# Towards the Automatic Design of Decision Tree Induction Algorithms

Rodrigo C. Barros,* Andre C. P. L. F. Carvalho
University of Sao Paulo
Sao Carlos, SP - Brazil
{rcbarros,andre}@icmc.usp.br

Marcio P. Basgalupp
UNIFESP
S. J. dos Campos, SP - Brazil
basgalupp@unifesp.br

Alex A. Freitas
University of Kent
Canterbury, Kent - United Kingdom
A.A.Freitas@kent.ac.uk

## ABSTRACT

Decision tree induction is one of the most employed methods to extract knowledge from data, since the representation of knowledge is very intuitive and easily understandable by humans. The most successful strategy for inducing decision trees, the greedy top-down approach, has been continuously improved by researchers over the years. This work, following recent breakthroughs in the automatic design of machine learning algorithms, proposes two different approaches for automatically generating generic decision tree induction algorithms. Both approaches are based on the evolutionary algorithms paradigm, which improves solutions based on metaphors of biological processes. We also propose guidelines to design interesting fitness functions for these evolutionary algorithms, which take into account the requirements and needs of the end-user.

## Categories and Subject Descriptors

I.2.6 [**Induction and Knowledge Acquisition**]: learning—*decision tree induction, evolutionary algorithms, automatic design*

## General Terms

Algorithms

## 1. INTRODUCTION

A decision tree is a classifier represented by a flowchartlike tree structure which has been widely used to represent classification models, due to its comprehensible nature that resembles the human reasoning. In a recent poll of the *kdnuggets* website [1], decision trees figured as the most used data mining/analytic method by researchers and practitioners, reaffirming its importance in machine learning tasks. Decision tree induction algorithms present several advantages over other learning algorithms, such as robustness to noise, low computational cost for generating the model, and ability to deal with redundant attributes [2].

Several attempts on optimizing decision tree algorithms have been made by researches within the last decades, even though the most successful algorithms date back to mid-80's [3] and early 90's [4]. Several strategies were employed for deriving accurate decision trees, such as bottom-up induction [5], linear programming [6], hybrid induction [7], evolutionary induction [8, 9], ensemble of trees [10], just to name a few. Nevertheless, no strategy has been more successful in generating accurate and comprehensible decision trees with low computational effort than the greedy top-down induction strategy.

A greedy top-down decision tree induction algorithm recursively analyzes whether a sample of data should be partitioned in subsets according to a given rule, or if no further partitioning is needed. This analysis takes into account a stopping criterion (for deciding when tree growth should halt) and a splitting criterion (responsible for choosing the "best" rule for partitioning a subset). Further improvements over this basic strategy include pruning tree nodes for enhancing the tree's capability of dealing with noisy data, and strategies for dealing with missing values, multiple classes, imbalanced classes, among others.

Endless approaches were proposed in the literature for each one of these *design components* of top-down decision tree induction algorithms. For instance, new measures for node splitting tailored for a vast number of application domains were proposed, as well as many different strategies for selecting multiple attributes for composing the node rule (multivariate split). There are even works in the literature that survey the numerous approaches for pruning a decision tree [11, 12]. It is clear that by improving these design components, we can obtain more robust top-down decision tree induction algorithms.

The pioneering work by Pappa and Freitas [13] on automatically evolving rule induction algorithms poses the following question: "if by changing these design components of rule induction algorithms can result in new, significantly better ones, why not keep on trying systematically?" Since the human manual approach would be unfeasible given the vast amount of different strategies regarding each major design component in a decision tree induction algorithm, we propose in this paper an approach similar to the one of Pappa and Freitas [13]: employing evolutionary algorithms to evolve generic decision tree induction algorithms. We propose two different strategies

*The first author is supported by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).


*GECCO'11,* July 12–16, 2011, Dublin, Ireland.

for doing so: (i) a linear genome representation, in which each major design component can be indexed by integers in a fixed-length string, and additional real-valued parameters can be incorporated; and (ii) a grammar-based genetic programming approach, in which a grammar is used for generating individuals in the form of derivation trees. Furthermore, we suggest some guidelines for designing appropriate fitness functions for the evolutionary algorithm. We first review the research on decision tree induction algorithms in Section 2, pointing out the major design components of a decision tree induction algorithm. Then, we detail our approach in Section 3 and we present our conclusions and suggest future research steps in Section 4.

# 2. DECISION TREE INDUCTION ALGORITHMS

Automatically generating rules in the form of decision trees has been object of study of most research fields in which data exploration techniques have been developed [14]. Disciplines such as engineering (pattern recognition), statistics, decision theory, and more recently artificial intelligence (machine learning) have a large number of works dedicated to the generation and application of decision trees. In statistics, we can trace the origins of decision trees in works which proposed building binary segmentation trees for understanding the relationship between predictors and dependent variable. Some examples are AID [15] and CHAID [16]. Decision trees, and induction methods in general, arose in machine learning to avoid the knowledge acquisition bottleneck for expert systems [14].

Specifically regarding top-down induction of decision trees (by far the most popular approach of decision tree induction), Hunt's Concept Learning System (CLS) [17] can be regarded as the pioneering work for inducing decision trees. Systems that directly descend from Hunt's CLS are ID3 [18], ACLS [19] and Assistant [20].

In a higher level of abstraction, Hunt's algorithm can be recursively defined in only two steps. Let $\mathbf{X_t}$ be the set of training instances associated with node $t$ and $y = \{y_1, y_2, ..., y_k\}$ be the class labels in a $k$-class problem [21]: (i) if all the instances in $\mathbf{X_t}$ belong to the same class $y_t$ then $t$ is a leaf node labeled as $y_t$; (ii) if $\mathbf{X_t}$ contains instances that belong to more than one class, an attribute test condition is selected to partition the instances into subsets. A child node is created for each outcome of the test and the instances in $\mathbf{X_t}$ are distributed to the children based on the outcomes. Recursively apply the algorithm to each child.

Hunt's simplified algorithm is the basis for all current top-down decision tree induction algorithm. Nevertheless, its assumptions are too stringent for practical use. For instance, it would only work if every combination of attribute values is present in the training data, and if the training data is inconsistency-free (each combination has a unique class label).

Hunt's algorithm was improved in many ways. Its stopping criterion, for example, as expressed in *step 1*, requires all leaf nodes to be pure (i.e., belonging to the same class). In most practical cases, this constraint leads to enormous decision trees, which tend to suffer from *overfitting*. Possible solutions to overcome this problem is prematurely stopping the tree growth when a minimum level of impurity is reached, or performing a *pruning* step after the tree has been fully grown. Another design issue is how to select the attribute test condition to partition the instances into smaller subsets. In Hunt's original approach, a cost-driven function was responsible for partitioning the tree. Subsequent algorithms such as ID3 [18] and C4.5 [4] make use of information theory based functions for partitioning nodes in purer subsets. Next, we present a review of functions for partitioning nodes in a decision tree.

## 2.1 Selecting splits

A major issue in top-down induction of decision trees is which attribute(s) to choose for splitting a node in subsets. For the case of *axis-parallel* decision trees (also known as *univariate*), the problem is to choose the attribute that better discriminates the input data. A decision rule based on such an attribute is thus generated, and the input data is filtered according to the outcomes of this rule. For *oblique* decision trees (also known as *multivariate*), the goal is to find a combination of attributes with good discriminatory power. Either way, both strategies are concerned with ranking attributes quantitatively.

### 2.1.1 Univariate Splitting Criteria

The most well-known univariate criteria are based, directly or indirectly, on Shannon's entropy [22]. Entropy is known to be a unique function which satisfies the four axioms of uncertainty. It represents the average amount of information when coding each class into a codeword with ideal length according to its probability.

The first splitting criterion that arose based on entropy is the *global mutual information* (GMI) [23]. Following, the well-known *information gain* [18] became a standard after appearing in algorithms such as ID3 [18] and Assistant [20]. It belongs to the class of the so-called *impurity-based criteria*. Quinlan [18] acknowledges the fact that the information gain is biased towards attributes with many values. He proposes a solution for this matter called *gain ratio* [4]. It basically consists of normalizing the information gain by the entropy of the attribute being tested. Nevertheless, the gain ratio has two deficiencies: (i) it may be undefined (i.e., the value of self-entropy may be zero); and (ii) it may choose attributes with very low self-entropy but not with high gain. For solving these issues, Quinlan suggests first calculating the information gain for all attributes, and then calculating the gain ratio only for those cases in which the information gain value is above the average value of all attributes.

Several variations of the gain ratio have been proposed (e.g., normalized gain [24], average gain [25], etc.). Alternatives to entropy-based criteria are the class of distance-based measures, i.e., criteria that evaluate separability, divergency or discrimination between classes. Examples are the *Gini Index* [3], the *twoing criteria* [3], the *Kolmogorov-Smirnoff distance* [26], among others.

### 2.1.2 Multivariate Splits

Decision trees with multivariate splits (known as *oblique*, *linear* or *multivariate* decision trees) are not so popular as the univariate ones, mainly because they are harder to interpret. Nevertheless, researchers reckon that multivariate splits can improve the performance of the tree in several data sets, while generating smaller trees [2]. Clearly, there is a tradeoff to consider in allowing multivariate tests: simple

tests may result in large trees that are hard to understand, yet multivariate tests may result in small trees with tests that are hard to understand [27].

A decision tree with multivariate splits is able to produce polygonal (polyhedral) partitions of the attribute space (hyperplanes at an oblique orientation to the attribute axes) whereas univariate trees can only produce hyper-rectangles parallel to the attribute axes. The tests at each node have the form $w_0 + \sum_{i=1}^{n} w_i a_i(x) \leq 0$, where $w_i$ is a real-valued coefficient associated to the $i^{th}$ attribute and $w_0$ the disturbance coefficient of the test.

CART (Classification and Regression Trees) [3] is one of the first systems that allowed multivariate splits. It employs a hill-climbing strategy with a backward attribute elimination for finding good (albeit suboptimal) linear combinations of attributes in non-terminal nodes. It is a fully-deterministic algorithm with no built-in mechanisms to escape local-optima. Breiman et al. [3] point out that the proposed algorithm has much room for improvement.

Another approach for building oblique decision trees is LMDT (Linear Machine Decision Trees) [28]. Each non-terminal node holds a linear machine, which is a set of $k$ linear discriminant functions that are used collectively to assign an instance to one of the $k$ existing classes. LMDT uses heuristics to determine when a linear machine has stabilized (since convergence cannot be guaranteed). More specifically, for handling non-linearly separable problems, a method similar to simulated annealing (SA) is used (called *thermal training*).

SADT (Simulated Annealing of Decision Trees) [29] is a system that employs SA for finding good coefficient values for attributes in non-terminal nodes of decision trees. First, it places a hyperplane in a canonical location, and then iteratively perturbs the coefficients in small random amounts guided by the SA algorithm. Although SADT can eventually escape from local-optima, its efficiency is compromised since it may consider tens of thousands of hyperplanes in a single node during annealing.

OC1 (Oblique Classifier 1) [30] is yet another oblique decision tree system. It is a thorough extension of CART's oblique decision tree strategy. OC1 presents the advantage of being more efficient than the previously described systems. It searches for the best univariate split as well as the best oblique split, and it only employs the oblique split when it improves over the univariate split. It uses both a deterministic heuristic search (as employed in CART) for finding local-optima and a non-deterministic search (as employed in SADT - though not SA) for escaping local-optima.

Ittner [31] proposes using OC1 over an augmented attribute space, generating *non-linear decision trees*. The key idea involved is to "build" new attributes by considering all possible pairwise products and squares of the original set of $n$ attributes.

Shah and Sastry [32] propose the APDT (Alopex Perceptron Decision Tree) system. It is an oblique decision tree inducer that makes use of a new splitting criterion, based on the level of non-separability of the input instances. They argue that because oblique decision trees can realize arbitrary piecewise linear separating surfaces, it seems better to base the evaluation function on the degree of separability of the partitions rather than on the degree of purity of them. APDT runs the Perceptron algorithm for estimating the number of non-separable instances belonging to each one of the binary partitions provided by an initial hyperplane. Then, a correlation-based optimization algorithm called Alopex is employed for tuning the hyperplane weights taking into account the need of minimizing the new split criterion based on the degree of separability of partitions. Shah and Sastry [32] also propose a pruning algorithm based on genetic algorithms.

For the interested reader, it is worth mentioning that there are methods that induce oblique decision trees with optimal hyperplanes, discovered through *linear programming* [6]. Though these methods can find the optimal hyperplanes for specific splitting measures, the size of the linear program grows very fast with the number of instances and attributes.

## 2.2 Stopping Criteria

The top-down induction of a decision tree is recursive and it continues until a stopping criterion (or some stopping criteria) is satisfied. Some popular stopping criteria (also known as *pre-pruning*) are [2]:

1. Reaching class homogeneity: when all instances that reach a given node belong to the same class, there is no reason to split this node any further;
2. Reaching attribute homogeneity: when all instances that reach a given node have the same attribute values (though not necessarily the same class value);
3. Reaching the maximum tree depth: a parameter *tree depth* can be specified to avoid deep trees;
4. Reaching the minimum number of instances for a non-terminal node: a parameter *minimum number of instances for a non-terminal node* can be specified to avoid (or at least alleviate) the data fragmentation problem;
5. Failing to exceed a threshold when calculating the splitting criterion: a parameter *splitting criterion threshold* can be specified for avoiding *weak* splits.

## 2.3 Pruning

Pruning (also referred as post-pruning) is usually performed in decision trees for enhancing tree comprehensibility (by reducing its size) while maintaining (or even improving) accuracy. It was originally conceived as a strategy for tolerating noisy data, though it was found that it could improve decision tree accuracy in many noisy data sets [3, 18, 33].

A pruning method receives as input an unpruned tree $T$ and outputs a decision tree $T'$ formed by removing one or more subtrees from $T$. It replaces non-terminal nodes with leaf nodes according to a given heuristic. Next, we present the five most well-known pruning methods for decision trees [11]: 1) reduced-error pruning; 2) pessimistic error pruning; 3) minimum error pruning; 4) cost-complexity pruning; and 5) error-based pruning.

Reduced-error pruning (REP) is a conceptually simple strategy proposed by Quinlan [33]. It uses a pruning set (a part of the training set) to evaluate the goodness of a given subtree from $T$. The idea is to evaluate each non-terminal node $t$ with regard to the classification error in the pruning set. If such an error decreases when we replace the subtree $T^{(t)}$ rooted on $t$ by a leaf node, then $T^{(t)}$ must be pruned.

Quinlan imposes a constraint: a node $t$ cannot be pruned if it contains a subtree that yields a lower classification error in the pruning set. The practical consequence of this constraint is that REP should be performed in a bottom-up fashion. The REP pruned tree $T'$ presents an interesting optimality property: it is the smallest most accurate tree resulting from pruning original tree $T$ [33]. Besides this optimality property, another advantage of REP is its linear complexity, since each node is visited only once in $T$. An obvious disadvantage is the need of using a pruning set, which means one has to divide the original training set, resulting in less instances to grow the tree. This disadvantage is particularly serious for small data sets.

Also proposed by Quinlan [33], the pessimistic error pruning (PEP) uses the training set for both growing and pruning the tree. The apparent error rate, i.e., the error rate calculated over the training set, is optimistically biased and cannot be used to decide whether pruning should be performed or not. Quinlan thus proposes adjusting the apparent error according to the continuity correction for the binomial distribution in order to provide a more realistic error rate. PEP is computed in a top-down fashion, and if a given node $t$ is pruned, its descendants are not examined, which makes this pruning strategy quite efficient in terms of computational effort. As a point of criticism, Esposito et al. [12] point out that the introduction of the continuity correction in the estimation of the error rate has no theoretical justification, since it was never applied to correct over-optimistic estimates of error rates in statistics.

Originally proposed in [34] and further extended in [35], minimum error pruning (MEP) is a bottom-up approach that seeks to minimize the *expected error rate* for unseen cases. It uses an ad-hoc parameter $m$ for controlling the level of pruning. Usually, the higher the value of $m$, the more severe the pruning. Cestnik and Bratko [35] suggest that a domain expert should set $m$ according to the level of noise in the data. Alternatively, a set of trees pruned with different values of $m$ could be offered to the domain expert, so he/she can choose the best one according to his/her experience.

Cost-complexity pruning (CCP) is the post-pruning strategy of the CART system, detailed in [3]. It consists of two steps: (i) generate a sequence of increasingly smaller trees, beginning with $T$ and ending with the root node of $T$, by successively pruning the subtree yielding the lowest *cost complexity*, in a bottom-up fashion; (ii) choose the best tree among the sequence based on its relative size and accuracy (either on a pruning set, or provided by a cross-validation procedure in the training set). The idea within step 1 is that pruned tree $T_{i+1}$ is obtained by pruning the subtrees that show the lowest increase in the apparent error (error in the training set) per pruned leaf. Regarding step 2, CCP chooses the smallest tree whose error (either on the pruning set or on cross-validation) is not more than one standard error (SE) greater than the lowest error observed in the sequence of trees.

Finally, error-based pruning (EBP) was proposed by Quinlan and it is implemented as the default pruning strategy of C4.5 [4]. It is an improvement over PEP, based on a far more pessimistic estimate of the expected error. Unlike PEP, EBP performs a bottom-up search, and it performs not only the replacement of non-terminal nodes by leaves but also *grafting* of subtree $T^{(t)}$ onto the place of parent $t$. For deciding whether to replace a non-terminal node by a leaf (subtree replacement), to graft a subtree onto the place of its parent (subtree raising) or not to prune at all, a pessimistic estimate of the expected error is calculated by using an upper confidence bound. An advantage of EBP is the new *grafting* operation that allows pruning useless branches without ignoring interesting lower branches. A drawback of the method is the presence of an ad-hoc parameter, $CF$. Smaller values of $CF$ result in more pruning.

## 2.4 Missing values

Handling missing values is an important task in decision tree induction. Missing values can be an issue during tree induction and also during classification. During tree induction, there are two moments in which we need to deal with missing values: splitting criterion evaluation and instances splitting.

During the split criterion evaluation in node $t$ based on attribute $a_i$, some common strategies are: (i) ignore all instances whose value of $a_i$ is missing [36, 3]; (ii) imputation of missing values with the mode (nominal attributes) or the mean/median (numeric attributes) of all instances in $t$ [37]; (iii) weight the splitting criterion value (calculated in node $t$ with regard to $a_i$) by the proportion of missing values [38]; and (iv) imputation of missing values with the mode (nominal attributes) or the mean/median (numeric attributes) of all instances in $t$ whose class attribute is the same of the instance whose $a_i$ value is being imputed [39].

For deciding which child node training instance $x_j$ should go to, considering a split in node $t$ over $a_i$, some possibilities are: (i) ignore instance $x_j$ [18]; (ii) treat instance $x_j$ as if it has the most common value of $a_i$ (mode or mean/median) [38]; (iii) assign instance $x_j$ to all partitions [36]; (iv) build an exclusive partition for missing values [38]; and (v) create a *surrogate split* for each split in the original tree based on a different attribute [3] - for instance, a split over attribute $a_i$ will have a surrogate split over attribute $a_j$, given that $a_j$ is the attribute which most resembles the original split.

Finally, for classifying an unseen test instance $x_j$, considering a split in node $t$ over $a_i$, some alternatives are: (i) explore all branches of $t$ combining the results [40]; (ii) treat instance $x_j$ as if it has the most common value of $a_i$ (mode or mean/median); (iii) halt the classification process and assign instance $x_j$ to the majority class of node $t$ [38].

# 3. EVOLVING FULL DECISION TREE INDUCTION ALGORITHMS

We have presented in Section 2 the main design choices one has to face when designing a new top-down decision tree induction algorithm. For the past 40 years, researchers have attempted to improve decision tree induction algorithms, either by proposing new splitting criteria for internal nodes, by investigating pruning strategies for avoiding overfitting, or even by discovering new approaches for dealing with missing values. Each new top-down decision tree induction algorithm presents a combination of these strategies, which were chosen in order to maximize performance in empirical analyses. Nevertheless, the number of different strategies for the several components of a decision tree algorithm is so vast after these 40 years of research that it would be impracticable for a human being to test all possibilities with the purpose of achieving the best performance in a given

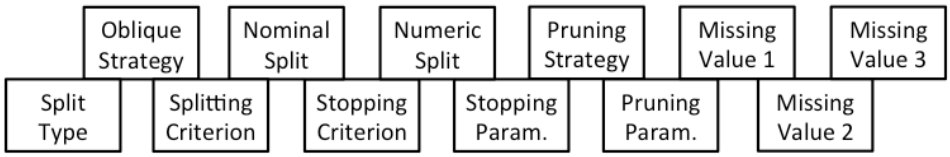


**Figure 1: A possible linear genome for top-down decision tree induction algorithms.**

data set (or in a set of data sets). Hence, we pose an important question for researchers in the area: "how can we automate the design of a decision tree induction algorithm?"

The answer for this question arose with the work of Pappa and Freitas [13], which proposes the automatic design of rule induction algorithms through an evolutionary algorithm. In their work, Pappa and Freitas propose using a grammar-based genetic programming approach for building and evolving individuals which are, in fact, rule induction algorithms. The use of evolutionary algorithms for generating decision trees has been extensively studied (e.g., [8, 9]), but to the best of our knowledge, no work has proposed an evolutionary algorithm to evolve decision tree *algorithms*.

A recent approach called HHDT (Hyper-Heuristic Decision Tree) [41] proposes an evolutionary algorithm for evolving heuristic rules in order to determine the best splitting criterion to be used in non-terminal nodes. Whereas this approach is a first step to automate decision tree induction algorithms, it evolves a single component of the algorithm (the choice of splitting criterion), and thus should be further extended for being able to generate full decision tree induction algorithms.

In this paper, we propose two distinct evolutionary approaches to evolve complete decision tree induction algorithms. The first one is based on a linear genome representation, in which each gene represents a design component of the algorithm, i.e., a *building block* (e.g., splitting criterion, post-pruning strategy, etc.). In the second, we make use of a grammar-based genetic programming approach, where a grammar is used for generating coherent decision tree algorithms. We present both approaches in the next sections, as well as some guidelines for designing the fitness function strategy for the evolutionary algorithm.

## 3.1 Linear Genome Representation

The individuals of the evolutionary algorithm can be represented as a linear genome, in which each gene is a building block (a design component) of the decision tree induction algorithm. Figure 1 presents one possible linear genome for coding a top-down decision tree induction algorithm.

In the linear genome proposed in Figure 1, we have 12 genes, each one representing a design component or parameter of a decision tree induction algorithm. Gene *split type* can be indexed by three integers, indicating whether the tree will hold univariate, multivariate or mixed nodes. Gene *oblique strategy* can also be indexed by integers, each one indicating the weight definition strategy for the case of multivariate splits (this gene can be ignored if *split type* is set to univariate). Gene *splitting criterion* defines which criterion will be used for splitting nodes (an integer for each criterion). Genes *nominal split* and *numeric split* indicate the strategies for partitioning nominal (numeric) attributes (e.g., an edge for each attribute category (interval) of a nominal (numeric) attribute). Gene *stopping criterion* indicates the strategy used for pre-pruning, and *stopping parameter* is the associated parameter (e.g., the stopping criterion = minimum number of instances, and stopping parameter = 10). Similarly, genes *pruning strategy* and *pruning parameter* define the pruning procedure and its corresponding parameter. Finally, three building blocks for missing value strategies (one for splitting evaluation, one for partitioning instances and one for instance classification).

One possible individual encoded as a linear genome is $[2, 4, 1, 1, 2, 3, 0.1, 4, 2.0, 0, 1, 0]$, which could account for [mixed nodes, hill-climbing with randomization, gain ratio, one-edge-for-category, binary-split, splitting criterion threshold, 0.1, EBP, 0.33, ignore-instances, assign-to-all, explore-all-branches], representing a mixed-decision tree with a hill-climbing procedure for generating weights for multivariate splits; gain ratio as a splitting measure; one edge for each category of a nominal attribute and a binary split of numeric attributes; the growth of the tree stops if the gain is inferior to 0.1; an error-based pruning with $CF = 33\%$ is performed; and the missing values strategies are: ignore missing values for calculating the splitting criterion; assign the instances with missing values to every child node during training; and explore all branches when the attribute value is missing and choose the class-value with greater associated probability during classification. It is easy to notice that the linear genome representation proposed in Figure 1 can generate algorithms such as C4.5 [4], CART [3] and potentially many new decision tree induction algorithms (infinite possibilities if we consider the stopping parameter and pruning parameter as real numbers).

## 3.2 Grammar-based Representation

Another possibility of generating individuals for an evolutionary algorithm that evolves generic decision tree algorithms is through a *grammar*. Grammar-based genetic programming is a specific type of genetic programming in which we can guarantee that individuals are syntactically correct and, in addition, we can add *prior* knowledge of the task at hand. In the context of decision tree induction, we can design a grammar that includes some well-known strategies for decision tree induction, though it is important we keep it flexible enough so it can generate potentially new and effective algorithms not previously idealized by researchers. Figure 2 presents one possible grammar for generating decision tree induction algorithms.

The proposed grammar has 29 non-terminals (NTs). The first non-terminal refers to the growth and (optional) pruning of the decision tree. NT #2 refers to the recursive growth of a decision tree until a stopping criterion is satisfied

(NTs #3, #4, #5, #6). NT #7 allows the creation of a leaf either by associating the most frequent class to it or by allowing a rare class (NT #8) to be selected (a counter-intuitive action in balanced problems, but maybe an interesting solution for imbalanced data sets). NT #9 allows the creation of internal nodes, by selecting: (i) an appropriate missing value strategy for usage during splitting evaluation (NT #10); (ii) a splitting criterion (NT #11) and an optional stopping criterion (NT #17); (iii) a missing value strategy for partitioning instances (NT #18); and (iv) the recursive call for growing the tree. NT #11 allows choosing among univariate trees (NT #12) , oblique trees (NTs #13 and #14) and omni (mixed) trees (NTs #15 and #16). NT #19 offers options of post-pruning strategies (NTs #20-28) and NT #29 allows to choose the missing value strategy for classifying a new instance.

Figure 3 presents a derivation tree of a possible individual generated by this grammar. The grammar can be further extended to include new strategies with minor effort. It can be seen as an approach that controls the initial population so every individual is syntactically correct. In order to guarantee that further evolved individuals keep their syntactic correctness, the genetic operators employed must be specially designed in order to avoid the generation of useless solutions.

At the same time, the grammar should be flexible enough to allow seemingly counter-intuitive actions that a human researcher would not allow. For instance, the grammar in Figure 2 allows leaf nodes to hold a class label which is not the one of the majority of training instances. This is counter-intuitive since the sole purpose of training a classifier is to take advantage of the available knowledge. Nevertheless, if we have a scenario of imbalanced classes, the evolutionary algorithm may end up evolving an algorithm that successfully detects the rare class based on this counter-intuitive action. Grammar flexibility is key for discovering new and potentially useful decision tree algorithms.

## 3.3 Guidelines for Fitness Evaluation

To evaluate generic decision tree induction algorithms, we can analyze both the algorithm itself (individual being evolved) and the decision tree it generates. For the former, we can evaluate aspects such as time complexity and structural complexity of the algorithm. We can penalize algorithms whose components are costly by assigning costs to the use of each component. For instance, a decision tree algorithm that performs multivariate splits whose weights are defined through a genetic algorithm, and that also employs cost-complexity pruning is far more costly in terms of computational effort than an algorithm that performs univariate splits and pessimistic-error pruning. We can choose to give greater importance to time complexity when large data sets are being employed, or lesser importance when data sets are smaller or when predictive performance is crucial.

Regarding the decision tree generated by the evolved algorithm, we can evaluate it with respect to the accuracy it obtains, or alternatively the F-Measure (both measures to be maximized), number of leaves and total number of nodes (measures to be minimized), among several other possibilities. Parsimony pressure can be controlled also according to the needs of the end-user.

```
1) <Start> ::= <Top-Down-Growing> [<PostPruning>]
<MissingValueStrategy>
2)<Top-Down-Growing> ::= if <StoppingCriteria> then <CreateLeaf>
else <CreateInternal>
3) <StoppingCriteria> ::= homogeneousTrainSet |
<MaxTreeDepthReached> | <MinInstancesReached> |
<TrainAccThresholdReached>
4) <MaxTreeDepthReached> ::= treeDepth (>2 | >5 | >10 | >15 |
>20)
5) <MinInstancesReached> ::= minNumInstances ( = 1 | = 2 | = 3 |
= 4 | = 5 | < 10 | < 15 | < 20) | minPercInstances ( <= 0.01 |
<= 0.03 | <= 0.05 | <= 0.1)
6) <TrainAccThresholdReached> ::= trainAcc (>= 0.8| >= 0.9 | >=
0.95 | >= 0.99 | >= 0.999)
7) <CreateLeaf> ::= AssignFrequentClassToLeaf |
<RareClassConditional>
8) <RareClassConditional> ::= if (NumRareClassInstances (>
1 | > 2 | > 3 | > 4 | > 5) | PercRareClass ( > 0.001 | >
0.005 | > 0.01 | > 0.05)) then AssignRareClassToLeaf else
AssignFrequentClassToLeaf
9) <CreateInternal> ::= <MissingValuesSplit> <SplitCriterion>
[if <SplittingCriterionMinValue> then <CreateLeaf>
else] <MissingValuePartition> for eachSubtreeGenerated
recursiveCall-Top-Down-Growing endFor
10) <MissingValueSplit> ::= FriedmanBreiman | ClarkNiblett |
Quinlan1989 | LohShih
11) <SplitCriterion> ::= <UnivariateSplit> | <MultivariateSplit>
| <OmniSplit>
12) <UnivariateSplitCriterion> ::= twoing | orthogonality |
Kolmogorov-Smirnov | informationGain | Gini-Index | gainRatio
| GMI | CAIR | GStatistics | ChiStatistics | NormalizedGain |
MantarasDistance | ORT | MPI | TAO | Permutation | P0 | CV |
DCSM
13) <MultivariateSplit> ::= <DefineHyperplaneWeights>
<UnivariateSplitCriterion>
14) <DefineHyperplaneWeights> ::= Hill-ClimbingSBE
| Hill-ClimbingRandomization | SimulatedAnnealing |
MLPNeuralNetwork | GeneticAlgorithm
15) <OmniSplit> ::= RandomChoice | <DataBasedChoice>
16) <DataBasedChoice> ::= if <MinInstancesReached> then
<UnivariateSplit> else <MultivariateSplit>
17) <SplittingCriterionMinValue> ::= NormalizedCriterionValue
(<= 0.01 | <= 0.05 | <= 0.1 | <= 0.2)
18) <MissingValuePartition> ::= Quinlan1986 | Quinlan1989-1 |
Quinlan1989-2 | Kononenko | Friedman | LohShih | Breiman
19) <PostPruning> ::= <ReducedError> | <PessimisticError>
| <MinimumError> | <CriticalValue> | <CostComplexity> |
<ErrorBased>
20) <ReducedError> ::= <PruningSet>
21) <PruningSet> ::= sizeOfPruningSet( 0.1 | 0.2 | 0.3 | 0.4 |
0.5) trainingSet
22) <PessimisticError> ::= 0.5SE | 1SE | 2SE | 3SE
23) <MinimumError> ::= valueOf-m (0.5 | 1 | 2 | 3 | 4 | 5 | k |
10 | 20)
24) <CriticalValue> ::= for i=0 to <SplittingCriterionMinValue>
pruneTree(i) endFor <SelectBestTree>
25) <SelectBestTree> ::= <PruningSet> (highestAccuracyOfAll |
ParetoOptimal)
26) <CostComplexity> ::= (<PruningSet> | <Cross-Validation>)
(0SE | 0.5SE | 1SE)
27) <Cross-Validation> ::= (2Fold | 3Fold | 4Fold | 5Fold |
10Fold | 20Fold)
28) <ErrorBased> ::= CF(0.05 | 0.1 | 0.15 | 0.2 | 0.25 | 0.3 |
0.35 | 0.4 | 0.45 | 0.5)
29) <MissingValueStrategy> ::= Quinlan1987 | Quinlan1989 |
ReplaceByMeanOrMode
```

**Figure 2: A possible grammar for generating top-down decision tree induction algorithms.**

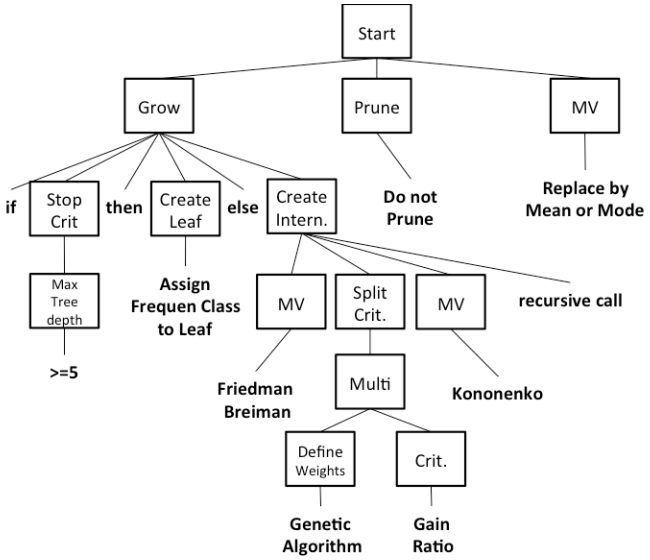


**Figure 3: Derivation tree of a possible individual generated by the proposed grammar.**

Even though our main purpose is to evolve generic decision tree induction algorithms, an interesting possibility is to evolve algorithms tailored for a given application domain, or for a particular statistical shape of data sets. We can bias the evolutionary algorithm to generate particular solutions by designing more specialized fitness functions. In domains such as modeling the expressive performance in music or in ordinal classification, accuracy or other traditional measures may not be well-suited for evolving specialized algorithms.

For evolving a decision tree induction algorithm tailored for data sets with a particular *statistical shape*, we need a set of data sets (a *meta-training set*) that share structural similarities. An interesting idea is to use *geometrical complexity measures*, such as those presented in [42], for analyzing the degree of similarity between data sets. It is argued that the difficulty of a classification problem is strongly related to its *geometrical shape*, and not so much to sample size and dimensionality [42]. Hence, we can select a meta-training set which is geometrically similar for evolving algorithms suited for a particular degree of complexity.

For evolving generic algorithms, an appropriate strategy is to select very distinct data sets, and to evaluate the average performance of the decision trees generated in these data sets. Since we have to select a different set of data sets for evolving the algorithms (meta-training set) and evaluating their relative performance (meta-testing set), some strategies for improving computational effort are desired, such as designing parallel/distributed environments or randomly subsampling data sets.

# 4. FINAL REMARKS

To the best of our knowledge, this paper is the first work to present an approach for automatically designing full decision tree induction algorithms. This ambitious task is strongly inspired by the pioneering work of Pappa and Freitas [13], which proposes a genetic programming algorithm to evolve rule induction algorithms. Since Pappa and Freitas managed to successfully evolve rule induction algorithms, we believe the same can be achieved with decision tree induction algorithms, regardless of it being a more complex task.

We have proposed in this paper two possible individual representations for an evolutionary algorithm that evolves full decision tree induction algorithms. In the first one, each individual is encoded as a linear genome, in which each gene is either a major design component of the decision tree algorithm or one of its parameters. Genes can take either integer values for indexing alternative strategies for each major component, or real values in the case of specific parameters. We have shown that the linear genome approach is comprehensive enough to generate classic decision tree induction algorithms such as C4.5 [4] and CART [3]. The second approach is a genetic programming algorithm supported by a grammar, so individuals can be generated in the form of syntactically correct derivation trees. Special care has to be taken regarding genetic operators such as crossover and mutation so individuals can keep a valid and coherent structure.

We have also proposed a few guidelines regarding the fitness function of an evolutionary algorithm that evolves decision tree induction algorithms. For instance, we suggest that both the algorithm and the trees generated be evaluated within a multi-objective fitness function. Regarding the algorithm, time complexity can be estimated if we assign costs to each one of the components that form the individual. Traditional classification measures can be used to evaluate the trees resulting from the individuals in multiple data sets, and then combined through an arithmetic or weighted average. We also suggest the use of specialized measures for evolving algorithms tailored for a given domain. If we are interested in data sets that share structural similarities, we propose using geometrical complexity measures [42] for deciding which data sets to be used as meta-training sets. Wide-use generic decision tree induction algorithms can be evolved by selecting a very heterogeneous set of data sets to be part of the meta-training set. Since we are interested in using several data-sets as meta-training set, parallel/distributed solutions should be used in order to allow the evolution of algorithms in a reasonable amount of time. A second option would be random subsampling.

This work accounts for the beginning of a project for automatically evolving decision tree induction algorithms efficiently and effectively. Some of our next steps include refining the design of the evolutionary algorithm, implementing both strategies depicted in this paper and evaluating their performance. We plan to extensively compare several of our automatically-designed algorithms to human-designed ones in public data sets. In addition, we intend to evolve algorithms tailored for specific domains such as gene array expression, oil and gas discovery and prediction of health indicators. We also intend to develop evolutionary algorithms to automatically design regression and model tree induction algorithms.

# 5. REFERENCES

[1] (2007). [Online]. Available: http://www.kdnuggets.com/polls/2007/data_mining_methods.htm

[2] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers - a survey," *IEEE T SYST MAN CY C*, vol. 35, no. 4, pp. 476 – 487, 2005.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.

[4] J. R. Quinlan, *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann, 1993.

[5] G. Landeweerd, T. Timmers, E. Gelsema, M. Bins, and M. Halie, "Binary tree versus single level tree classification of white blood cells," *PATTERN RECOGN*, vol. 16, no. 6, pp. 571–577, 1983.

[6] K. Bennett and O. Mangasarian, "Multicategory discrimination via linear programming," *OPTIM METHOD SOFTW*, vol. 2, pp. 29–39, 1994.

[7] B. Kim and D. Landgrebe, "Hierarchical classifier design in high-dimensional numerous class cases," *IEEE T GEOSCI REMOTE*, vol. 29, no. 4, pp. 518–528, 1991.

[8] M. Basgalupp, A. Carvalho, R. Barros, D. Ruiz, and A. Freitas, "Lexicographic multi-objective evolutionary induction of decision trees," *INT J BIOINSP COMPUT*, vol. 1, no. 1/2, pp. 105–117, 2009.

[9] R. Barros, D. Ruiz, and M. Basgalupp, "Evolutionary model trees for handling continuous classes in machine learning," *INFORM SCIENCES*, vol. 181, pp. 954–971, 2011.

[10] L. Breiman, "Random forests," *MACH LEARN*, vol. 45, no. 1, pp. 5–32, 2001.

[11] L. Breslow and D. Aha, "Simplifying decision trees: A survey," *KNOWL ENG REV*, vol. 12, no. 01, pp. 1–40, 1997.

[12] F. Esposito, D. Malerba, and G. Semeraro, "A Comparative Analysis of Methods for Pruning Decision Trees," *IEEE T PATTERN ANAL*, vol. 19, no. 5, pp. 476–491, 1997.

[13] G. Pappa and A. Freitas, "Automatically evolving rule induction algorithms," in *17th European Conference on Machine Learning*, 2006, pp. 341–352.

[14] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *DATA MIN KNOWL DISC*, vol. 2, no. 4, pp. 345–389, 1998.

[15] J. A. Sonquist, E. L. Baker, and J. N. Morgan, "Searching for structure," Institute for Social Research, University of Michigan, Tech. Rep., 1971.

[16] G. V. Kass, "An exploratory technique for investigating large quantities of categorical data," *APPL STATIST*, vol. 29, no. 2, pp. 119–127, 1980.

[17] E. B. Hunt, J. Marin, and P. J. Stone, *Experiments in induction*. New York, NY, USA: Academic Press, 1966.

[18] J. R. Quinlan, "Induction of decision trees," *MACH LEARN*, vol. 1, no. 1, pp. 81–106, 1986.

[19] A. Patterson and T. Niblett, *ACLS user manual*, Glasgow: Intelligent Terminals Ltd, 1983.

[20] I. Kononenko, I. Bratko, and E. Roskar, "Experiments in automatic learning of medical diagnostic rules," Jozef Stefan Institute, Ljubljana, Yugoslavia, Tech. Rep., 1984.

[21] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison-Wesley, 2005.

[22] C. E. Shannon, "A mathematical theory of communication," *BELL SYST TECH*, vol. 27, no. 1, pp. 379–423, 625–56, 1948.

[23] M. Gleser and M. Collen, "Towards automated medical decisions," *COMPUT BIOMED RES*, vol. 5, no. 2, pp. 180–189, 1972.

[24] B. Jun, C. Kim, Y.-Y. Song, and J. Kim, "A New Criterion in Selection and Discretization of Attributes for the Generation of Decision Trees," *IEEE T PATTERN ANAL*, vol. 19, no. 2, pp. 1371–1375, 1997.

[25] D. Wang and L. Jiang, "An improved attribute selection measure for decision tree induction," in *4TH INT CONF FUZZY KNOWL DISC*, 2007, pp. 654–658.

[26] E. M. Rounds, "A combined nonparametric approach to feature selection and binary decision tree design," *PATTERN RECOGN*, vol. 12, no. 5, pp. 313–317, 1980.

[27] P. Utgoff and C. Brodley, "An incremental method for finding multivariate splits for decision trees," in *7th INT CONF MACH LEARN*, 1990, pp. 58–65.

[28] P. E. Utgoff and C. E. Brodley, "Linear machine decision trees," University of Massachusetts, Dept of Comp Sci, Tech. Rep., 1991.

[29] D. Heath, S. Kasif, and S. Salzberg, "Induction of oblique decision trees," *J ARTIF INTELL RES*, vol. 2, pp. 1–32, 1993.

[30] S. K. Murthy, S. Kasif, S. Salzberg, and R. Beigel, "OC1: A Randomized Induction of Oblique Decision Trees," in *AAAI*, 1993, pp. 322–327.

[31] A. Ittner, "Non-linear decision trees-NDT," in *13th INT CONF MACH LEARN*, 1996, pp. 1–6.

[32] S. Shah and P. Sastry, "New algorithms for learning and pruning oblique decision trees," *IEEE T SYST MAN CY C*, vol. 29, no. 4, pp. 494 –505, 1999.

[33] J. R. Quinlan, "Simplifying decision trees," *INT J MAN-MACH STUD*, vol. 27, pp. 221–234, 1987.

[34] T. Niblett and I. Bratko, "Learning decision rules in noisy domains," in *6th ANN TECH CONF RES DEV EXPER SYST III*, 1986, pp. 25–34.

[35] B. Cestnik and I. Bratko, "On estimating probabilities in tree pruning," in *MACH LEARN - EWSL'91*. Springer Berlin/Heidelberg, 1991, pp. 138–150.

[36] J. H. Friedman, "A recursive partitioning decision rule for nonparametric classification," *IEEE T COMPUT*, vol. 100, no. 4, pp. 404–408, 1977.

[37] P. Clark and T. Niblett, "The CN2 induction algorithm," *MACH LEARN*, vol. 3, no. 4, pp. 261–283, 1989.

[38] J. R. Quinlan, "Unknown attribute values in induction," in *6tH INT MACH LEARN WORK*, 1989, pp. 164–168.

[39] W. Loh and Y. Shih, "Split selection methods for classification trees," *STAT SINICA*, vol. 7, pp. 815–840, 1997.

[40] J. R. Quinlan, "Decision trees as probabilistic classifiers," in *4th INT MACH LEARN WORK*, 1987.

[41] A. Vella, D. Corne, and C. Murphy, "Hyper-heuristic decision tree induction," *W CONF NAT BIOINSP COMP*, pp. 409–414, 2010.

[42] T. Ho and M. Basu, "Complexity measures of supervised classification problems," *IEEE T PATTERN ANAL*, vol. 24, no. 3, pp. 289–300, 2002.