A Survey of Evolutionary Algorithms for Decision Tree Induction

Rodrigo C. Barros, Márcio P. Basgalupp, André C. P. L. F. de Carvalho and Alex A. Freitas

***Abstract*—This paper presents a survey of evolutionary algorithms designed for decision tree induction. In this context, most of the paper focuses on approaches that evolve decision trees as an alternate heuristics to the traditional top-down divide-and-conquer approach. Additionally, we present some alternative methods that make use of evolutionary algorithms to improve particular components of decision tree classifiers. The paper original contributions are the following. First, it provides an up-to-date overview that is fully focused on evolutionary algorithms and decision trees and does not concentrate on any specific evolutionary approach. Second, it provides a taxonomy which addresses works that evolve decision trees and works that design decision tree components using evolutionary algorithms. Finally, a number of references is provided that describe applications of evolutionary algorithms for decision tree induction in different domains. The paper ends by addressing some important issues and open questions that can be subject of future research.**

***Index Terms*—Evolutionary algorithms, decision tree induction, soft computing classification, regression.**

I. Introduction

A DECISION tree is a classifier depicted in a flowchart-like tree structure which has been widely used to represent classification models, due to its comprehensible nature that resembles the human reasoning. 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. Besides, the induced model usually presents a good generalization ability [1], [2].

Most decision tree induction algorithms are based on a greedy top-down recursive partitioning strategy for tree growth. They use different variants of impurity measures, like, information gain [3], gain ratio [4], gini-index [5] and distance-based measures [6], to select an input attribute to be associated with an internal node. One major drawback of greedy search is that it usually leads to sub-optimal solutions. Moreover, recursive partitioning of the data set may result in very small data sets for the attribute selection in the deepest nodes of a tree, which in turn may cause data overfitting.

Several alternatives have been proposed to overcome these problems, including the induction of an ensemble of trees. Ensembles are created by inducing different trees from training samples and the ultimate classification is frequently given through a voting scheme (see [7], [8]). However, a disadvantage of ensembles is that the comprehensibility of analyzing a single decision tree is lost. Indeed, classification models being combined in an ensemble are often, to some extent, inconsistent with each other; an inconsistency that is necessary to increase the predictive accuracy of the ensemble [9]. Therefore, ensembles are not a good option for applications where comprehensibility is crucial.

Hence, an approach that has been increasingly used is the induction of decision trees through Evolutionary Algorithms (EAs). Instead of local search, EAs perform a robust global search in the space of candidate solutions. As a result, EAs tend to cope better with attribute interactions than greedy methods [10]. They are essentially algorithms inspired by the principle of natural selection and genetics. In nature, individuals are continuously evolving, adapting to their living environment. In EAs, each "individual" represents a candidate solution to the target problem. Each individual is evaluated by a fitness function, which measures the quality of its corresponding candidate solution. At each generation, the best individuals have a higher probability of being selected for reproduction. The selected individuals undergo operations inspired by genetics, such as crossover and mutation, producing new offspring which will replace the parents, creating a new generation of individuals. This process is iteratively repeated until a stopping criterion is satisfied [11], [12]. Figure I presents a common algorithmic framework for both Genetic Algorithms (GAs) and Genetic Programming (GP), well-known EAs.

1: Create initial population of individuals
2: Compute the fitness of each individual
3: **repeat**
4: Select individuals based on fitness
5: Apply genetic operators to selected individuals, creating new individuals
6: Compute fitness of each new individual
7: Update the current population (new individuals replace previous individuals)
8: **until** (stopping criteria)

Fig. 1. Generic algorithmic framework for both GA and GP [10].

The number of EAs for decision tree induction has grown in the past few years, mainly because they report good predictive accuracies whilst keeping the comprehensibility of decision trees. In this context, we provide a detailed survey of EAs to evolve decision trees for classification (Section IV) and regression (Section V). Furthermore, we discuss EAs designed

R. Barros and A. Carvalho are with the Department of Computer Sciences of the University of São Paulo (USP) at São Carlos, SP, Brazil. E-mails: {rcbarros;andre}@icmc.usp.br.

M. Basgalupp is with Instituto de Ciência e Tecnologia, Universidade Federal de São Paulo, SP, Brazil. E-mail: basgalupp@unifesp.br.

A. Freitas is with the Computer Science Department of the University of Kent at Canterbury, Kent, UK. E-mail: A.A.Freitas@kent.ac.uk.

Manuscript received April XX, 20XX; revised January XX, 20XX.

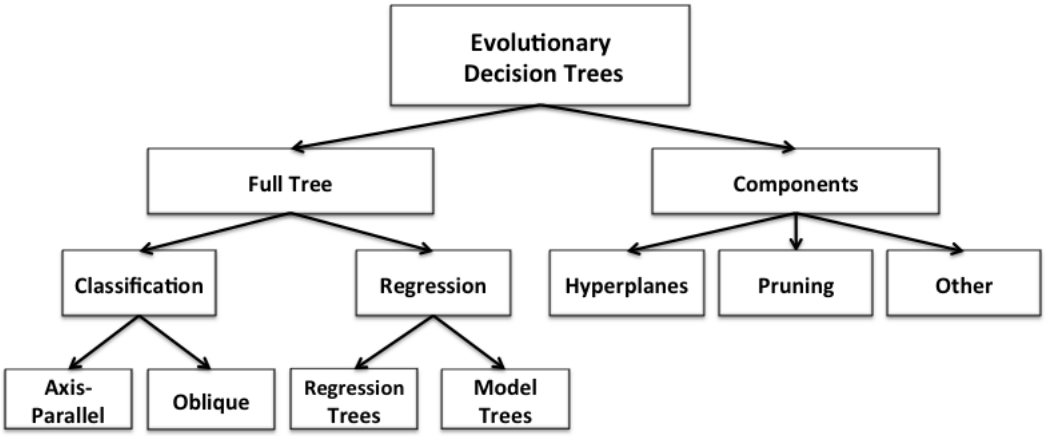

Fig. 2. Taxonomy of evolutionary algorithms for decision tree induction.

to improve specific components of decision tree classifiers (Section VI). For instance, we discuss EAs for finding the optimal hyperplane in oblique decision trees [13]–[16], and for improving tree pruning [17], [18] and other components related to decision tree induction [19], [20]. Finally, we review applications of evolutionary algorithms for decision tree induction in different domains, such as software estimation [21], software modules protection [22] and cardiac imaging data [23] (Section VIII). We end this paper by addressing some important issues and open questions for future research (Section IX).

It is important to stress that comprehensive surveys on decision trees have been previously published, such as the papers by Safavian and Landgrebe [24], Murthy [25], and Rokach and Maimon [26]. Also, another paper has been recently published addressing EAs in classification [27]. Nevertheless, to the best of the authors' knowledge, none of them have been fully devoted to address evolutionary induction of decision trees for classification and regression, as well as their respective applications.

II. Taxonomy

We propose a taxonomy of EAs for decision tree induction that is divided into two main threads: evolutionary induction of decision trees and evolutionary design of decision tree components. Regarding the former, each individual of the evolutionary algorithm is a decision tree, whereas in the latter, individuals are components of decision tree classifiers. This taxonomy is presented in Figure 2.

Evolutionary induced decision trees for classification can be either axis-parallel, when there is a single attribute that splits the training data per node, or oblique, when there is a (non-) linear combination of attributes per split. Decision trees for regression can be divided into regression trees, when each leaf-node assigns a value to a test instance, and model trees, when the leaf-nodes contain (non-) linear regression models that are used to predict a value for a new test instance.

Evolutionary design of components can be divided into:

- Hyperplane evolution, where, at each tree node, an EA evolves a near-optimal (non-) linear combination of attributes for oblique trees;
- Pruning method evolution, where an EA is used to handle pruning over an induced decision tree;
- Evolution of other methods, such as parameters of the impurity measure used to split nodes.

Throughout this paper, we will use the following notation. A data set consists of a set of m instances. Each instance $\mathbf{x^j}$ is a n-dimensional attribute vector $\mathbf{x^j} = [x_1^j, x_2^j, ..., x_n^j]$, $(j = 1, 2, ..., m)$, $(\mathbf{x^j} \in \Re^n)$.

III. Evolutionary Algorithms Background

In this section we present some punctual remarks over EAs which are important for the further discussion of using EAs for decision tree induction.

A. Solution Encoding Issues

The type of solution encoding in an EA usually defines the type of EA used. For instance, if solutions are encoded in a fixed-length linear string a Genetic Algorithm (GA) is normally used. Conversely, tree-encoding schemes usually imply Genetic Programming (GP). Although solution encoding can differentiate between GAs and GP, the main question is not what the representation is (e.g. a linear string or a tree) but rather how the representation is interpreted [28].

In this sense, Woodward [29] recommends defining GAs and GP according to the genotype-phenotype mapping: if there is a one-to-one mapping, the EA in question is a GA; if there is a many-to-one mapping, the EA is a GP. Nevertheless, this definition is tricky. For instance, assume a feature selection problem in data mining, where an individual (chromosome) consists of n genes, one for each attribute. Now assume that each gene contains a real value in the range $[0, 1]$, representing the probability of the corresponding attribute being selected. Assume also that, for decoding a chromosome, a threshold is predefined, and an attribute is selected only if the value of its gene is larger than that threshold. In this case, we have a many-to-one mapping, because there are many different genotypes (different arrays of probabilities) that may be decoded into the same phenotype (the same set of selected features). This particular many-to-one mapping does not indicate we are dealing with GP. Actually, we can use the same set of genetic

operators and remaining parameters of a typical GA for this scenario.

We believe that a good distinction between GA and GP is whether a solution encodes data only (GA) or data and functions (GP). Notwithstanding this point, in this survey we review both GAs and GPs indistinguishably.

B. Selection Methods and Genetic operators

Selection is a procedure that chooses which individuals will undergo crossover and mutation. It is usually performed with a bias towards higher fitness in the belief that good solutions have higher potential of generating better individuals for the next generation. Some well-known selection methods are: tournament selection, roulette wheel selection and rank-based selection.

Tournament selection works as follows: a predefined number of individuals (known as tournament size) is drawn from the population, and the fittest of them is chosen to be a part of the reproducers pool. This procedure is repeated until the pool of reproducers is full.

The roulette wheel selection, also known as stochastic sampling with replacement, is analogous to the use of a casino's roulette wheel, with each slice of the wheel proportional in size to the fitness of an individual. As a result, the probability of an individual being chosen is proportional to its fitness.

Rank-based selection, unlike fitness-proportional selection (e.g., roulette wheel), rank individuals according to their fitness (absolute fitness values are discarded). The individuals are then selected based on the value of their rank positions. This method overcomes the scaling problems of fitness-proportional assignment, e.g. premature convergence when few individuals with very high fitness values dominate the rest of the population.

Crossover is the operator responsible for exchanging genetic material - usually between two individuals - for the creation of mixed individuals for the next population. Regarding the fixed-length binary string encoding, a common approach for crossover is the well-known 1-point crossover. Each parent binary string selected to reproduce is split into two (in a predefined position, the "1-point") and the parents generate two new offspring by concatenating the substrings before and after this position from different parents.

Finally, mutation is the operator responsible for modifying the genetic structure of a given individual to allow any solution to be reached. In doing so, it reduces the chances of premature convergence to local optima. Usually, the mutation operator operates over a single randomly selected individual, changing its genotype accordingly (e.g., flipping a bit in a binary string or growing a new branch in a tree-based genotype).

C. Multi-Objective Optimization

A crucial issue in data mining is how to evaluate the quality of a candidate model. EAs naturally allow the evaluation of a candidate solution as a whole, in a global fashion, through the fitness function. This is in contrast with data mining paradigms which evaluate a partial solution [11]. For instance, a conventional greedy decision tree induction algorithm incrementally builds a decision tree by partitioning one node at a time. When the algorithm is evaluating several candidate divisions, the tree is still incomplete, being just a partial solution, so that the decision tree evaluation function is somewhat shortsighted.

In decision tree induction, it is often desirable to maximize both the predictive accuracy and the comprehensibility of the induced tree [30]. Once again, EAs seem to be a natural choice for this task, since they naturally allow the evaluation of a candidate solution by simultaneously considering different quality criteria. This is not so easily performed in other data mining paradigms [11]. Three general approaches are used multi-objective optmization: weighted-formula, Pareto dominance, and lexicographic analysis.

In the weighted-formula approach, a weight (typically a user-defined parameter) is assigned to each objective (measure) to be optimized, according to its importance within the application domain. Next, these weighted objectives are summed or multiplied accordingly, reducing multiple objectives into a single objective.

The concept of Pareto dominance can be formally defined as: A solution $\mathbf{A} = \{a_1, a_2, ..., a_o\}$ is said to *dominate* solution $\mathbf{B}$ (for a set of objectives o), symbolically expressed by $\mathbf{A} \prec \mathbf{B}$, when the following conditions hold:

$$(\mathbf{A} \prec \mathbf{B}) \Leftrightarrow (\forall i)(a_i \leq b_i) \wedge (\exists i)(a_i < b_i) \quad (1)$$

Thus, the Pareto optimal set is said to be the set of solutions that are not dominated by any other solution, i.e.

$$\{\mathbf{A} = (a_1, a_2, ..., a_o) | \neg(\exists \mathbf{B} = (b_1, b_2, ..., b_o), \mathbf{B} \prec \mathbf{A})\} \quad (2)$$

Unlike the weighted-formula approach, the Pareto dominance provides a set of non-dominated solutions instead of a single "best" solution.

The lexicographic approach determines priorities among the objectives, and the best solution is the one that is significantly better according to a higher-priority objective. If no such best solution is found, the next objective is chosen following a priority order. To better understand this approach, consider the following example. Let x and y be two decision trees and a and b two evaluation measures. Besides, consider that a has the highest priority between the measures and that t_a and t_b are tolerance thresholds associated with a and b respectively. The lexicographic approach works according to the following analysis: if $|a_x - a_y| > t_a$ then it is possible to establish which decision tree is "better" considering objective a alone. Otherwise, the lower-priority measure b must be evaluated. In this case if $|b_x - b_y| > t_b$ then the fittest tree between x and y can be decided by considering measure b alone. If the difference between values falls within the assigned threshold t_b, the best value of the highest-priority measure a is used to determine the fittest tree.

For a critical review of the pros and cons of each multi-objective strategy discussed in this section, see [30]. For further information on multi-objective optimization, we recommend reading [31], [32].

IV. Classification

This section reviews EAs that evolve decision trees for classification tasks, where each individual is a classification

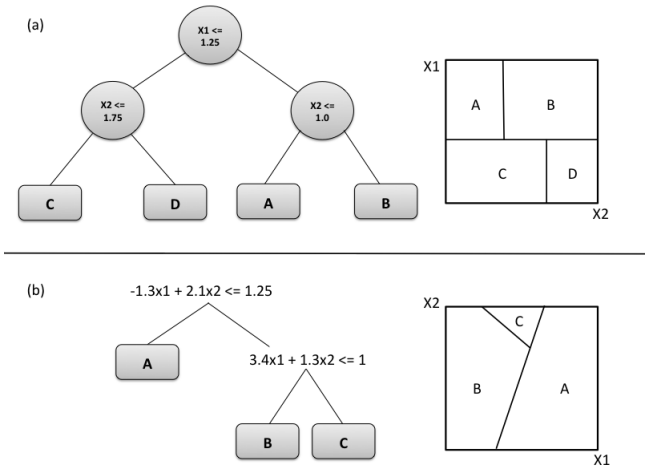

Fig. 3. Axis-parallel (a) and oblique (b) decision trees, and their respective two-dimensional partitioning. (Adapted from [33].
)

tree. We divide this section into two parts: EAs that evolve axis-parallel decision trees and EAs that evolve oblique decision trees. The difference between these approaches is that whereas axis-parallel trees make use of a single attribute to split each node (e.g., $w_i x_i < 0$), oblique trees use a linear (or some times a non-linear) combination of attributes (e.g., $\sum_{i=1}^{n} w_i x_i < 0$). Figure 3 presents both axis-parallel (a) and oblique (b) decision trees.

A. Axis-Parallel Decision Trees

Axis-parallel decision trees are the most common type found in the literature, mainly because this type of tree is usually much easier to interpret than an oblique tree. We divide our analysis on axis-parallel decision trees according to the main steps of the evolutionary process. That is, we analyze how solutions are encoded; which methods are used for initializing the population of decision trees; the most common strategies for fitness evaluation; the genetic operators that are designed to evolve individuals; and other related issues.

1) Solution Encoding: In Section III-A, we have explained some terminology issues which are usually dictated according to the EA solution encoding scheme. Nomenclature aside, decision tree encoding is usually either tree-based or non-tree based. We comment on both next.

Tree-based encoding is the most common approach for coding individuals in EAs for decision tree induction, and it seems a natural choice when we are dealing with decision trees. In [34], the author applies competitive co-evolution for decision tree induction and uses a tree-encoding scheme. The system designs binary decision trees where each node is represented by a 4-tuple (Figure 4). Each component in Figure 4 is a numeric (integer or real) value that can be modified during the evolutionary process. The first element of the 4-tuple is an integer that indexes each data set attribute; the second one is an integer that indicates whether the node is non-terminal or terminal; the third one is an integer that indexes

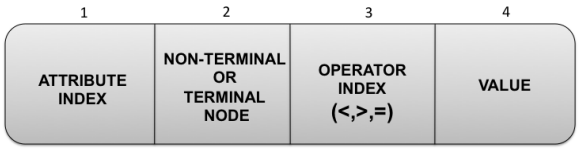

Fig. 4. Node representation in tree-encoding [34]. A tuple of 4 elements that defines (1) the data set attribute to be tested or predicted; (2) the type of node (non-terminal or terminal); (3) the operator to be used; and (4) the value to be tested by the attribute in (1) according to the operator in (3) or, alternately, the binary classification value.

which operator is to be used ($<, >, =$); and the fourth one is a real number that indicates the value to be tested (in a non-terminal node) or the binary classification (in a terminal node). It is not clear how the linkage of nodes is handled, but we assume each node described in Figure 4 has two pointers for the children nodes (which assume *null* values for terminal nodes).

A similar approach is presented in [35]–[37], where the authors use a tree-encoding solution in which each node is a 7-tuple: $node = \{t, label, P, L, R, C, size\}$, where t is the node number (t = 0 is the root node), $label$ is the class label of a terminal node (meaningful only for terminal nodes), P is a pointer for the parent node, L and R are pointers to the left and right children, respectively ($null$ for terminal nodes), and C is a set of registers, where $C[0]$ stores the attribute id and $C[1]$ the threshold value for the test $feature_{C[0]} < C[1]$, whose possible outcomes are "yes" (path to the left-child node) or "no" (path to the right-child node).

Zhao's genetic programming system [38] also encodes individuals as trees. Zhao defines terminal and function nodes. Terminals can be integer (attribute ids), real (attribute values) or binary (a class label or a function node) values. Each function node takes four arguments and returns a binary result. Its signature is $N : integer \times real \times binary \times binary \rightarrow binary$. It can be represented as a four-tuple, $N = \{a, v, L, R\}$, where a is an attribute terminal, v is a value terminal, and L and R are class terminals or node functions. Note that since both the type of a class terminal and the output type of a node function are binary, L and R can be either class terminals or function nodes.

Papagelis and Kalles [39] defend the use of the tree-encoding scheme in EAs for decision tree induction. They state that whereas GAs use binary strings to represent points in search space, such representations do not appear well suited for representing the space of variable-size decision trees. Furthermore, they affirm that it is natural to use a tree structure to represent decision trees and that the mutation-crossover operators can be efficiently altered to match this structure. For other tree-encoding scheme examples, see [22], [28], [40]–[47].

Even though the tree-encoding scheme is the most used approach for decision tree evolution, several works use different approaches for coding individuals. Most of these approaches adopt a fixed-length string representation, an inheritance from the misconception that GAs are defined by their representations. Fixed-length string representations, also called "linear chromosomes", are typically tricky to implement for non-

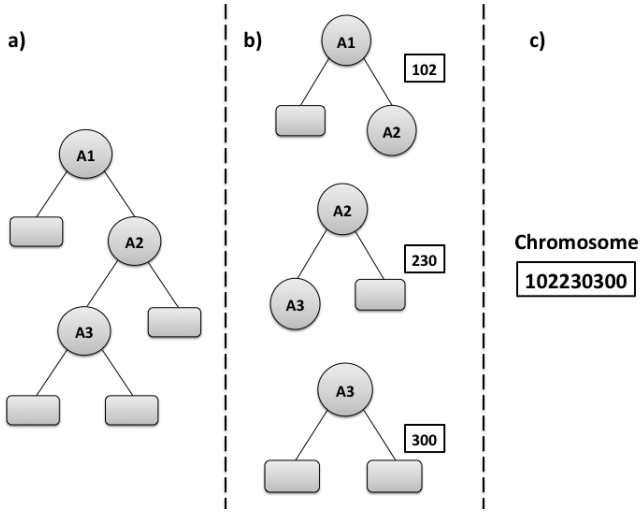

Fig. 5. Mapping decision trees in linear chromosomes [48]. a) decision tree. b) subtrees mapped into caltrops. c) chromosome resulting from the union of caltrops.

binary decision trees. Thus, most works propose EAs that evolve binary decision trees, arguing, for instance, that any non-binary decision tree can be converted into a binary tree.

In [48], linear chromosomes are composed by genes, named caltrops, an allusion to the spiked devices used in medieval warfare to disable charging horses. Caltrops are subtrees formed by a root node and two child nodes. Each caltrop is represented by three integers, where non-terminal nodes are identified by an attribute index and terminal nodes by the value 0 (zero). Figure 5 shows how a decision tree (a) is divided into subtrees, which in turn are mapped into triples of integers named caltrops (b), genes that will form the individual's chromosome (c). Notice that caltrops do not encode tests over the attributes. This representation assumes that each data set attribute is boolean, and assigns the left (right) child node when the attribute value is true (false).

Bandar et al. [49] provide a GA for induction of multiple decision trees. Linear chromosomes are formed by integers that correspond to attribute indexes. The total number of genes in a chromosome is $2^{depth} - 1$, where $depth$ is the maximum number of attribute split levels, a variable parameter. To create the decision tree, one must select the attribute corresponding to the first gene of the chromosome. This attribute will be used to split the root node in left and right children. Afterwards, a binary branching is performed using the training set instances contemplated by the node (for the root node, all instances are used). The measure used to perform the binary branching is not mentioned in their work. The procedure is recursive, and the binary branching is performed until the tree has reached its maximum depth.

Smith [50] also designs binary decision trees coded through linear chromosomes. Even though his strategy for evolving decision trees is domain-specific (RNA search acceleration), it could easily be extended to a generic framework. Each gene of the chromosome consists of two integers: a node type n (in reality the correspondent to an attribute index), which varies in the interval $[1, 5]$ (domain constraint), and a type/index v, which is the threshold value for which the test $n \geq v$ is performed. Once again, the instances resulting from the test are filtered to the left (right) node if the result is false (true).

In spite of the fact that decision trees encoded as linear chromosomes seem to be easier to handle than those encoded by tree-encoding schemes, fixed-length linear chromosomes have some disadvantages, such as: (i) the need of constant mapping between genotype and phenotype for fitness evaluation; (ii) difficulty in handling non-binary decision trees and (iii) difficulty in defining a maximum number of genes for fixed-length structures. For instance, chromosomes with a large number of genes may have several genes with *null* values, which can in turn harm operations like one-point crossover. Conversely, structures with a low number of genes can ultimately restrain the size of the trees to be discovered in the evolutionary process. Dynamic-length string structures, on the other hand, may add an unnecessary complexity to the EA design. Genetic operations, like crossover, may have to be modified when chromosomes with different sizes are to be reproduced.

2) Population Initialization: An EA's initial population has to provide enough diversity of individuals so that the genetic operators can search for solutions in a more comprehensive search-space, avoiding local optima. Nonetheless, a large search-space may result in very slow convergence, preventing the EA from finding a near-optimal solution. In this case, task-dependent knowledge constraints may speed-up convergence by avoiding the search in "dead zones" of the solution space. It is clear that there is a thin line between the precise amount of diversification for avoiding local optima and task-dependent knowledge constraints for speeding-up convergence.

Most works on evolutionary induction of decision trees propose a partially random initialization of trees. We use the term "partially" because the randomness of individuals is constrained to the data set attributes and their possible values within the training set. Some approaches propose additional constraints during initialization in order to to guarantee the logic validity of the created decision trees. For instance, DeLisle and Dixon [51] state that their method initializes decision trees "at random based upon the training set and given mild constraints on the minimum number of observations allowed in terminal or leaf nodes".

For the cases where each decision tree is encoded as a fixed-length string, a random initialization is often used ([48], [52], [53]). When the decision tree is encoded as a tree, a common strategy for initialization is randomly choosing attributes and split values from a predefined list and halting the decision tree growth when the tree reaches a depth that is randomly selected from an interval, typically $[2, maxDepth]$. This strategy usually creates fully balanced trees (the distance from the root to any leaf node is the same). This generation procedure is called the full method [35]–[38], [54]. The full method has the disadvantage of not providing enough diversification of tree shapes, which may demand an increased number of generations for convergence to a good solution.

Trees can be generated with varying distances from root to the leaves, namely the grow method [55]. Finally, the population can be a mixture of trees generated by either the

full or the grow method, a procedure named ramped half and half [56]–[60]. Random initialization of decision trees encoded as trees is further discussed in [40], [61]–[64].

Another approach for initializing decision trees in an EA is to restrain the initial population to 2-level decision trees, i.e., a root node and its respective leaves [34], [39], [65], [66]. EAs that implement this strategy rely on the further application of genetic operators to obtain deeper trees.

One could say the only task-dependent knowledge that the previously commented EAs employ is related to the choice of possible split values. Although most of them claim to randomly choose split values, most of the times these values are not really random, i.e. they are a set of observed values from the training set. This is the most rudimentary use of task-dependent knowledge in the initialization process. For instance, Ma and Wang [67] reduce the impact of randomness by backtracking a "bad" random choice and replacing it by a "better" random choice. The definition of good and bad choices, in this case, is given by the amount of training data that are filtered to each child node of the currently generated node. If no random choice is considered to be good enough, a leaf node stops the growth of that particular path.

A more robust approach for generating task-dependent knowledge-based initial trees is executing a greedy traditional algorithm, e.g. C4.5 or CART, in samples from the training set and incorporating the resulting decision trees in the initial population. This strategy is implemented in [41], [44], [68], where C4.5 is used for generating the initial population of trees. Kretowski and Grzes [69]–[71] also implemented this approach but instead of using a well-established split measure such as the gain ratio or the gini index, they opted for a dipolar split measure.

Basgalupp et al. [28], [45] also proposed a task-dependent knowledge based initialization for their EA for decision tree induction. It generates ten 2-level decision trees for each data set attribute, where each one of these ten trees per attribute will possibly have different split values, since they are calculated using the information gain measure upon a different subsample of the training set. After the generation of all 2-level decision trees, they are combined according to the growing method previously described.

3) Fitness Evaluation Methods: Evolutionary decision tree induction algorithms can be roughly divided into two threads regarding fitness evaluation: single-objective optimization and multi-objective optimization.

EAs that perform single-objective optimization use a single measure to guide the search for near-optimal solutions. The most common measure for evaluating individuals in evolutionary algorithms for decision tree induction is classification accuracy (or its complement, classification error):

$$acc = \frac{c}{m} \tag{3}$$

where c is the number of correctly classified instances and m is the total number of instances. An accuracy-based fitness function is used in [41]–[43], [49], [61], [67], [72].

In [55], [73], the authors propose using acc^2 in the fitness function, because it provides "a non- linear bias toward correctly classifying instances in decision tree T while providing differential reward for imperfect decision trees". Folino et al. [62], [63] propose the use of the J-Measure, which is used to measure the quality of the disjunction of rules (paths from the root to the leaf) that describe each class. More specifically, for a k-class problem, there are k rules of the kind (**if** Y_i **then** ω_i), where Y_i is a set of disjunctions among paths that are used to label instances as belonging to class ω_i. The J-Measure can be thus defined as

$$J = \sum_{i=1}^{k} p(Y_i)p(\omega_i|Y_i)\log\left(\frac{p(\omega_i|Y_i)}{p(\omega_i)}\right) \tag{4}$$

where $p(Y_i)$ is the fraction of instances satisfying condition Y_i, $p(\omega_i)$ is the fraction of instances belonging to class ω_i, $p(\omega_i|Y_i)$ is the fraction of instances that both satisfy Y_i and belong to class ω_i divided by the fraction of instances satisfying Y_i, and k is the total number of classes. The higher the J-Measure value, the higher the tree's predictive accuracy.

Aitkenhead [34] proposes a distance score for evaluating individuals which is a measure of how close a decision tree came to the correct classification. We assume that this score can only be applied to problems where the class attribute is ordinal and is converted to sequential integers that preserve the order among values. The distance score is given by

$$D_{score} = \frac{1}{m}\sum_{i=1}^{m} 1 - \left(\frac{y^i - y'^i}{y_{max} - y_{min}}\right)^2 \tag{5}$$

where m is the number of instances, y^i is the actual class value for the i^{th} instance, y'^i is the predicted value for the same instance and y_{max} (y_{min}) is the maximum (minimum) value of the class attribute (thus the necessity of converting categories into integers).

Kennedy et al. [48] use the number of decisions necessary to classify all the members of the instance set as a fitness measure to be minimized. Finally, Fu et al. [44] propose a bilinear loss function that, according to a given parameter, selects one of the possible percentiles of the classification accuracy distribution to estimate tree accuracy.

Works using a single-objective fitness function usually do not defend this choice for decision tree induction against a multi-objective strategy. A small number of works, however, argue that a multi-objective approach that seeks a compromise between predictive accuracy and solution complexity (tree size) as a form of parsimony pressure is not as beneficial as it may sound. Ma and Wang [67], for instance, argue that this compromise reduces the search space and leads to a slower overall increase in accuracy. Moreover, they reckon that the search may get stuck in regions containing less accurate trees of the same size as those produced without using the complexity penalty in the multi-objective fitness function, and that, as a result, the parsimony pressure would be a disadvantage instead of an advantage.

In works where multi-objective optimization is performed, it is argued that the balance between accuracy and parsimony is very important for efficient evolutionary search. The argument is centered on the fact that the use of accuracy alone may result in an arbitrarily complex classifier that fits the noise within the data set. Such a classifier would have high accuracy on the

training set but would likely perform poorly on previously unseen data. Based on this assumption, most EAs try to optimize both predictive accuracy and simplicity (which is usually assumed to be inversely proportional to the number of nodes in a decision tree). Other works seek a compromise between distinct measures, such as sensitivity and specificity in cost-sensitive approaches, which also demands the application of a multi-objective optimization strategy.

Three general approaches are used for coping with multi-objective optimization in EAs for decision tree induction: (a) weighted-formula; (b) Pareto dominance; and (c) lexicographic analysis.

Works on (a) are by far the most common. A typical strategy is to combine accuracy ($acc(I)$) and tree size ($size(I)$), as follows:

$$f(I) = \alpha \times acc(I) - \beta \times size(I) \tag{6}$$

where α and β are weights and the formula has to be maximized. This approach or minor variations of it are found in [39], [60], [65], [66], [74].

Tsakonas and Dounias [75] also propose a weighted-formula that combines accuracy and a simplicity component, though the idea is to penalize for smaller-sized trees due to domain constraints. Kretowski and Grzes [54] propose a small variation of (6) where the complexity component takes into account the number of leaves and the number of features associated to each test in non-terminal nodes (for the case of oblique decision trees). In [70], [71], the authors propose a cost-sensitive approach that replaces accuracy in (6) by the misclassification cost, which is summed to the tree size in an equation whose value should be minimized. Other similar cost-sensitive approaches, which usually consider measures like sensitivity and specificity, are reported in [69], [76]–[81].

Haizhou and Chong [68] introduce a weighted-formula that combines classification error, tree depth and number of attributes used in each path of the tree. Similarly, Reynolds and Al-Shehri [52] propose a fitness function based on accuracy, number of nodes, number of attributes used and homogeneity of each partition. Nikolaev and Slavov [82] offer a variation of Quinlan's pruning strategy presented in [83] as a stochastic fitness function, and perform a detailed analysis of the fitness landscape structure.

DeLisle et al. [51] propose the evaluation of accuracy and complexity of decision trees by using minimum description length (MDL) as fitness function, defined as follows:

$$MDL = error_{CL} + tree_{CL} \tag{7}$$

$$error_{CL} = \sum_{l \in leaves} \log_2 \binom{m_l}{e_l} \tag{8}$$

$$tree_{CL} = (ni + nt) + ni \log_2 s + nt \log_2 k \tag{9}$$

where m_l is the number of instances in leaf node l, e_l is the number of misclassified instances in leaf node l, ni is the total number of internal nodes, nt is the total number of terminal nodes (leaves), s is the total number of splits and k the number of classes.

The error coding length ($error_{CL}$) is based upon the binomial distribution and represents the number of possible combinations given the total number of observations (m) and the number of incorrectly predicted observations (e). This relates to the likelihood of a particular (m, e) combination arising by random chance, and this value should be minimized. The tree coding length ($tree_{CL}$) is dependent upon the overall size of the decision tree and should also be minimized. The authors of this particular application state that only the number of nodes and the number of leaves were actually used in the tree coding length component, since the removal of the remaining terms resulted in no alterations in performance apart from a reduced computational cost.

For the evaluation of each rule extracted from the evolved decision tree, Garcia-Almanza and Tsang [59] propose a fitness function based on recall and a slightly variation of precision that severely penalizes the false positive cases. Since no complexity penalty is proposed, the authors suggest a pruning method to simplify the evolved decision trees.

A smaller number of works investigate the use of the two other multi-objective strategies (namely Pareto dominance and lexicographic analysis). In one of these works, Kim [64] and Zhao [38] implement a Pareto dominance approach that, instead of providing a single optimal solution based on the weighted combination of objectives, provides an optimal set of *non-dominated* solutions.

The Pareto multi-objective strategy proposed by Kim [64] tries to minimize two objectives: classification error rate and tree size (measured by the number of decision rules). For ranking the individuals and discovering the Pareto optimal set, Kim implements a dominating rank method [84], where the rank of a given solution in a Pareto distribution is given by the number of elements dominating that solution. Hence, the highest possible rank is zero, for an element that has no dominator. The Pareto optimal set is given by all elements whose rank is zero. Each non-dominated solution in a discrete space of tree size represents the minimized error fitness for each number of decision rules.

Zhao [38] also proposes a Pareto dominance approach for minimizing two conflicting objectives (e.g., false negative rate vs. false positive rate). This approach allows the decision maker to specify partial preferences on the two conflicting objectives in order to further reduce the number of alternative solutions. This trade-off can be similarly adopted on other pairs of performance measures, such as sensitivity vs. specificity or recall vs. precision, which have been typically employed in domains such as medical diagnosis and information retrieval.

The lexicographic approach for multi-objective optimization has also been employed for evolutionary induction of decision trees. Eggermont et al. [56], [57] propose a lexicographic fitness[1] whose highest-priority measure is misclassification error, followed by tree size. In that work, there are no tolerance thresholds, i.e., only in cases where the misclassification error of two individuals is exactly the same the tree size measure will be used to choose the best individual. Zhao et al. [35]–[37] also propose a lexicographic fitness[2] that evaluates accuracy

[1]In both references ([56], [57]), the authors refer to the lexicographic analysis as "multi-layer fitness function".

[2]No mention of the term "lexicographic analysis" is made throughout the three references ([35]–[37]).

(highest-priority measure) and tree size (total number of tree nodes). Again, there are no tolerance thresholds.

Basgalupp et al. [28], [45] propose LEGAL-Tree (Lexicographic Genetic Algorithm for decision Tree induction), a GA whose fitness function implements the lexicographic approach. Once again, the objectives that will guide the search for the best individual are accuracy and tree size. The authors use both validation-set and training-set accuracy (highest and second highest priority measures, respectively), aiming to avoiding both overfitting (validation-set) and underfitting (training-set). The last objective in the priority rank is tree size, measured by the total number of tree nodes.

4) Selection Methods and Genetic Operators: Selection is the procedure that chooses which individuals will undergo crossover and mutation. In evolutionary induction of decision trees, the most frequently used approach for selection is tournament selection.This selection method is used in [28], [38], [45], [48], [51], [52], [56]–[59], [64], [66], [67], [75], [76], [85].

Another popular choice in EAs for decision tree induction is the roulette wheel selection.Works that implement this strategy for evolving decision trees are [35]–[37], [42]–[44], [48], [51], [55], [60]–[63], [73], [82], [86], [87].

A less-common selection method in EAs for decision tree induction is rank-based selection. This selection method in decision tree induction is used in [70], [71], [74], [77], [78], [80], [81].

Two operators normally used to evolve a population of individuals are crossover and mutation.

In EAs for decision tree induction, crossover is usually performed in two different ways (with small variations) according to the individual representation. For fixed-length binary string encoding, it is a common approach to perform the well-known 1-point crossover. It is used in [48]–[50], [53], [68].

Nonetheless, the vast majority of EAs encode decision trees in a tree representation, and as a result implement the standard GP crossover. This crossover selects nodes in two individuals and exchanges the entire subtrees corresponding to each selected node, generating two offspring. This operator is used in [28], [35]–[45], [51], [55]–[58], [61]–[67], [69], [73], [77]–[82], [85]–[88]. Two small variations of this strategy are found in the literature. In [74] the authors add the constraint that selected nodes from the two parents must represent a test over the same data set attribute in order to be exchanged. Kretowski and Grzes [54], [70], [71] introduce a "test-only exchange crossover", in which instead of replacing the entire subtrees, only the test represented by an attribute-value pair is replaced. This type of crossover demands the number of outcomes of the selected nodes to be the same in order to preserve the original tree structure.

EAs for induction of decision trees usually implement more than one mutation strategy. Recalling that most such EAs deal with tree-based encoding, two strategies are most used: (i) replacing a subtree by a randomly generated one; and (ii) replacing information regarding the test corresponding to the selected node. In (i), a randomly selected subtree is replaced by a randomly generated one. Usually no further details are given on how the new subtree is randomly generated. This approach is used in [38], [51], [56]–[58], [61]–[64], [67], [79], [81], [82], [85], [86]. In a more restricted version of this approach a subtree is replaced by a random leaf node or a leaf node is replaced by a random subtree [28], [45], [54], [64], [69]–[71], [77], [78], [80], [82]. In (ii), instead of replacing subtrees (a structural mutation), a test-based modification (a semantical mutation) is performed. EAs implementing this strategy usually allow the replacement of either the attribute, the corresponding test value or both [35]–[37], [51], [54], [55], [64], [66], [67], [69]–[71], [73], [77], [78], [80], [81], [87], [88]. In [39], [65], [74], this strategy is restricted by allowing mutation of the test-value only.

A few alternative mutation strategies are as follows. Fu et al. [41]–[44] propose a "self-contained" mutation strategy, in which a randomly selected (non-)terminal node is replaced by another (non-)terminal node already present in the tree, so there is no need for randomly generating a new subtree during mutation, saving processing time. Sorensen and Janssens [40] propose two types of mutation: switch and translocation. The first switches children from the same parent and the second exchanges children from different parents in the same tree level. Both can be seen as special cases of self-contained mutation, since nodes are replaced by nodes already present in a tree. Rouwhorst and Engelbrecht [66] propose a relational mutation, which modifies the test operator corresponding to the randomly selected node. This strategy allows multiple types of operators, not only the traditional $\leq$ and $>$. For instance, suppose a node contains the test $x > 5$. A relational mutation over this node could replace the operator $>$ by $\neq$, resulting in $x \neq 5$.

In fixed-length strings [48], [53], mutation is performed by simply altering a randomly chosen value in the string, which may change the attribute being used, its test value or both, depending on the approach. Hence, mutation in fixed-length strings is mainly semantical, i.e., it does not affect the decision tree structure. Nevertheless, in cases where the node type (non-terminal or terminal) is also encoded in the gene/chromosome, it is possible that the structure of the decision tree is modified by mutation.

5) Parameter Setting: The parameter values of an EA can largely influence whether the algorithm will find a near-optimum solution, and whether it will find such a solution efficiently. Choosing correctly the parameters, however, is a time-consuming task and considerable effort has been dedicated to the development of good heuristics able to overcome this problem [89]. Espejo et al. [27] state that the large number of parameters that must be defined in order to have a working system is in fact one of the pitfalls of GP-based classifiers (an argument easily generalizable for other EAs).

The most common parameters in EAs for decision tree induction are population size, number of generations, probabilities of application of different genetic operators and maximum size of decision trees at initialization or during the evolutionary process. In practice, several preliminary runs are usually required in order to tune these parameters. In some works, parameters are fine-tuned to suit each data set employed in the experiments [66], [67]. However, most authors prefer to present a set of default parameter values followed by a

sentence like "parameter values were empirically defined". We believe this is due to the fact that no work has exhaustively investigated the influence of different parameter settings in EAs for decision tree induction, like De Jong and Spears have for function optimization [90]. A possible explanation is that decision tree induction is data dependent, which means different classification data sets may present very distinct error surfaces, favoring particular sets of parameter values.

The population size parameter influences the number of candidate solutions evaluated by the EA. Some values for this parameter used in the literature are 50 individuals in [49], [82]. 100 individuals in [41], [56]–[58], [66], 200 individuals in [35]–[37], [39], [65], 400 individuals in [61]–[63], 500 individuals in [28], [45], [64], [85] and 1000 individuals in [51], [59].

Another parameter, the number of generations, is usually the EA's stopping criterion. A popular choice in EAs for inducing decision trees is evolving individuals for 1000 generations [51], [64], [70], [71], [78]. However, in some EAs the number of generations is as small as 10 [42], 30 [76] or 50 [41], [43], [44], [49], whilst in other EAs it is as large as 5000 [36], [37] or 10000 [38], [54]. A useful strategy is to set a secondary stopping criterion to identifying cases of fast convergence. For instance, if the EA does not improve the best individual during a pre-determined number of generations, it can be stopped, being assumed that it has already converged to the (near) optimal solution. Once again, there is no consensus on the most suitable value for this parameter. Values like 3% [28], [45], 10% [54], 20% [70], [71] and 25% [60], [65] of the number of generations have been used.

Crossover is the main genetic operator for evolving individuals. Most EAs use a high probability of crossover, such as 60% [49], [64], [75], 80% [59], [60], [66], [82], [85] or 90% [28], [39], [45], [56]–[58], [61], [65], [76]. Mutation, on the other hand, usually occurs much less often. The most common value for mutation rate is 1% [35]–[37], [41], [43], [43], [44], [60], [61], [76]. Other popular values range from 2 to 10% [28], [38], [39], [45], [49], [59], [64], [65], [78], [85].

In the popular tournament selection method, the size of the tournament is yet another parameter that needs to be set. Blickle and Thiele [91] claim that typical values for tournament size are 2 and 3, and that there is a great loss of diversity when the value is higher than 5. However, there is no clear consensus on tournament size in EAs for decision tree induction research. Some works seem to follow the suggestion of Blickle and Thiele by setting a tournament size of up to 5 individuals [56]–[59], [64], [76]. Others choose arbitrary values such as 6 [75], 7 [38], 8 [67], 10 [66], 30 [28], [45], 80 [36], [37] or 100 [35]. None of these works make a detailed analysis on the effects of different tournament sizes.

Most EAs for decision tree induction are generational, i.e., individuals of the current population are replaced by their offspring at each generation (iteration). In order to avoid losing good solutions in this process, it is a common practice to keep the best individual(s) from the current generation to the next, which is named elitism. The number of individuals to be kept is yet another parameter that needs to be set. A popular choice is a small number of individuals to be part of the elite (1 to 5 individuals) [59], [67], [78].

Finally, most EAs use a maximum tree size at initialization or during the evolutionary process. This maximum size is usually defined in terms of tree depth. Trees are usually size-limited during initialization in order to speed-up the algorithms' running time. Typical initialization size limits are 3 levels [28], [45], 4 levels [61]–[63], [76], 7 levels [60] and 10 levels [38]. Some EAs also limit the tree depth during evolution; for instance, with a limit of 17 levels in [76] or 30 levels in [38] in 30 levels.

Note that parameter setting in EAs is itself a research field and it goes way beyond trial and error analysis. However, none of the EAs for decision tree induction reviewed so far make a detailed analysis on parameter values. Ma and Wang [67] suggest a heuristic method to set some EA parameters. The idea is to use a neural network a priori of the EA execution. For instance, the maximum number of generations can be defined as the maximum number of iterations a feed forward neural network needs in order to find an acceptable result. They also state that setting parameters in an EA is no more difficult than choosing the numbers of layers and hidden neurons in feed forward neural networks.

B. Oblique Decision Trees

Oblique decision trees, also referred to as (non-) linear decision trees, are a common alternative to the traditional axis-parallel approach. Oblique decision trees are usually much smaller and often more accurate than axis-parallel decision trees, though at the expense of more computational effort and loss of comprehensibility. In oblique decision trees, an hyperplane that divides the feature space into two separate regions can be formally defined as

$$H(\mathbf{w}, \theta) = \{\mathbf{x} \colon \mathbf{w^T x} = \theta\} \tag{10}$$

where $\mathbf{w} = [w_1, w_2, ..., w_n]^T$, $\mathbf{w} \in \Re^n$ is a weight vector, θ is a threshold and $\mathbf{w} \cdot \mathbf{x}$ is the inner product between the weight vector and the data set instances. If $\mathbf{w^T x^i} - \theta > \mathbf{0}$, it means that the instance $\mathbf{x^i}$ is in the positive side of hyperplane $H(\mathbf{w}, \theta)$. Conversely, $\mathbf{w^T x^i} - \theta \leq \mathbf{0}$ means that the instance is in the negative side of the hyperplane.

Several approaches based on EAs have been proposed to evolve oblique decision trees. Bot and Langdon [33], [92] propose a GP for evolving oblique decision trees. In this work, each GP individual is encoded as a tree with function nodes and terminals. A function node has as its children a tuple $(\{w_i, x_i\}, threshold, ifTrue, ifFalse)$, where w_i and x_i are the i^{th} weight constant and attribute pair, respectively. Depending on the function node type, there can be n pairs $\{w_i, x_i\}$. Still in this tuple, $threshold$ is a constant terminal and $ifTrue$ ($ifFalse$) can be either classification terminals or other function nodes. Terminals are either constants (doubles), variables (integers) or classifications (integers). Nodes are evaluated as follows:

if $\sum_{i=1}^{n} w_i x_i \leq threshold$ **then**
  return value of ifTrue branch
else

return value of ifFalse branch
end if

This GP is strongly typed [93], i.e., the data types of functions and terminals are specified to ensure that only valid individuals are generated. A terminal of *variable* type is an integer ranging from 0 to the number of data set attributes -1 (an attribute index). A terminal of *constant* type is a floating-point within a predefined range. A terminal of *classification* type is an integer ranging from 0 to the number of classes -1 (class index). The function set allows up to 3 attributes to be tested within each node of the decision tree.

This GP has the following parameters. Individuals are selected by tournament selection with size 7. A population of 250 individuals evolve during 1000 generations. The initial population is created through the ramped half and half method [94], [95]. Crossover and mutation rates were empirically set to 0.5 each. The fitness function is a weighted-formula that includes validation set accuracy and a penalty factor to lower the fitness value, multiplied by either the tree depth or the total number of nodes. The penalty factor is used to alleviate the bloat problem [96].

In [97], Bot expanded his work from [92] by introducing limited error fitness (LEF) [98], Pareto scoring and fitness sharing [99]. LEF is a technique for speeding up fitness calculation in supervised learning problems where each individual is evaluated on a number of training cases. If the error score achieved by the individual in these training cases is above a given threshold, all the remaining training cases are counted as errors. This is to prevent poorly-designed individuals to be tested over the entire training set, wasting computational time. Individuals whose number of errors is below the given threshold are evaluated over the entire training set.

Pareto scoring and fitness sharing are also introduced. Since the use of Pareto scoring leads to populations that tend to converge to a few (and possibly suboptimal) solutions [99], fitness sharing is used to ensure that more different local minima are found. It is a niching technique that avoids premature convergence by penalizing solutions with many "neighbors", i.e. individuals very similar to others. For such, a sharing function $s(i) = \sum_j s(i,j)$ is calculated for each individual i in the Pareto front. The distance (similarity) of individual i to every other individual is computed and, the more similar an individual j is to i, the higher the value of $s(i,j)$ and thus of $s(i)$. Finally, the individual of the Pareto front with the lowest value of $s(.)$ is selected to reproduce.

Another work, by Kretowski and Grzes [100], introduced GEA-ODT (Global Evolutionary Algorithm for Oblique Decision Tree induction). GEA-ODT encodes each decision tree as a tree with splitting hyperplanes in non-terminal nodes and class labels in the leaves. Each hyperplane is represented by a $(n+1)$-dimensional table, accounting for $\mathbf{w}$ and θ. The population of individuals is initialized by generating each individual through a top-down algorithm that searches at each node for the best hyperplane. Hyperplanes are randomly generated based on randomly chosen mixed dipole $(\mathbf{x^i}, \mathbf{x^j})$. A dipole [101] is a pair of instances $(\mathbf{x^i}, \mathbf{x^j})$, and it is referred as a mixed dipole if and only if its instances belong to different classes. Thus, a hyperplane $H(\mathbf{w}, \theta)$ splits the dipole $(\mathbf{x^i}, \mathbf{x^j})$ if and only if:

$$(\mathbf{w^T x^i} - \theta) \times (\mathbf{w^T x^j} - \theta) < 0 \tag{11}$$

which means that $\mathbf{x^i}$ and $\mathbf{x^j}$ are on opposite sides of the dividing hyperplane.

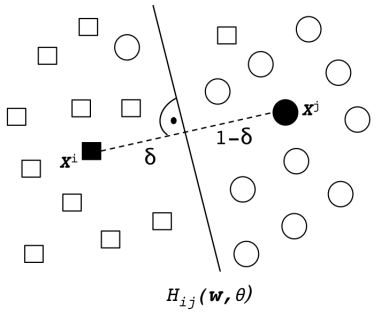

Fig. 6. Initializing a hyperplane with randomly selected mixed dipole. Adapted from [100].

For generating random hyperplanes, a dipole $(\mathbf{x^i}, \mathbf{x^j})$ is randomly selected and the $H_{ij}(\mathbf{w}, \theta)$ hyperplane is generated by setting $\mathbf{w}$ and θ as:

$$\mathbf{w} = \mathbf{x^i} - \mathbf{x^j} \tag{12}$$

$$\theta = \delta(\mathbf{w^T x^i}) + (1-\delta)(\mathbf{w^T x^j}) \tag{13}$$

where $\delta \in \{0, 1\}$ is a random coefficient that scales the distance to the opposite ends of the dipole. Figure 6 presents this rationale.

The fitness function in GEA-ODT is a weighted-formula that penalizes model complexity (tree size). It is given by

$$f(I) = Q(I) - (\alpha \times size(I)) \tag{14}$$

where $Q(I)$ is the quality measure over the training set (we assume accuracy is a suitable quality measure for the purpose of maximizing $f(I)$), and $size(I)$ is the size of individual I (total number of nodes). The parameter α scales the importance of the penalty term.

In GEA-ODT, mutation occurs with a 1% probability. It can either alter the node role or its corresponding hyperplane. More specifically, a non-terminal node can be pruned to a leaf or have its hyperplane modified. A hyperplane can be modified by either standard mutation (perturbation of the weights that form the hyperplane) or by the dipole operator. The dipole operator is a task-dependent knowledge based operator that seeks to shift the hyperplane in order to divide mixed dipoles or unite pure dipoles. Finally, when dealing with a leaf node, the mutation operator can swap this node for a new non-terminal node. Standard one-point crossover in trees is performed. Selection is performed through linear ranking with elitism.

V. Regression

Decision trees for regression are an effective approach for predicting continuous values while enabling the analysis of the variables responsible for that particular prediction. They are usually regarded as either regression trees or model trees, according to the content of their leaf nodes. We review EAs for inducing regression or model trees next.

A. Regression Trees

A regression tree is a special type of decision tree where the target attribute is continuous. Thus, each leaf node of the regression tree holds a continuous value instead of a class label. This continuous value is the average value for the target attribute of all instances that reach that particular leaf node.

TARGET (Tree Analysis with Randomly Generated and Evolved Trees) [102], [103] is an EA that evolves regression trees where each candidate solution is an axis-parallel regression tree of variable size and shape. The initial population consists of 25 randomly created trees. The authors report that experiments performed with forest sizes between 10 and 100 trees, showed no apparent impact on the outcome. The random generation of trees starts with a single root node, that has a probability p_{split} of becoming a split. Otherwise, it will be a leaf. If the node bocomes a split, an attribute from the data set and a split value are randomly chosen from a pool of candidate attributes and split values and two child nodes are created. This procedure is repeated until no more nodes can be split.

For evaluating the trees, TARGET measures their fitness by using the Bayesian information criterion (BIC), a statistical model selection criterion based on likelihood [104]. It is a weighted-formula that penalizes model complexity (size of the tree). It is expressed as

$$BIC = -\frac{m}{2} ln\, 2\pi - \frac{m}{2} ln \frac{SSE}{m} - \frac{m}{2} - \frac{1}{2} p \log m \qquad (15)$$

where p is the effective number of parameters in the tree model, m is the size of the training set and SSE is the residual sum of squares. The last term in (15) is the model complexity penalty, which is a function of both the effective number of parameters p and the training sample size m.

One critical problem of using the BIC criterion in TARGET is determining the value of p, an essential part of the model complexity penalty term. Large values of p may lead to smaller trees with less predictive performance. The authors defined $p = nt + 1$ to account for estimating the constant error variance term and a mean parameter within each of the nt terminal nodes. Even though they acknowledge the fact that the effective number of parameters estimated is actually much higher than $nt + 1$, due to split rule selections made during the tree construction process, they state that further research is required to determine the appropriate adjustment of the model complexity penalty term.

Kretowsky and Czajkowski [105] propose an EA for regression tree induction called GRT (Global induction of Regression Trees). Each candidate solution is an axis-parallel regression tree. The trees in the initial population are initialized through traditional top-down regression tree algorithms such as M5 [106], [107] and CART [5] on random subsamples from the original data set. The recursive partitioning of the initial trees stops when: (i) all training objects in a node have the same predicted value; (ii) the number of instances in a node is lower than a predefined value; or (iii) the predefined maximum tree depth is reached.

GRT's crossover operation has three variants: (i) subtrees are swapped between two individuals, with random selection of nodes; (ii) the split test between two non-terminal nodes of different individuals are swapped, though keeping their subtree structures; and (iii) branches from two randomly selected nodes in two different individuals whose subtrees have the same size are swapped.

Mutation is stochastically performed (the default probability is 0.8) in either non-terminal nodes or leaves. In non-terminal nodes, nodes in higher levels of the tree are mutated with a lower probability than those in lower levels. Among nodes in the same level, the absolute error is calculated for the node subtree in order to rank nodes: a higher value of absolute error increases the probability a node being mutated. The same ranking procedure is applied to leaf nodes (except for pure nodes). For non-terminal nodes, mutation can assume the following variants: (i) replacement of a split test according to a traditional split search mechanism; (ii) shifting splitting thresholds (for continuous attributes) or regrouping attribute values (for nominal attributes); (iii) replacement of a split test by a pre-existing test; (iv) replacement of a subtree by another subtree from the same node; (v) replacing a subtree by a leaf node. Mutation may transform leaf nodes into non-terminal nodes if the instances contemplated by that particular node are heterogeneous.

The selection mechanism adopted by GRT is linear ranking, and the best individual of each generation is kept to the next (elitism). The fitness function searches for a compromise between mean absolute error and tree size, as follows:

$$f(I) = \frac{MAE(I)}{MAE_{max}} + \alpha(size(I) - 1) \qquad (16)$$

where $MAE(I)$ stands for the mean absolute error of individual I over the training set, MAE_{max} is the mean absolute error of the entire learning set, and $size(I)$ is the size of individual I. The parameter α weights the relative importance of the complexity penalty and subtracting 1 from the complexity term eliminates the penalty factor for trees that consist of a single leaf.

Hazan et al. [108], [109] propose a strongly-typed GP approach (STGP)[3] for axis-parallel regression tree induction. Although they focus on inducing regression trees for modeling expressive performance (musical computation), their approach is easily generalized to any domain. In STGP function nodes can be either a test over an attribute (e.g., a "less than" operator, hereby called **LT**), an if-then-else construct (**IF**) or a constant generator of either attribute or regression values, namely **EFV** and **ERV**, respectively. The **LT** construct operates over two arguments of fixed types: *FeatValue* and *RegValue*, and returns a *boolean* value. **IF** operates over three arguments: a *boolean*, resulting from the constructor **LT** and two arguments of *RegValue* type. **IF** returns a *RegValue* value, that can be either a constant or any other construct. Terminal nodes are constants representing either attribute values or regression values (target attribute outputs), generated by either **EFV** or **ERV** functions. Figure 7 demonstrates a typical tree from STGP.

[3]STGP was designed within the Open Beagle Framework for evolutionary computation [110].

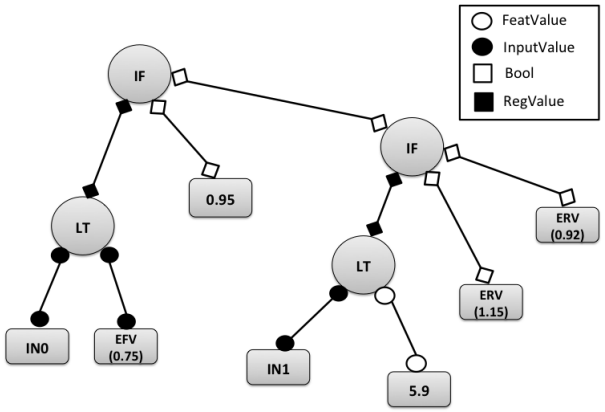

Fig. 7. STGP regression tree example. Typed connections are presented. Adapted from [109].

The population size of STGP is fixed in 200 individuals, and the algorithm halts when either the limit of 500 generations has been reached or the best solution fitness is ≥ 0.95. Tournament selection is the strategy for selecting individuals to reproduce. A typical subtree swap crossover is performed with a 0.9 rate. Mutation can either swap a subtree with a newly generated one (0.01 rate), replace a branch by one of its children (0.05), swap subtrees from the same individual (0.1) or disturb constant values (rate not informed). A maximum depth of 10 levels is set, though the authors state that they do not search for low complexity (smaller sized) solutions.

STGP's fitness function is given by the inverse of RMSE (root mean squared error), $f(I) = 1/1 + RMSE(I)$, where RMSE is taken over the training set and predicted duration ratio, averaged over the notes of a musical fragment, itself averaged over all the training fragments. For more conventional applications, the RMSE could be taken directly over the predicted values of the individual I. The authors also present domain-specific fitness functions not presented here due to space constraints.

B. Model Trees

Model trees are a special case of decision trees also developed for regression problems. The main difference between a model tree and a regression tree is that whereas each leaf node in a regression tree outputs a single continuous value, in model trees each leaf node holds a (non-) linear model whose output is the final prediction value.

GPMCC [111] (Genetic Programming approach for Mining Continuous-valued Classes) is a framework proposed to evolve model trees with non-linear models in their leaves. Its structure is divided into three different parts: (1) GASOPE [112], a GA that evolves polynomial expressions; (2) K-Means [113], a traditional clustering algorithm used to sample the training set; and (3) a GP to evolve the structure of model trees. Trees are generated by randomly expanding a node and randomly selecting attributes and split values. Tree growth is dictated by a parameter that indicates the maximum tree depth. The fitness function used by GPMCC is an extended form of the *adjusted coefficient of determination* (17)

$$R_a^2 = 1 - \frac{\sum_{i=1}^{m}(y^i - y'^i)^2}{\sum_{i=1}^{m}(y^i - \overline{y})^2} \times \frac{m-1}{m-d} \tag{17}$$

where m is the size of the set, y^i is the actual output of the i^{th} instance, y'^i is the predicted output for the same instance, $\overline{y}$ is the average output of all instances and d is a complexity factor that penalizes both the size of an individual (number of nodes) and the complexity of each model of the terminal nodes (number of terms and their corresponding order). The higher the value of complexity factor d, the lower the value of R_a^2. The best individuals are those with the higher values of R_a^2.

One important remark is that GPMCC has a total of 42 configurable initialization parameters that control several steps of the algorithm. The authors claim that the influence of GPMCC parameter values on the performance obtained was empirically investigated and that it was shown that the performance was not significantly affected by different parameter values. Based on these results, it is not clear the real utility of so many configurable options. As a second point of criticism, GPMCC seems to be overly-complex, and the results do not seem to justify all the choices made during its development.

Barros et al. [46], [47] propose an EA called E-Motion (Evolutionary MOdel Tree InductiON) for axis-parallel model tree induction, where each individual is represented as a tree of variable shape and size.The initialization of individuals is domain knowledge-based, as it combines single nodes whose attribute tests are dictated by the expectation of standard deviation reduction (SDR), given by

$$SDR = sd(D) - \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times sd(D_i) \tag{18}$$

where $sd(.)$ stands for the standard deviation, D is the data set that reaches the node to be divided, D_i is the data resulting from the i^{th} split (in a total of k splits) and $|.|$ is the size of a specific data partition. It is easy to notice that the split rule that yields the lower standard deviation values for the child nodes will maximize (18). E-Motion generates a split rule for each data set attribute that maximizes (18). The initial trees are random combinations of these basic trees that consist of a single node (attribute + split rule).

E-Motion allows the user to choose between two types of multi-objective optimization in the fitness function. The first one is a weighted-formula that accounts for RMSE (root mean squared error), MAE (mean absolute error) and tree size, where each measure has a user-defined weight. The second one involves a lexicographic analysis, where these three measures are ranked based on the user's priorities. In this case, user-defined thresholds may define whether the highest-priority measure is enough to select the best individual or if the subsequent measures should also be evaluated.

E-Motion implements standard one-point crossover in trees. Two different mutation strategies are used to variate individuals' sizes: a shrinking mutation, where a subtree is replaced by a leaf node, and an expanding mutation, where a leaf node is replaced by a two-level subtree. Finally, a filter is applied to guarantee consistency of the linear models at each leaf node.

VI. Decision Tree Components

All EAs reviewed so far evolve individuals which represent decision trees. In this section, we discuss EAs with a different purpose: to improve a decision tree classifier's component. Hence, they do not evolve decision trees per se, but components of decision tree classifiers. We divide such EAs into 3 groups, according to the type of component being evolved: (i) Hyperplanes; (ii) Pruning Methods; (iii) Other Components.

A. Hyperplanes

EAs evolving full oblique trees were reviewed in Section IV-B. However, some EAs evolve only a hyperplane for each node of the oblique tree. More specifically, each individual in these EAs is a combination of attributes and constants that define a hyperplane.

In [13], [14], Chai et al. propose a binary linear decision tree approach for piecewise linear classification, named BTGA. At each non-terminal node of the tree, a GA searches for a linear decision function, optimal in the sense of maximum impurity reduction. We can formalize the impurity reduction as follows. Assume $X^t = \{\mathbf{x^1}, \mathbf{x^2}, ..., \mathbf{x^{m_t}}\}$ as the training subset that falls at the current node t with sample size m_t and $\mathbf{x^i} = \{x_1^i, x_2^i, ..., x_n^i, 1\}^T$ the augmented attribute vector with dimension n. Besides, assume that $X^t = X^{t_1} \cup X^{t_2} \cup ... \cup X^{t_k}$, where k represents the total number of classes and X^{t_i} represents all instances in X^t that belong to class ω_i. Thus the impurity of subset X^t can be defined as:

$$i(X^t) = \sum_{i=1}^{k} \sum_{j \neq i} p(\omega_i|X^t) p(\omega_j|X^t) \tag{19}$$

where

$$p(\omega_i|X^t) = \frac{|X^{t_i}|}{|X^t|}. \tag{20}$$

Notice that a pure division, i.e. $i(X^t) = 0$, will only happen when all instances in X^t belong to a same class. Assume now that a linear decision function $\mathbf{w^T x}$ is responsible for splitting X^t into two subsets, X^{t_L} and X^{t_R}, i.e.,

$$X^t = X^{t_L} \cup X^{t_R} \tag{21}$$

and

$$X^{t_L} = \{\mathbf{x} | \mathbf{x} \in X^t \wedge \mathbf{w^T x} < 0\} \tag{22}$$

$$X^{t_R} = \{\mathbf{x} | \mathbf{x} \in X^t \wedge \mathbf{w^T x} \geq 0\} \tag{23}$$

where $\mathbf{w} = \{w_1, w_2, ..., w_n, w_{n+1}\}^T$ is the weight vector with dimension $n+1$. The total impurity of splitting X^t in X^{t_L} and X^{t_R} is

$$i'(X^t, \mathbf{w}) = p(X^{t_L}|X^t) i(X^{t_L}, \mathbf{w}) + p(X^{t_R}|X^t) i(X^{t_R}, \mathbf{w}) \tag{24}$$

where

$$p(X^{t_K}) = \frac{|X^{t_z}|}{|X^t|}, \text{ for } z = \{L, R\}. \tag{25}$$

The impurity reduction is finally defined as

$$\Delta i(X^t, \mathbf{w}) = i(X^t) - i'(X^t, \mathbf{w}). \tag{26}$$

Hence, we can formalize the problem of inducing a binary linear decision tree as finding the vector $\mathbf{w}^*$ that maximizes the impurity reduction at node t, i.e.,

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \Delta i(X^t, \mathbf{w}). \tag{27}$$

A GA is applied to globally search the solution-space for the best possible combination of values in $\mathbf{w}$, which is $\mathbf{w}^*$. In this GA, $\mathbf{w}$ is encoded as a binary-string, such that the gene A^i encodes the i^{th} possible value of $\mathbf{w}$ and the chromosome A consists of the union of genes, i.e., $A = A^1 \cup A^2 ... \cup A^{n+1}$.

The fitness function is given by the impurity reduction (26), calculated for each chromosome. The selection scheme is the roulette wheel, where chromosome A_j is assigned a selection probability $f(A_j)/\sum_{i=1}^{nc} f(A_i)$, where nc is the total number of chromosomes and $f(.)$ is the fitness function. The selected chromosomes participate in a two-point crossover, an effective alternative to the traditional one-point crossover, since more chromosome substructures can be preserved and combined. Mutation is given by simply bit-flipping genes with a low-probability.

Palaniappan et al. [114] extended BTGA (BTGA+) by implementing three different impurity measures (namely Gini index, information gain and twoing rule). Besides, they investigated a Bayesian initialization of the individuals (weights) with randomization process. The Bayesian initialization uses the common covariance weighted vector between the means of the dominant class, ω_0, and the remaining node examples, ω_1, with $\mathbf{w} = \Sigma^{-1}(\mu_0 - \mu_1)$ and an independent parameter, w_0, given by:

$$w_0 = -\frac{1}{2}[(\mu_0^T \times \Sigma^{-1} \times \mu_0) - (\mu_1^T \times \Sigma^{-1} \times \mu_1)] + \ln \frac{p(\omega_0)}{p(\omega_1)} \tag{28}$$

where Σ^{-1} is the inverse covariance matrix (concentration matrix).

Kim et al. [15], [16] propose a hybrid approach that uses a GA for selecting hyperplanes in each node of a binary oblique decision tree. In this work, the GA performs feature selection so as to reduce the classification error rate in a binary decision tree. The GA looks for the optimal linear combination of features able to divide the instances in two disjunct subsets. This process is carried out at each node of the binary tree that is being created. Thus, for each node, a linear combination of features selected by the GA divides the attribute space into two different subsets until each subset has instances belonging to a unique class.

Each individual in the GA is a n-dimensional linear decision function. A binary string representation encodes the linear decision function, and each string has $n^2 \times \gamma$ bits, where n is the number of dimensions and also the number of segments necessary to encode a single feature, and γ is the length of each segment. The fitness function is given by:

$$f(I) = \frac{1}{1 + (w_e \times error) + (w_b \times balance)} \tag{29}$$

and

$$balance = \sqrt{\frac{\sum_{j=1}^{2}(m_j - \frac{m}{2})^2}{(\frac{m}{2})^2}} \quad (30)$$

where m is the total number of instances, m_j is the number of instances that reach the j^{th} node, $error$ is the classification error and w_e and w_b are weights associated to error and balance, respectively. In these equations, $balance$ is the balance coefficient whose values tend to decrease when the number of instances in each group become similar.

Cantu-Paz et al. [115], [116] also investigate the use of EAs for designing optimal hyperplanes. They propose two different strategies for expanding OC1 [117] (an oblique decision tree induction algorithm): (i) a (1+1) evolution strategy (ES) with self-adaptive mutations (named OC1-ES); and (ii) a GA with real-valued genes (named OC1-GA).

OC1-ES evolves a single individual[4], a vector of real-valued coefficients, $w_1, w_2, ..., w_{N+1}$, for a n-dimensional data set. The individual is initialized with the best axis-parallel split found by OC1. For each hyperplane coefficient, there is a corresponding mutation rate, $\sigma_1, \sigma_2, ..., \sigma_{n+1}$, initially set to 1. At each iteration of the evolution strategy, the mutation rates are updated and so are the coefficients, according to the following rule:

$$\begin{aligned} \upsilon &= N(0,1) \\ \sigma_i^{t+1} &= \sigma_i^t exp(\tau' \upsilon + \tau \upsilon) \\ w_i^{t+1} &= w_i^t + \sigma_i^{t+1} \upsilon \end{aligned} \quad (31)$$

where $N(0,1)$ is a realization of a unitary normal variate, $\tau = (\sqrt{2\sqrt{n}})^{-1}$ and $\tau' = (\sqrt{2n})^{-1}$. The ES is run for 1000 iterations.

OC1-GA evolves a population of individuals represented by a real-valued set of coefficients (the same used in OC1-ES). Tournament selection without replacement selects the individuals that will be subjected to uniform crossover (probability of 100% to happen). No mutation is implemented in OC1-GA. The population size is set to $20\sqrt{n}$, along the lines of a population-sizing theory described in [118][5]. To generate the initial population of individuals, the best axis-parallel hyperplane is assigned to 10% of the individuals, and the remaining 90% are randomly generated in the range $[-200, 200]$. The fitness is evaluated by the calculus of the impurity measure twoing (I_{two}) [5], given by:

$$I_{two} = \frac{m_L}{m} \times \frac{m_R}{m} \times \left(\sum_{i=1}^{k} \frac{L_i}{m_L} - \frac{R_i}{m_R} \right)^2 \quad (32)$$

where m is the total number of instances under consideration in each node, m_L (m_R) is the number of instances in the left (right) portion of the split, and L_i (L_R) is the number of instances belonging to class i on the left (right) portion of the split.

[4]Note that this ES, unlike GAs or GP, evolves a single individual instead of a population

[5]Harik et al. state that the population size required to reach a solution of a particular quality is $O(\sqrt{n})$

Kretowsky [119] proposes an EA that evolves hyperplanes for oblique decision trees. The proposed approach is based on the dipole concept (see Section IV-B). The oblique decision tree is built through a top-down approach, and the "optimal" decision functions is selected at each tree node. For evolving decision functions, a fixed-length real-valued chromosome of size $n+1$ represents the hyperplane (n weights and θ). For generating the initial population, a random mixed dipole ($\mathbf{x^i}, \mathbf{x^j}$) is selected and the hyperplane $H_{ij}(\mathbf{w}, \theta)$, which is perpendicular to the segment that connects the opposite sides of the dipole (placed in halfway), is formed by setting $\mathbf{w} = \mathbf{x^i} - \mathbf{x^j}$ and $\theta = 1/2[(\mathbf{w^T x^i} + \mathbf{w^T x^j})]$. The fitness function is given by

$$f(x) = f(\mathbf{w}, \theta) \times [(1-\beta) + \beta \frac{n'}{n}], \text{ where} \quad (33)$$

$$f(\mathbf{w}, \theta) = f_{mixed} + \alpha(1 - f_{pure}) \quad (34)$$

where f_{mixed} (f_{pure}) is the fraction of divided mixed (pure) dipoles, α controls the importance of pure dipoles, $\beta \in \{0, 1\}$ defines the complexity of the test (in terms of number of parameters), n' is the number of non-zero weights and n is the dimensionality of the problem.

Standard two-point crossover is employed and the mutation operator is slightly modified in order to increase the chances a gene can be set to 0. Since the weights are real-valued, chances of eliminating the importance of one attribute (i.e., setting it to 0) are slim, so the mutation operator is modified to enhance the chances of feature selection (eliminating attributes). Additionally, a dipolar operator is implemented, as follows. First, the dipole type is drawn (mixed or pure). If the mixed type is selected, one dipole is drawn from the set of non divided mixed dipoles and the hyperplane is shifted to separate the pair of instances. The new position is obtained by modifying only one randomly chosen attribute. If it is the pure type, one dipole is drawn from the set of divided pure dipoles. The hyperplane is shifted to avoid separation of objects from the same class by once again modifying one randomly chosen weight. Selection is made through linear ranking and the best individual from a generation is kept to the next (elitism).

Shali et al. [120] propose a GP, named GIODeT for evolving combinations of attributes for each node split. The C4.5 algorithm [4] is used for constructing the decision tree in a top-down fashion, but instead of using the typical univariate tests at each node, a GP evolves a set of relations for combining the data set attributes. The authors propose the following functions for the internal nodes of GIODeT:

- Mathematical: $+, -, \times, /, \log, \sqrt{}$
- Relational: $\leq$
- Logical: $nand, not$

An expression generator component is responsible for combining these unary and binary functions with the set of nominal and numeric attributes, generating a mathematical expression with arbitrary size. Note that the GP is run once for each node of the decision tree. Tournament selection is performed on 15 randomly selected individuals in order to choose those that will undergo crossover and mutation. Standard GP crossover and mutation are applied with probabilities of 0.65 and 0.2,

respectively. The fitness function is given by

$$f(I) = GainRatio \times \left(GainRatio + \frac{1}{size(I)} \right) \quad (35)$$

where $GainRatio$ is the standard split criterion implemented in C4.5 [4] and $size(I)$ is a function that returns the total number of nodes of the GP individual I.

B. Pruning Methods

Pruning is an important component of decision tree induction algorithms, because it can avoid model overfitting. Given a hypothesis space H, a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$ such that h has a smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

Overfitting is particularly critical in decision tree induction, since decision trees can perfectly classify the training examples. If there is noise in the training set, the induced decision tree will learn how to classify it and thus will perform poorly on unseen data. Lack of representative samples in the training data can also lead to model overfitting, because the resulting decision tree will make its classification based on small number of instances. Common approaches for avoiding overfitting are prepruning (halting the algorithm before generating a fully grown tree that perfectly fits the data) and post-pruning (grow a full decision tree and later pruning subtrees according to error estimates). Evolutionary alternatives to decision tree pruning are reviewed next.

Chen et al. [17] propose a GA for decision tree pruning where each individual is a fixed-length linear chromosome. Each gene dictates whether a subtree should be pruned or not. More specifically, a decision tree is fully grown by a traditional decision tree induction algorithm (ID3 [3]) and then linear chromosomes with size equal to the number of edges of the full tree are randomly generated. One-point crossover and bit-flip mutation are implemented (rates are not informed). The fitness function to be minimized is given by the sum of the total number of nodes and the error rate. Surprisingly, during fitness evaluation, the error rate is said to be calculated over the test set (a serious experimental mistake, since the test set class labels are supposed to be unknown, and can only be used to validate the best individual of the EA, and not every individual in each generation).

Shah and Sastry [18] propose a new pruning algorithm for oblique decision trees in binary classification problems using either an automata learning model (LA) or a GA. They map the decision tree pruning as boolean-vector learning problem. For such, each decision tree is mapped into a 3-layer feedforward neural network. Each node in the first hidden layer consists of one hyperplane (split rule) and its complement. Each node in the second layer represents a leaf node labeled as class 1. The third layer consists of a single OR unit.

Formally, the first layer has M units, and the i^{th} unit represents a hyperplane H_i, parametrized by $\mathbf{w_i} = [w_{i0}, ..., w_{in}] \in \Re^{n+1}$, $1 \leq i \leq M$, assuming an n-dimensional space of attributes. The output of the i^{th} unit over a given input instance

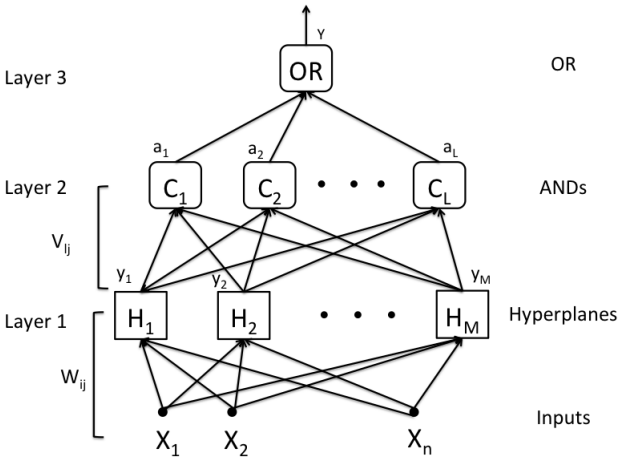

Fig. 8. Three-layer feedforward neural network representation. Adapted from [18].

$\mathbf{x} = [x_1, x_2, ..., x_n] \in \Re^n$ is $y_i = 1$ if $\sum_{j=1}^{n} w_{ij}x_j + w_{i0} > 0$ and $y_i = 0$ otherwise.

The second layer has L units, each one implementing an AND function. The l^{th} unit is connected to all first-layer units through a weight vector $\mathbf{v_l} = [v_{l1}, v_{l2}, ..., v_{lM}]$, where $v_{li} \in \{0, 1\} \forall i$, $1 \leq i \leq L$.

The output for the l^{th} unit is $a_l = 1$ if $\forall i, y_i = 1$ and $v_{li} = 1$, otherwise $a_l = 0$. The third layer outputs 1 if at least one of the second-layer unit outputs is 1, and 0 if all second-layer outputs are 0 (OR operation).

Each second-layer unit is connected (with weight 1) to all hyperplanes in the first layer that appear on the path from the root to the leaf node corresponding to this unit. All the other connections will have weight 0. It is easy to see that the final output of the network will be 1 on all patterns classified as belonging to class 1 by the decision tree. Given this framework, pruning a decision tree consists of learning the boolean-vector $\mathbf{V_l}$ that maximizes some measure (e.g., accuracy). Figure 8 presents the proposed 3-layer neural network.

Note that the hyperplanes in the first-layer are those discovered during the full growth of a decision tree by any oblique decision tree induction algorithm. Besides, note that the number of second-layer units (L) can be fixed heuristically according to the desired level of pruning. Different values for $\mathbf{V_l}$ will lead to different ways to restructure the decision tree. This strategy is much more sophisticated than simply replacing a subtree by a leaf node, because it can lead to a more drastic restructuring of the original tree. A GA is used to evolve a set of fixed-length binary strings that represent the $\mathbf{V_l}$ vector. The authors mention that they applied "standard" crossover and mutation operations (no further details were informed).

C. Other Decision Tree Components

Turney [19] proposes a hybrid approach, ICET (Inexpensive Classification with Expensive Tests), where a GA is used to evolve a population of biases for a decision tree induction algorithm (variation of Quinlan's C4.5). ICET considers both

the cost of tests (i.e., cost of attributes) and the cost of classification errors.

In ICET, instead of using C4.5's gain ratio, attributes are evaluated by the information cost function (ICF) [121], defined for each attribute i as follows:

$$ICF_i = \frac{2^{\Delta I_i} - 1}{(C_i + 1)^{\alpha}} \quad (36)$$

where ΔI_i is the information gain associated to the i^{th} attribute when splitting a given portion of the data set, C_i is the cost of measuring the i^{th} attribute and $\alpha = [0, 1]$ is the strength of the bias towards lower cost attributes. Note that $\alpha = 0$ means the attribute cost is ignored and maximizing ICF is equivalent to maximizing ΔI_i, whereas $\alpha = 1$ means ICF is strongly biased by cost.

ICET evolves a population of individuals encoded as a fixed-length binary string consisting of $12 \times n + 16$ bits, where n is the number of data set attributes. Each attribute cost C_i in (36) is encoded as a 12-bit value. In addition, each string also encodes the parameter α in (36) and the pruning confidence factor for C4.5, both as 8-bit values. ICET uses a random initialization of individuals, a crossover rate of 0.6 and a mutation rate of 0.001.

The fitness function is the average cost of classification. To calculate the cost of a particular instance, we follow its path down the decision tree. We add up the cost of each attribute that is chosen (i.e., each test that occurs in the path from the root to the leaf). If the same test appears twice, we only charge for the first occurrence of the test. The leaf of the tree specifies the tree?s guess for the instance class. Given the actual instance class, we use the cost matrix to determine the cost of the classification. This cost is added to the costs of the tests, determining the total cost of classification for the instance. The total cost of classification of all instances are summed, and then divided by the total number of instances, resulting in the average cost of classification.

Bratu et al. [20] propose extending ICET by modifying components of the original GA. By introducing elitism, increasing the search variability factor, and extending the number of iterations, the proposed extension manages to outperform other cost-sensitive algorithms, even for data sets on which the initial implementation yielded modest results. Results appear to suggest there is an urge for a more comprehensive analysis on the impact of varying the GA parameters, as it is empirically demonstrated that such changes can lead to significant performance improvements.

VII. Performance Analyses

Many of the papers that we review include performance comparisons of decision trees obtained by means of evolutionary algorithms and other traditional approaches. The most well-known greedy decision tree induction algorithms used in these comparisons are ID3 [3], C4.5 [4], CART [5], M5 [106], [122], REPTree [123] and OC1 [117]. Other algorithms used for comparison purposes include Bayesian CART [124], [125], EG2 [121], CS-ID3 [126]–[128] and IDX [129]. Most of these comparisons make use of public data sets from the UCI repository [130].

TABLE I
Application areas of EAs for decision tree induction.

Application area	References
Astronomy	[133]
Cold Mill Strip	[15], [16]
Character Recognition	[35]–[37], [87], [88]
Finance	[134]
Marketing	[40], [42]
Medicine	[23], [50], [78], [80], [135]–[138]
Natural Language Processing	[139]
Software Engineering	[22], [77], [140]–[142]

As it would have been expected, no work reviewed in this paper has been shown to be superior than other methods for every tested data set. This is coherent with the no free lunch theorem [131], which states that learners which excel in a certain data set will perform poorly in others.

Nevertheless, many works affirm to present similar predictive performance to baseline algorithms while providing smaller trees (e.g., [17], [46], [47], [100], [102], [103], [105], [111]). Only one work [120] states that the proposed EA generates larger trees than its baseline algorithm (C4.5). The ability of generating smaller trees is mainly due to the fitness functions of the EAs, which usually incorporate a size penalty for avoiding large trees. This explicit mechanism for reducing tree size is indeed one of the advantages of using EAs for decision tree induction.

Regarding computational time analyses, most of the reviewed works state that the proposed approaches are much more time-consuming than the greedy strategy, as it is the case of most evolutionary algorithms. Notwithstanding, very few papers report runtimes. Among those few works, some report a difference of seconds for growing the decision tree (e.g., [41]), whilst others report differences of minutes (e.g., [42], [43], [70], [97]) or hours (e.g., [38], [44], [100]). The time complexity of EAs for evolving decision trees is not detailed in the reviewed papers, but it is estimated to be much higher than the complexity of typical top-down decision tree induction algorithms. For more details on the time complexity of traditional algorithms such as C4.5 please refer to [132].

VIII. Applications of EAs for Decision Tree Induction

Considering that most works focus on the EAs ability for dealing with the drawbacks of the greedy strategy, usually these approaches are validated on public benchmark data sets for comparison purposes, instead of focusing on a specific application domain. Nevertheless, there are some papers that limit their scope to a specific domain. In these cases, we can highlight the application areas listed in Table I, whose EAs are briefly reviewed next.

In [133], a GA for decision tree induction is applied to an astronomy problem, the classification of galaxies with a bent-double morphology. This EA is the same proposed in [115] for inducing oblique decision trees. Kim et al. [15], [16] present a method for recognizing various defect patterns of a cold mill strip by using evolutionary binary decision trees.

Several EAs evolve binary decision trees for character recognition (classification). Works like [35]–[37], [87], [88] evolve binary decision trees using GAs and later apply the evolved decision tree to digit recognition tasks. Kuo et al. [134] evolve decision trees through GP and analyze their performance over a credit card data set.

Several EAs evolve decision trees for specific medical problems. For instance, Podgorelec and Kokol [135] investigate a multi-objective weighted-formula based on GAs for decision tree induction for detecting cardiac prolapse. Its fitness function incorporates domain knowledge by not considering all errors equally. The most costly errors are made when a patient with cardiac prolapse or silent prolapse is classified as with no prolapse. Conversely, it is not so costly when healthy patients are classified as with prolapse or silent prolapse.

In [78], the authors propose an integrated computerized environment, DIAPRO, which is a computer tool based on EAs for medical diagnosis. DIAPRO attempts to improve accuracy, sensitivity and specificity for decision making using evolutionary generated decision trees. In [80], a real-world orthopedic fracture data set is investigated through multiple approaches of decision tree induction, and it is shown that the evolutionary approach presents the best compromise among the measures evaluated.

In [136], the authors present a new outlier prediction system for improving the classification performance in medical data mining. Two cardiovascular data sets are investigated. The method introduces the class confusion score metric that is based on the classification results of a set of classifiers. These classifiers are generated by the EA proposed in [22], and the main idea is to verify if there is any inconsistence when classifying a specific example through different classifiers. In [23] a GP evolves decision trees for cardiac diagnoses. The GP was tested by using cardiac single proton emission computed tomography images.

In [137], a GP evolves decision trees to detect interactions in genetic variants. Preliminary experiments using GPDTI have been able to find a 3-marker interaction in a data set of 1000 markers and a sample size of 600 subjects. Smith [50] evolved decision trees in a bioinformatics application, where they used a GA for finding RNA family-specific decision trees. This work is well adapted to the application domain, i.e. both individual representation and fitness function are domain-specific.

Siegel [139] proposes evolving decision trees for the problem of word sense disambiguation. The approach Siegel takes to word sense disambiguation is to evolve decision trees which attempt to establish word sense by looking only at immediate context, i.e., the tokens, (words and punctuations marks) which are located within a small distance of the word to be disambiguated. The words being disambiguated are *discourse cue words*. Results indicate that evolved decision trees often include rules that provide insightful hints for linguists.

Many works propose evolving decision trees for software engineering tasks. For instance, in [143] an EA for decision tree induction is used as a software fault predictive approach. It is shown that software complexity measures can be successfully used to evolve decision trees for predicting dangerous modules (with many undetected faults). The authors recommend redesigning the fault-detected modules or devoting more testing or maintenance effort to them in order to enhance the software quality and reliability. In [22], Podgorelec and Kokol present a self-adapting EA for decision tree induction and its application for predicting faults in dangerous software modules. Khoshgoftaar et al. [140] present an automated and simplified GP-based decision tree modeling technique for the software quality classification problem. The proposed technique is based on multi-objective optimization using strongly typed GP. In the context of industrial high-assurance software system, two fitness functions are used: one for minimizing the predictive error and another for parsimony.

Khoshgoftaar and Liu [141] propose a GP-based multi-objective optimization modeling technique for calibrating a goal-oriented software quality classification model geared toward a cost-effective resource utilization. It is shown that the proposed model achieves good performance in the context of optimization of the three modeling objectives: 1) minimizing a modified expected cost of misclassification measure for software quality classification models; 2) enabling the number of predicted fault-prone modules to be equal to the number of modules which can be inspected by the allocated resources; and 3) controlling the size of the decision tree to facilitate comprehensibility in model interpretation, and providing faster GP-runs. In [142], the authors present an evolutionary approach for decision tree induction and its application for classifying software modules as defective or defect-free. A set of 21 predictive attributes, containing various software complexity measures and metrics for each software module was used. The authors concluded that the discovered decision rules are useful for developing new rules in software defect identification.

IX. Remarks on Evolutionary Induction of Decision Trees

Decision trees are one of the most frequently used representations for classifiers. A very large number of articles have been devoted to their study. Whereas most well-known algorithms for decision tree induction rely on a greedy divide-and-conquer strategy for partitioning the tree, alternative approaches have become more and more common in the past few years. More specifically, a fairly large amount of studies have been dedicated to the evolutionary induction of decision trees.

We presented in this paper a survey of articles that combine evolutionary algorithms and decision trees. We proposed a taxonomy to better organise the works in this area, conveniently separating studies that evolve decision trees from those that evolve components of decision tree classifiers. Moreover, we documented important steps of an evolutionary algorithm for decision tree induction and component evolution, which can support interested readers in the design of their own EAs for decision tree induction according to an extensive enumeration of design options and strategies. We presented, when suitable, criticism over specific choices and guidelines for handling problems that will eventually come up when designing EAs for decision tree induction. In this section, we make a brief critical

analysis on the advantages and disadvantages of evolutionary induction of decision trees. We finalize this paper by pointing out tendencies of future work in the area.

A. The Benefits of EAs for Decision Tree Induction

The main benefit of evolving decision trees is the EAs' ability to escape from local optima. EAs are able to perform a robust global search in the space of candidate solutions, and thus they are less susceptible to convergence to local optima. In addition, as a result of this global search, EAs tend to cope better with attribute interactions than greedy methods [10], which means that complex attribute relationships which are not detected during the greedy evaluation can be discovered by an EA when evolving decision trees.

Another advantage of evolving decision trees is the possibility of explicitly biasing the search space through multi-objective optimization. EAs can naturally optimize multi-objectives, which may be crucial in several application domains. For instance, cost sensitive classification - which is the case of most medical classification problems - can be enhanced by explicitly optimizing measures that address different error costs. Parsimony pressure is yet another important feature that is easily implemented in a multi-objective EA for decision tree induction.

B. The Drawbacks of EAs for Decision Tree Induction

The main disadvantage of evolutionary induction of decision trees is related to time constraints. EAs are a solid but computationally expensive heuristic. Nevertheless, progressively faster computational resources have allowed EAs to be increasingly used in a variety of applications over the years. Advances on parallel processing have also enabled EAs to be better explored in acceptable execution times. It should also be noted that, in real-world problems, the time spent preparing the data for data mining purposes tends to be much longer than the time spent by inducing a classification or regression model. Hence, in real-world problems, the long processing time of evolutionary algorithms tends not to be the bottleneck of the entire knowledge discovery process.

Another disadvantage is the large number of parameters that must be tuned in order to run a full execution of an EA. Espejo et al. [27] acknowledge this problem in GP-based classifiers, though they reckon it is a secondary one. The same difficulty occurs for genetic algorithms, since GAs and GP share much of the same parameters.

C. What is Next on Evolutionary Induction of Decision Trees?

The theoretical assumption that evolved decision trees can provide better predictive accuracy than greedily-designed ones has been confirmed empirically in most of the works reviewed in this paper. Nevertheless, this advantage comes with a price. Most strategies reviewed here point out that evolving decision trees is a costly, time-consuming task. Not much is said on how this can be alleviated. Parallel implementations are usually the suggestion for speeding up decision tree evolution, even though very few works actually implement parallel algorithms or care to make a comparison on the time savings. We believe that a future trend on evolutionary induction of decision trees is the analysis of techniques able to speed up evolution. A recent work by Kalles and Papagelis [144] proposes time savings in evolutionary decision tree induction algorithms through fitness inheritance. They reckon that storing instance indices at leaf nodes is enough for fitness to be piecewise computed in a lossless fashion, thus leading to substantial speed-ups within the evolutionary cycle. They show the derivation of the expected speed-up on two bounding cases and their claims are supported by an extensive empirical analysis. We believe that further exploration of this kind of analysis is beneficial in order to establish EAs for decision tree induction as a strong alternative to greedy methods.

Evaluation of EAs on synthetic data is a methodology that could be more investigated by researchers of the field. It allows for a more theoretical exploration that is complementary to the prevailing empirical evaluation of EAs for decision tree induction. Examples of works that make use of such a methodology are [39], [48], [54], [64], [73], [145]. In [145], for example, instances were artificially generated in order to produce data sets with specific characteristics, such as representative enough categorical distributions and extreme cases of easiness or hardness for decision tree induction.

Another future trend is the development of data-based parameter guidelines. Researchers in the area should start establishing the relationship between EAs' parameter settings and data sets' features. For that, we believe that the study of data set complexity measures, such as those proposed by Ho et al. [146], [147], may be beneficial for understanding the geometrical shape of classification data sets. Once the complexity of classification data is better-understood, we believe it would be possible for researchers to recommend parameter values for decision tree evolution backed up by theoretical and empirical evidence.

Finally, we believe that an important and challenging future trend in this area concerns evolving decision tree induction *algorithms*. In such an approach, the EA evolves a full decision tree algorithm (with loops and procedures for growing, pruning and evaluating candidate decision trees), rather than just evolving a decision tree classifier. Considering the different procedures for growing the trees (top-down, bottom-up, hybrid), and also the large number of splitting criteria and pruning techniques, we believe it is possible to develop an EA able to automatically design brand new decision tree induction algorithms. One of the advantages of this strategy is the possibility of tailoring an algorithm for a given domain, and instead of using a generic decision tree induction algorithm (such as C4.5), we can make use of an algorithm that was explicitly designed to be efficient in specific data sets.

ACKNOWLEDGMENT

The authors would like to thank Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) for funding this research.

REFERENCES

[1] J. Han, *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[2] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[3] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

[4] ——, *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

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

[6] R. L. De Mántaras, "A distance-based attribute selection measure for decision tree induction," *Mach. Learn.*, vol. 6, no. 1, pp. 81–92, 1991.

[7] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, 2nd ed., ser. Springer Series in Statistics. Springer, September 2009.

[8] G. Seni and J. Elder, *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*. Morgan and Claypool Publishers, 2010.

[9] A. A. Freitas, D. C. Wieser, and R. Apweiler, "On the importance of comprehensible classification models for protein function prediction," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 7, no. 1, 2010.

[10] A. A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

[11] ——, *Soft Computing for Knowledge Discovery and Data Mining*. Springer US, 2008, ch. A Review of evolutionary Algorithms for Data Mining, pp. 79–111.

[12] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer, October 2008.

[13] B.-B. Chai, "Binary linear decision tree with genetic algorithm," in *International Conference on Pattern Recognition (ICPR '96)*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 530–534.

[14] B.-B. Chai, T. Huang, X. Zhuang, Y. Zhao, and J. Sklansky, "Piecewise linear classifiers using binary tree structure and genetic algorithm," *Pattern Recognition*, vol. 29, no. 11, pp. 1905 – 1917, 1996.

[15] K. M. Kim, J. J. Park, M. H. Song, I. C. Kim, and C. Y. Suen, "Binary decision tree using genetic algorithm for recognizing defect patterns of cold mill strip," in *Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, A. Y. Tawfik and S. D. Goodwin, Eds. Springer Berlin, 2004, vol. 3060, pp. 461–466.

[16] ——, "Binary decision tree using k-means and genetic algorithm for recognizing defect patterns of cold mill strip," in *17th international conference on Innovations in applied artificial intelligence*. Springer Verlag Inc, 2004, pp. 341–350.

[17] J. Chen, X. Wang, and J. Zhai, "Pruning decision tree using genetic algorithms," in *International Conference on Artificial Intelligence and Computational Intelligence (AICI '09)*, 2009, pp. 244 –248.

[18] S. Shah and P. Sastry, "New algorithms for learning and pruning oblique decision trees," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 29, no. 4, pp. 494 –505, nov. 1999.

[19] P. D. Turney, "Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm," *Journal of Artificial Intelligence Research*, pp. 369–409, 1995.

[20] C. Bratu, C. Savin, and R. Potolea, "A hybrid algorithm for medical diagnosis," in *International Conference on Computer as a Tool (EUROCON)*, sep. 2007, pp. 668 –673.

[21] C. J. Burgess and M. Lefley, "Can genetic programming improve software effort estimation? a comparative evaluation," *Information and Software Technology*, vol. 43, no. 14, pp. 863 – 873, 2001.

[22] V. Podgorelec and P. Kokol, "Evolutionary induced decision trees for dangerous software modules prediction," *Information Processing Letters*, vol. 82, no. 1, pp. 31 – 38, 2002.

[23] C. To and T. Pham, "Analysis of cardiac imaging data using decision tree based parallel genetic programming," in *6th International Symposium on Image and Signal Processing and Analysis*, 16-18 2009, pp. 317 –320.

[24] S. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 21, no. 3, pp. 660 –674, may. 1991.

[25] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data Min. Knowl. Discov.*, vol. 2, no. 4, pp. 345–389, 1998.

[26] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers - a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 35, no. 4, pp. 476 – 487, nov. 2005.

[27] P. G. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 40, no. 2, pp. 121–144, 2010.

[28] M. Basgalupp, A. de Carvalho, R. C. Barros, D. Ruiz, and A. Freitas, "Lexicographic multi-objective evolutionary induction of decision trees," *International Journal of Bio-Inspired Computation*, vol. 1, no. 1/2, pp. 105–117, January 2009.

[29] J. R. Woodward, "GA or GP? that is not the question." in *IEEE Congress on Evolutionary Computation*, 2003, pp. 1056–1063.

[30] A. A. Freitas, "A critical review of multi-objective optimization in data mining: a position paper," *SIGKDD Explor. Newsl.*, vol. 6, no. 2, pp. 77–86, 2004.

[31] C. Coello, "A comprehensive survey of evolutionary-based multiobjective optimization techniques," *Knowledge and Information Systems*, vol. 1, no. 3, pp. 129–156, 1999.

[32] C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[33] M. C. Bot and W. B. Langdon, "Application of genetic programming to induction of linear classification trees," in *Genetic Programming*, ser. Lecture Notes in Computer Science, R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, and T. C. Fogarty, Eds. Springer Berlin / Heidelberg, 2004, vol. 1802, pp. 247–258.

[34] M. J. Aitkenhead, "A co-evolving decision tree classification method," *Expert Syst. Appl.*, vol. 34, no. 1, pp. 18–25, 2008.

[35] M. Shirasaka, Q. Zhao, O. Hammami, K. Kuroda, and K. Saito, "Automatic design of binary decision trees based on genetic programming," in *Second Asia-Pacific Conference on Simulated Evolution and Learning*, Australian Defence Force Academy, Canberra, Australia, 1998.

[36] Q. Zhao and M. Shirasaka, "A study on evolutionary design of binary decision trees," in *IEEE Congress on Evolutionary Computation*. Mayflower Hotel, Washington D.C., USA: IEEE Press, 6-9 1999, pp. 1988–1993.

[37] T. Tanigawa and Q. Zhao, "A study on efficient generation of decision trees using genetic programming," in *Genetic and Evolutionary Computation Conference (GECCO'2000)*. Morgan Kaufmann, 2000, pp. 1047–1052.

[38] H. Zhao, "A multi-objective genetic programming approach to developing pareto optimal decision trees," *Decis. Support Syst.*, vol. 43, no. 3, pp. 809–826, 2007.

[39] A. Papagelis and D. Kalles, "Breeding decision trees using evolutionary techniques," in *Eighteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 393–400.

[40] K. Sorensen and G. K. Janssens, "Data mining with genetic algorithms on binary trees," *European Journal of Operational Research*, vol. 151, no. 2, pp. 253–264, December 2003. [Online]. Available: http://ideas.repec.org/a/eee/ejores/v151y2003i2p253-264.html

[41] Z. Fu and F. Mae, "A computational study of using genetic algorithms to develop intelligent decision trees," in *IEEE Congress on Evolutionary Computation*, vol. 2, 2001, pp. 1382 –1387.

[42] Z. Fu, B. L. Golden, S. Lele, S. Raghavan, and E. A. Wasil, "A genetic algorithm-based approach for building accurate decision trees," *INFORMS J. on Computing*, vol. 15, no. 1, pp. 3–22, 2003.

[43] Z. Fu, B. Golden, S. Lele, S. Raghavan, and E. Wasil, "Genetically engineered decision trees: Population diversity produces smarter trees," *Oper. Res.*, vol. 51, no. 6, pp. 894–907, 2003.

[44] Z. Fu, B. L. Golden, S. Lele, S. Raghavan, and E. Wasil, "Diversification for better classification trees," *Comput. Oper. Res.*, vol. 33, no. 11, pp. 3185–3202, 2006.

[45] M. Basgalupp, R. C. Barros, A. de Carvalho, A. Freitas, and D. Ruiz, "Legal-tree: a lexicographic multi-objective genetic algorithm for decision tree induction," in *ACM Symposium on Applied Computing (SAC '09)*. ACM Press, March 2009, pp. 1085–1090.

[46] R. C. Barros, M. P. Basgalupp, D. D. Ruiz, A. C. P. L. F. de Carvalho, and A. A. Freitas, "Evolutionary model tree induction," in *ACM Symposium on Applied Computing (SAC '10)*. New York, NY, USA: ACM, 2010, pp. 1131–1137.

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

[48] H. Kennedy, C. Chinniah, P. Bradbeer, and L. Morss, "The contruction and evaluation of decision trees: A comparison of evolutionary and concept learning methods," in *Evolutionary Computing*, ser. Lecture Notes in Computer Science, D. Corne and J. Shapiro, Eds. Springer Berlin / Heidelberg, 1997, vol. 1305, pp. 147–161, 10.1007/BFb0027172. [Online]. Available: http://dx.doi.org/10.1007/BFb0027172

[49] A.-A. Z. Bandar and H. McLean, "Genetic algorithm based multiple decision tree induction," in *6th International Conference on Neural Information Processing*, 1999, pp. 429–434.

[50] S. F. Smith, "Rna search acceleration with genetic algorithm generated decision trees," in *Seventh International Conference on Machine Learning and Applications*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 565–570.

[51] R. K. DeLisle and S. L. Dixon, "Induction of decision trees via evolutionary programming," *Journal of Chemical Information and Computer Sciences*, vol. 44, no. 3, pp. 862–870, 2004.

[52] R. Reynolds and H. Al-Shehri, "The use of cultural algorithms with evolutionary programming to guide decision tree induction in large databases," in *IEEE Congress on Evolutionary Computation*, 1998, pp. 541 –546.

[53] S.-H. Cha and C. Tappert, "A genetic algorithm for constructing compact binary decision trees," *Journal of Pattern Recognition Research*, vol. 1, pp. 1–13, 2009.

[54] M. Kretowski and M. Grzes, "Mixed decision trees: An evolutionary approach," in *International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2006)*, Krakow, Poland, 2006, pp. 260–269.

[55] X. Llora and J. M. Garrell, "Evolution of decision trees," in *4th Catalan Conference on Artificial Intelligence*. ACIA Press, 2001, pp. 115–122.

[56] J. Eggermont, J. Kok, and W. Kosters, "Genetic programming for data classification: Refining the search space," in *15th Belgium/Netherlands Conference on Artificial Intelligence (BNAIC'03)*, 2003, pp. 123–130.

[57] J. Eggermont, J. N. Kok, and W. A. Kosters, "Genetic programming for data classification: partitioning the search space," in *ACM symposium on Applied computing (SAC '04)*. New York, NY, USA: ACM, 2004, pp. 1001–1005.

[58] ——, "Detecting and pruning introns for faster decision tree evolution," in *Parallel Problem Solving from Nature - PPSN VIII*, ser. Lecture Notes in Computer Science, X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervos, J. A. Bullinaria, J. Rowe, P. Tino, A. Kaban, and H.-P. Schwefel, Eds. Springer Berlin / Heidelberg, 2004, vol. 3242, pp. 1071–1080.

[59] A. Garcia-Almanza and E. Tsang, "Simplifying decision trees learned by genetic programming," in *IEEE Congress on Evolutionary Computation (CEC 2006)*, 2006, pp. 2142 –2148.

[60] U. Johansson and L. Niklasson, "Evolving decision trees using oracle guides," in *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2009)*, 2009, pp. 238–244.

[61] G. Folino, C. Pizzuti, and G. Spezzano, "Genetic programming and simulated annealing: A hybrid method to evolve decision trees," in *European Conference on Genetic Programming*. London, UK: Springer-Verlag, 2000, pp. 294–303.

[62] ——, "Parallel genetic programming for decision tree induction," in *13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '01)*. Washington, DC, USA: IEEE Computer Society, 2001, p. 129.

[63] ——, "Improving induction decision trees with parallel genetic programming," in *Euromicro Conference on Parallel, Distributed, and Network-Based Processing*. Los Alamitos, CA, USA: IEEE Computer Society, 2002, pp. 1–7.

[64] D. Kim, "Structural risk minimization on decision trees using an evolutionary multiobjective optimization," in *Genetic Programming*, ser. Lecture Notes in Computer Science, M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, Eds. Springer Berlin / Heidelberg, 2004, vol. 3003, pp. 338–348.

[65] A. Papagelis and D. Kalles, "Ga tree: genetically evolved decision trees," in *IEEE International Conference on Tools with Artificial Intelligence*. Los Alamitos, CA, USA: IEEE Computer Society, 2000, pp. 203–206.

[66] S. Rouwhorst and A. Engelbrecht, "Searching the forest: using decision trees as building blocks for evolutionary search in classification databases," in *IEEE Congress on Evolutionary Computation*, 2000, pp. 633 –638.

[67] C. Y. Ma and X. Z. Wang, "Inductive data mining based on genetic programming:automatic generation of decision trees from data for process historical data analysis," *Computers and Chemical Engineering*, vol. 33, no. 10, pp. 1602 – 1616, 2009.

[68] D. Haizhou and M. Chong, "Study on constructing generalized decision tree by using dna coding genetic algorithm," in *International Conference on Web Information Systems and Mining (WISM '09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 163–167.

[69] M. Kretowski and M. Grzes, "Evolutionary induction of cost-sensitive decision trees," in *Foundations of Intelligent Systems*, ser. Lecture Notes in Computer Science, F. Esposito, Z. Ras, D. Malerba, and G. Semeraro, Eds. Springer Berlin / Heidelberg, 2006, vol. 4203, pp. 121–126.

[70] ——, "Evolutionary induction of decision trees for misclassification cost minimization," in *8th international conference on Adaptive and Natural Computing Algorithms (ICANNGA '07)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 1–10.

[71] ——, "Evolutionary induction of decision trees for misclassification cost minimization," in *Adaptive and Natural Computing Algorithms*, ser. Lecture Notes in Computer Science, B. Beliczynski, A. Dzielinski, M. Iwanowski, and B. Ribeiro, Eds. Springer Berlin / Heidelberg, 2007, vol. 4431, pp. 1–10.

[72] J. R. Koza, "Concept formation and decision tree induction using the genetic programming paradigm," in *1st Workshop on Parallel Problem Solving from Nature*. London, UK: Springer-Verlag, 1991, pp. 124–128.

[73] X. Llorà and S. W. Wilson, "Mixed decision trees: Minimizing knowledge representation bias in lcs," in *Genetic and Evolutionary Computation – GECCO 2004*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, vol. 3103, pp. 797–809.

[74] V. Podgorelec and P. Kokol, "Self-adapting evolutionary decision support model," in *IEEE International Symposium on ndustrial Electronics (ISIE '99)*, vol. 3, 1999, pp. 1484 –1489.

[75] A. Tsakonas and G. Dounias, "Hierarchical classification trees using type-constrained genetic programming," in *First International IEEE Symposium on Intelligent Systems*, 2002, pp. 50–54.

[76] J. Li, X. Li, and X. Yao, "Cost-sensitive classification with genetic programming," in *IEEE Congress on Evolutionary Computation*, vol. 3, 2005, pp. 2114 – 2121.

[77] V. Podgorelec and P. Kokol, "Evolutionary decision forests—decision making with multiple evolutionary constructed decision trees," in *Problems in Applied Mathematics and Computational Intelligence*, W. Press, Ed., 2001, pp. 97–103.

[78] ——, "Towards more optimal medical diagnosing with evolutionary algorithms," *Journal of Medical Systems*, vol. 25, pp. 195–219, 2001, 10.1023/A:1010733016906. [Online]. Available: http://dx.doi.org/10.1023/A:1010733016906

[79] M. Sprogar, P. Kokol, M. Zorman, V. Podgorelec, L. Lhotska, and J. Klema, "Evolving groups of basic decision trees," in *Fourteenth IEEE Symposium on Computer-Based Medical Systems*. Washington, DC, USA: IEEE Computer Society, 2001, p. 183.

[80] M. Zorman, V. Podgorelec, P. Kokol, M. Peterson, M. Sprogar, and M. Ojstersek, "Finding the right decision tree's induction strategy for a hard real world problem," *International Journal of Medical Informatics*, vol. 63, no. 1-2, pp. 109 – 121, 2001.

[81] Š. Hleb Babič, P. Kokol, V. Podgorelec, M. Zorman, M. Šprogar, and M. Molan Štiglic, "The art of building decision trees," *Journal of Medical Systems*, vol. 24, pp. 43–52, 2000.

[82] N. Nikolaev and V. Slavov, "Inductive genetic programming with decision trees," in *Machine Learning (ECML-97)*, ser. Lecture Notes in Computer Science, M. van Someren and G. Widmer, Eds. Springer Berlin / Heidelberg, 1997, vol. 1224, pp. 183–190.

[83] J. R. Quinlan, "MDL and Categorcal Theories (continued)," in *Twelfth International Conference in Machine Learning*. Morgan Kaufmann, 1995, pp. 464–470.

[84] U. M. Fayyad, "On the induction of decision trees for multiple concept learning," Ph.D. dissertation, Ann Arbor, MI, USA, 1992.

[85] A. Niimi and E. Tazaki, "Genetic programming combined with association rule algorithm for decision tree construction," in *4th International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, 2000, pp. 746 –749.

[86] G. Folino, C. Pizzuti, and G. Spezzano, "A cellular genetic programming approach to classification," in *Genetic and Evolutionary Computation Conference (GECCO99)*. Morgan Kaufmann, 1999, pp. 1015–1020.

[87] S. Oka and Q. Zhao, "Design of decision trees through integration of C4.5 and GP," in *4th Jpn.-Australia Joint Workshop Intell. Evol. Syst.*, 2000, pp. 128–135.

[88] S. Haruyama and Q. Zhao, "Designing smaller decision trees using multiple objective optimization based gps," in *IEEE International Conference on Systems, Man and Cybernetics*, oct. 2002, p. 5 pp.

[89] Z. Michalewicz and M. Schmidt, "Parameter control in practice," in *Parameter Setting in Evolutionary Algorithms*, ser. Studies in Computational Intelligence, F. Lobo, C. Lima, and Z. Michalewicz, Eds. Springer Berlin / Heidelberg, 2007, vol. 54, pp. 277–294.

[90] K. D. Jong and W. Spears, "An analysis of the interacting roles of population size and crossover in genetic algorithms," in *1st Workshop on Parallel Problem Solving from Nature*, London, UK, 1991, pp. 38–47.

[91] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, 1996.

[92] M. C. J. Bot and W. B. Langdon, "Application of genetic programming to induction of linear classification trees," in *Third European Conference on Genetic Programming*, 2000, pp. 247–258.

[93] D. J. Montana, "Strongly typed genetic programming," *Evol. Comput.*, vol. 3, no. 2, pp. 199–230, 1995.

[94] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press, 1992.

[95] J. Koza, "A genetic approach to the truck backer upper problem and the inter-twined spiral problem," in *International Joint Conference on Neural Networks (IJCNN 92)*, vol. 4, 1992, pp. 310 –318.

[96] W. Banzhaf, F. D. Francone, R. E. Keller, and P. Nordin, *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[97] M. C. J. Bot, D. Boelelaan, and W. B. Langdon, "Improving induction of linear classification trees with genetic programming," in *Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2000, pp. 403–410.

[98] C. Gathercole, "An investigation of supervised learning in genetic programming," Ph.D. dissertation, University of Edinburgh, 1998.

[99] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[100] M. Kretowski and M. Grzes, "Global induction of oblique decision trees: An evolutionary approach," in *Intelligent Information Processing and Web Mining*, ser. Advances in Soft Computing, M. Klopotek, S. Wierzchon, and K. Trojanowski, Eds. Springer Berlin / Heidelberg, 2005, vol. 31, pp. 309–318.

[101] L. Bobrowski, "Piecewise-linear classifiers, formal neurons and separability of the learning sets," in *13th International Conference on Pattern Recognition*, vol. 4, aug. 1996, pp. 224 –228.

[102] G. Fan and J. Gray, "Regression tree analysis using TARGET," *Journal of Computational and Graphical Statistics*, vol. 14, no. 1, pp. 206–218, 2005.

[103] J. Gray and G. Fan, "Classification tree analysis using TARGET," *Computational Statistics & Data Analysis*, vol. 52, pp. 1362–1372, 2008.

[104] G. Schwarz, "Estimating the dimension of a model," *Ann Stat*, vol. 6, pp. 461–464, 1978.

[105] M. Kretowski and M. Czajkowski, "An evolutionary algorithm for global induction of regression trees," in *Artifical Intelligence and Soft Computing*, ser. Lecture Notes in Computer Science, L. Rutkowski, R. Scherer, R. Tadeusiewicz, L. Zadeh, and J. Zurada, Eds. Springer Berlin / Heidelberg, 2010, vol. 6114, pp. 157–164.

[106] J. Quinlan, "Learning with Continuous Classes," in *5th Australian Joint Conference on Artificial Intelligence*, 1992, pp. 343–348.

[107] Y. Wang and I. H. Witten, "Induction of model trees for predicting continuous classes," in *Poster papers of the 9th European Conference on Machine Learning*. Springer, 1997.

[108] A. Hazan and R. Ramirez, "Modelling expressive performance using consistent evolutionary regression trees," in *17th European Conference on Artificial Intelligence (Workshop on Evolutionary Computation)*, Riva del Garda, Italy, August 2006.

[109] A. Hazan, R. Ramirez, E. Maestre, A. Perez, and A. Pertusa, "Modelling expressive performance: A regression tree approach based on strongly typed genetic programming," in *Applications of Evolutionary Computing*, ser. Lecture Notes in Computer Science, F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J. Moore, J. Romero, G. Smith, G. Squillero, and H. Takagi, Eds. Springer Berlin / Heidelberg, 2006, vol. 3907, pp. 676–687.

[110] C. Gagné and M. Parizeau, "Genericity in evolutionary computation software tools: Principles and case study," *International Journal on Artificial Intelligence Tools*, vol. 15, no. 2, pp. 173–194, Apr. 2006, 22 pages.

[111] G. Potgieter and A. Engelbrecht, "Evolving model trees for mining data sets with continuous-valued classes," *Expert Systems with Applications*, vol. 35, pp. 1513–1532, 2008.

[112] ——, "Genetic algorithms for the structural optimisation of learned polynomial expressions," *Appl Math Comput*, vol. 186, no. 2, pp. 1441–1466, 2007.

[113] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129 – 137, mar. 1982.

[114] K. Palaniappan, F. Zhu, X. Zhuang, Y. Zhao, and A. Blanchard, "Enhanced binary tree genetic algorithm for automatic land cover classification," in *IEEE International Geoscience and Remote Sensing Symposium (IGARSS 2000)*, vol. 2, 2000, pp. 688 –692.

[115] E. Cantu-Paz and C. Kamath, "Using evolutionary algorithms to induce oblique decision trees," in *Genetic and Evolutionary Computation Conference (GECCO-2000)*. Las Vegas, Nevada, USA: Morgan Kaufmann, 10-12 2000, pp. 1053–1060.

[116] ——, "Inducing oblique decision trees with evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 1, pp. 54 – 68, feb. 2003.

[117] S. K. Murthy, S. Kasif, S. Salzberg, and R. Beigel, "Oc1: A randomized induction of oblique decision trees," in *AAAI*, 1993, pp. 322–327.

[118] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller, "The gambler's ruin problem, genetic algorithms, and the sizing of populations," *Evol. Comput.*, vol. 7, no. 3, pp. 231–253, 1999.

[119] M. Kretowski, "An evolutionary algorithm for oblique decision tree induction," in *Artificial Intelligence and Soft Computing - ICAISC 2004*, ser. Lecture Notes in Computer Science, L. Rutkowski, J. Siekmann, R. Tadeusiewicz, and L. A. Zadeh, Eds. Springer Berlin / Heidelberg, 2004, vol. 3070, pp. 432–437.

[120] A. Shali, M. Kangavari, and B. Bina, "Using genetic programming for the induction of oblique decision trees," in *6th International Conference on Machine Learning and Applications (ICMLA 2007)*, dec. 2007, pp. 38 –43.

[121] M. Nú nez, "The use of background knowledge in decision tree induction," *Mach. Learn.*, vol. 6, no. 3, pp. 231–250, 1991.

[122] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I. H. Witten, "Using model trees for classification," *Mach. Learn.*, vol. 32, no. 1, pp. 63–76, 1998.

[123] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, October 1999.

[124] H. Chipman, E. George, and R. Mcculloch, "Bayesian CART model search," *J Am Stat Assoc*, vol. 93, pp. 935–960, 1997.

[125] D. Denison, B. Mallick, and A. Smith, "A Bayesian CART algorithm," *Biometrika*, vol. 85, pp. 363–377, 1998.

[126] M. Tan and J. C. Schlimmer, "Cost-sensitive concept learning of sensor use in approach and recognition," in *Sixth International Workshop on Machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 392–395.

[127] M. Tan, "Csl: a cost-sensitive learning system for sensing and grasping objects," in *IEEE International Conference on Robotics and Automation*, may. 1990, pp. 858 –863.

[128] ——, "Cost-sensitive learning of classification knowledge and its applications in robotics," *Mach. Learn.*, vol. 13, no. 1, pp. 7–33, 1993.

[129] S. W. Norton, "Generating better decision trees," in *11th international joint conference on Artificial intelligence (IJCAI'89)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 800–805.

[130] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[131] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[132] J. K. Martin and D. S. Hirschberg, "The time complexity of decision tree induction," University of California at Irvine, Tech. Rep., 1995.

[133] E. Cantu-Paz, C. Kamath, and R. Kamath, "Combining evolutionary algorithms with oblique decision trees to detect bent-double galaxies," in *Applications and science of neural networks, fuzzy systems, and evolutionary computation*, 2000, pp. 63–71.

[134] C.-S. Kuo, T.-P. Hong, and C.-L. Chen, "Applying genetic programming technique in classification trees," *Soft Comput.*, vol. 11, no. 12, pp. 1165–1172, 2007.

[135] V. Podforelec and P. Kokol, "Evolutionary construction of medical decision trees," in *20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 3, 1998, pp. 1202 –1205 vol.3.
[136] V. Podgorelec, M. Hericko, and I. Rozman, "Improving mining of medical data by outliers prediction," in *18th IEEE Symposium on Computer-Based Medical Systems*, 2005, pp. 91 – 96.
[137] J. K. Estrada-Gil, J. C. Fernandez-Lopez, E. Hernandez-Lemus, I. Silva-Zolezzi, A. Hidalgo-Miranda, G. Jimenez-Sanchez, and E. E. Vallejo-Clemente, "Gpdti: A genetic programming decision tree induction method to find epistatic effects in common complex diseases," *Bioinformatics*, vol. 23, no. 13, pp. 167–174, July 2007.
[138] C.-S. Kuo, T.-P. Hong, and C.-L. Chen, "An improved knowledge-acquisition strategy based on genetic programming," *Cybern. Syst.*, vol. 39, no. 7, pp. 672–685, 2008.
[139] E. V. Siegel, "Competitively evolving decision trees against fixed training cases for natural language processing," in *Advances in genetic programming*, K. E. Kinnear, Jr., Ed. Cambridge, MA, USA: MIT Press, 1994, pp. 409–423.
[140] T. M. Khoshgoftaar, Y. Liu, and N. Seliya, "Genetic programming-based decision trees for software quality classification," in *IEEE International Conference on Tools with Artificial Intelligence*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2003, p. 374.
[141] T. Khoshgoftaar and Y. Liu, "A multi-objective software quality classification model using genetic programming," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 237 –245, jun. 2007.
[142] M. Chis, "Evolutionary decision trees and software metrics for module defects identification," in *World Academy of Science, Engineering and Technology (WASET)*, vol. 2, 2008, pp. 273–277.
[143] P. Kokol, V. Podgorelec, and M. Pighin, "Using software metrics and evolutionary decision trees for software quality control," in *European Software Control and Metrics (ESCOM '01)*, 2001, pp. 181–191.
[144] D. Kalles and A. Papagelis, "Lossless fitness inheritance in genetic algorithms for decision trees," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 14, pp. 973–993, 2010.
[145] V. Slavov and N. I. Nokolaev, "Fitness landscapes and inductive ge- netic programming," in *3rd Int. Conf. Artif. Neural Nets Genet. Algorithms*. Berlin, Germany: Springer-Verlag, 1997, pp. 414–418.
[146] T. K. Ho and M. Basu, "Complexity measures of supervised classification problems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 3, pp. 289–300, 2002.
[147] T. Ho, M. Basu, and M. Law, "Measures of geometrical complexity in classification problems," in *Data Complexity in Pattern Recognition*, ser. Advanced Information and Knowledge Processing, L. Jain, X. Wu, M. Basu, and T. K. Ho, Eds. Springer London, 2006, pp. 1–23.

Rodrigo Coelho Barros obtained his BSc in Computer Science from UFPel-RS, Brazil, in 2007; his MSc in Computer Science from PUC-RS, Brazil, in 2009; and he is currently a PhD student in Computer Science at University of São Paulo where he works with machine learning and data mining topics. He has published papers in peer-reviewed journals and conferences. His current research interests are machine learning, data mining, knowledge discovery and biologically-inspired computational intelligence algorithms.

Marcio Porto Basgalupp is an associate professor at UNIFESP, São Paulo, Brazil. He obtained his BSc in Computer Science from UFPel-RS, Brazil, in 2005; his MSc in Computer Science from PUC-RS, Brazil, in 2007; his PhD in Computer Science from University of São Paulo, Brazil, in 2010. In 2010, he held a post-doc position at NTNU, Trondheim, Norway, where he worked with bio-medical data mining. He has published papers in peer-reviewed journals and conferences. His current research interests are machine learning, data mining and bio-inspired computation.

Andre C. P. L. F. de Carvalho received his B.Sc. and M.Sc. degrees in Computer Science from the Universidade Federal de Pernambuco, Brazil. He received his Ph.D. degree in Electronic Engineering from the University of Kent, UK. Prof. André de Carvalho is Full Professor at the Department of Computer Science, Universidade de São Paulo, Brazil. He has published around 60 Journal and 200 Conference refereed papers. He has been involved in the organization of several conferences and journal special issues. His main interests are Machine Learning, Data Mining, Bioinformatics, Evolutionary Computation, Bioinspired Computing and Hybrid Intelligent Systems.

Alex A. Freitas obtained his BSc in Computer Science from FATEC-SP , Brazil, in 1989; his MSc in Computer Science from UFSCar, Brazil, in 1993; and his PhD in Computer Science from the University of Essex, UK, in 1997. He is a Reader (position equivalent to Associate Professor) at the School of Computing, University of Kent, UK. He is a member of the editorial board of four international journals, has (co)-authored three research-oriented books on data mining, and has published more than 130 refereed papers in journals and conference proceedings. His current research interests are data mining and knowledge discovery, biologically-inspired algorithms, bioinformatics and the biology of ageing.