenerator.com.
Mitchell, T. (1997). *Machine Learning*. McGraw-Hill Series in Computer Science.
Murthy, S. & Salzberg, S. (1995), Lookahead and pathology in decision tree induction, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1025-1031
Murthy S., K (1998). Automatic construction of decision trees from data: A multidisciplinary survey. *Data Mining and Knowledge Discovery*.
Nikolaev N., Slavov V. (1998) *Inductive Genetic Programming with Decision Trees*, Intelligent Data Analysis v.2 number 1.
Norton, S., W. (1989). Generating Better Decision Trees. *In Proceedings of the Eleventh International Joint Conference on AI*, 800-815.
Quinlan, R. (1986). *Induction of decision trees*, Machine Learning, 1:81-106,1986
Quinlan, R. (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, San Mateo, CA.
Quinlan, J. R. and Cameron-Jones, R. M.(1995) Oversearching and layered search in empirical learning. *In Proceedings of the 14th International Joint Conference on AI*,1019-1024, Montreal, Canada.
Ragavan, H. and L. Rendell (1993), Lookahead Feature Construction for Learning Hard Concepts, *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, MA, pp. 252--259 (Morgan Kaufmann, San Francisco, CA).
Schaffer, C. (1993). Overfitting avoidance as bias, *Machine Learning*, 10, 153-178
Wall, M. (1996). *GAlib: A C++ Library of Genetic Algorithm Components.* M.I.T.
Wilson, S.W. (1986). *Classifier system learning of a boolean function.* Cambridge, MA: Rowland Institute for Science.
Witten, I., Frank, E. (2000) *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann Publishers, San Mateo, CA

## Appendix

We can estimate the average number of instances that have to be re-classified as a function of tree levels and the original number of instances.

This analysis is based on the assumption that instances are equally distributed between nodes. This means that if a father-node has $k$ instances, then $k/2$ instances arrives at each one of its two children. Another assumption is that nodes are chosen for crossover or mutation with equal probability. Therefore, if we have a tree with size $n$ then the probability of a node to be chosen is $1/n$.

Our average analysis deals with the two extremes of binary decision trees: the *linear* binary decision tree and the *complete* binary decision tree. Let $l$ be the number of levels of a binary decision tree. Then, a *linear* binary tree has $l+1$ leaves and a total of $2l+1$ nodes while a *complete* binary decision tree has $2^l$ leaves and a total of $2^{l+1}-1$ nodes. Any other binary decision tree with $l$ levels lies somewhere between those two ends.

Figure 6 presents the *linear* and *complete* binary decision trees of three levels together with the number of instances at each node (supposing that instances are equally distributed and that their total number is $k$).

If the root node of the *complete* decision tree was chosen for crossover/mutation then all $k$ instances should be re-classified. On the other hand, if a leaf was chosen then only $k/8$ instances should be re-classified. Since every node has a probability $\frac{1}{2^{l+1}-1}$ to be chosen it can be proven that the average number of instances that have to be used for the new hypotheses evaluation is: $C(k,l) = k\frac{(l+1)}{2^{l+1}-1}$

Using the same line of though, the average number of instances that have to be re-classified in a *linear* decision tree is: $L(k,l) = \frac{k}{2l+1}(1+\sum_{m=1}^{l}\frac{1}{2^{m-1}})$

We know that the number of instances that have to be re-classified lies somewhere between those two extreme averages. For example, in a tree with eight levels we need to re-classify between 2% and 18% of the initial instances.

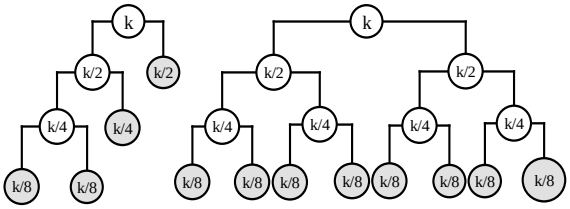


*Figure 6. Linear* and *Complete* binary decision trees of 3 levels

## Acknowledgment

This work was originally conceived and partially done while the authors were at the Computer Technology Institute of Patras, Greece. They are currently employed by AHEAD Relationship Mediators SA, Greece.