A Hyper-Heuristic Evolutionary Algorithm for Automatically Designing Decision-Tree Algorithms

Rodrigo C. Barros
ICMC-USP
University of São Paulo
São Carlos, SP, Brazil
rcbarros@icmc.usp.br

Márcio P. Basgalupp
ICT-UNIFESP
Federal University of SP
S. J. dos Campos, SP, Brazil
basgalupp@unifesp.br

André C. P. L. F. de Carvalho
ICMC-USP
University of São Paulo
São Carlos, SP, Brazil
andre@icmc.usp.br

Alex A. Freitas
School of Computing
University of Kent
Canterbury, Kent, UK
A.A.Freitas@kent.ac.uk

ABSTRACT

Decision tree induction is one of the most employed methods to extract knowledge from data, since the representation of knowledge is very intuitive and easily understandable by humans. The most successful strategy for inducing decision trees, the greedy top-down approach, has been continuously improved by researchers over the years. This work, following recent breakthroughs in the automatic design of machine learning algorithms, proposes a hyper-heuristic evolutionary algorithm for automatically generating decision-tree induction algorithms, named HEAD-DT. We perform extensive experiments in 20 public data sets to assess the performance of HEAD-DT, and we compare it to the traditional decision-tree algorithms C4.5 and CART. Results show that HEAD-DT can generate algorithms that significantly outperform C4.5 and CART regarding predictive accuracy and F-Measure.

Categories and Subject Descriptors

I.2.6 [**Induction and Knowledge Acquisition**]: Learning

General Terms

Algorithms

Keywords

Decision trees, hyper-heuristics, evolutionary algorithms

1. INTRODUCTION

Top-down induction of decision trees is a powerful classification method in machine learning and data mining, due to its comprehensible nature that resembles the human reasoning. Decision-tree (DT) induction algorithms present several advantages over other learning algorithms, such as robustness to noise, low computational cost and ability to deal with redundant attributes.

Several attempts on optimizing DT algorithms have been made by researches within the last decades, even though the most successful algorithms date back to mid-80's [5] and early 90's [32]. Different strategies were employed for deriving accurate DTs, such as bottom-up induction [2], hybrid induction [22], evolutionary induction [1, 3] and ensemble of trees [4], just to name a few. Nevertheless, no strategy has been more successful in generating accurate and comprehensible decision trees with low computational effort than the greedy top-down induction strategy.

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

A great number of approaches were proposed in the literature for each one of these *design components* of top-down DT algorithms. For instance, new measures for node splitting tailored for a vast number of application domains were proposed, as well as many different strategies for selecting multiple attributes for composing the node rule. There are even works in the literature that survey the numerous approaches for pruning a decision tree [6, 16]. It is clear that by improving these design components, we can obtain more robust top-down DT algorithms.

The *manual* improvement of DT design components has been carried out for the past 40 years. Though effective, we believe that *automatically* designing DT algorithms could provide a faster, less-tedious - and equally effective - strategy for improving DT algorithms in the next years. Hence, we propose a hyper-heuristic evolutionary algorithm for automatically design new (and effective) DT algorithms.

GECCO'12, July 7–11, 2012, Philadelphia, Pennsylvania, USA.

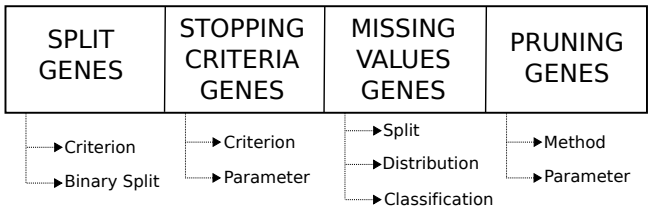

Figure 1: Linear-genome for evolving decision-tree algorithms.

We name our method HEAD-DT, short for Hyper-heuristic Evolutionary Algorithm for Designing DT algorithms. We detail HEAD-DT in Section 2, and perform a thorough experimental analysis in Section 3, where we compare the algorithms evolved by HEAD-DT with C4.5 [32] and CART [5]. We discuss the results in Secion 4 and we comment on related work in Section 5. Finally, we present our conclusions and future work directions in Section 6.

2. HEAD-DT

HEAD-DT is a hyper-heuristic algorithm to automatically design top-down DT algorithms. Hyper-heuristics can automatically generate new heuristics suited to a given problem or class of problems. This is done by combining, through an evolutionary algorithm, components or building-blocks of human designed heuristics [7]. In HEAD-DT, individuals are collections of building blocks of DT algorithms. Each individual is encoded as an integer string (see Figure 1), and each gene has a different range of supported values. We divided the genes in four categories that represent the major building blocks (design components) of a DT algorithm: (i) split genes; (ii) stopping criteria genes; (iii) missing values genes; and (iv) pruning genes. We detail each category next.

2.1 Split Genes

These genes are concerned with the task of selecting the attribute to split the data in the current node of the decision tree. A decision rule based on the selected attribute is thus generated, and the input data is filtered according to the outcomes of this rule, and the process continues recursively.

We have used two genes to model the split component of a DT algorithm. The first gene, **criterion**, is an integer that indexes one of the 15 splitting criteria we have implemented: information gain [29], Gini index [5], global mutual information [19], G statistics [26], Mantaras criterion [13], hypergeometric distribution [25], Chandra-Varghese criterion [10], DCSM [9], χ^2 [27], mean posterior improvement [33], normalized gain [21], orthogonal criterion [17], twoing [5], CAIR [11] and gain ratio [32].

The most well-known univariate criteria are based on Shannon's entropy. Entropy is known to be a unique function which satisfies the four axioms of uncertainty. It represents the average amount of information when coding each class into a codeword with ideal length according to its probability. The first splitting criterion that arose based on entropy is the *global mutual information* (GMI) [19]. Following, the well-known *information gain* [29] became a standard after appearing in algorithms such as ID3 [29] and Assistant [23]. It belongs to the class of the so-called *impurity-based criteria*. Quinlan [29] acknowledges the fact that the information gain is biased towards attributes with many values. He proposes a solution for this matter called *gain ratio* [32]. It basically consists of normalizing the information gain by the entropy of the attribute being tested. Several variations of the gain ratio have been proposed, e.g., the normalized gain [21]. Alternatives to entropy-based criteria are the class of distance-based measures, i.e., criteria that evaluate separability, divergency or discrimination between classes. Examples are the Gini index [5], the twoing criterion [5], the orthogonality criterion [17], among others. We have also included lesser-known criteria such as CAIR [11] and mean posterior improvement [33], as well as the more recent Chandra-Varghese [10] and DCSCM [9], to enhance the diversity of options for generating splits in a decision tree.

The second gene used to model the split component of a DT algorithm is **binary split**, which is a binary gene that indicates whether the splits of a DT are going to be necessarily binary or perhaps multi-way. In a binary tree, every split has only two outcomes, which means that nominal attributes with many categories will be divided in two subsets, each representing an aggregation over several categories. In a multi-way tree, nominal attributes are divided according to their number of categories, i.e., one edge (outcome) for each category. In both cases, numeric attributes always partition the tree in two subsets, ($\leq$ *threshold* and $>$ *threshold*).

2.2 Stopping Criteria Genes

The second category of genes is related to the stopping criteria component of DT algorithms. The top-down induction of a DT is recursive and it continues until a stopping criterion (also known as *pre-pruning*) is satisfied. We have implemented the following stopping criteria:

- Reaching class homogeneity: when all instances that reach a given node belong to the same class, there is no reason to split this node any further. This strategy can be combined with any of the following strategies;
- Reaching the maximum tree depth: a parameter *tree depth* can be specified to avoid deep trees. We have fixed its range in the interval $[2, 10]$ levels;
- Reaching the minimum number of instances for a non-terminal node: a parameter *minimum number of instances for a non-terminal node* can be specified to avoid (or at least alleviate) the data fragmentation problem in DTs. Range: $[1, 20]$ instances;
- Reaching the minimum percentage of instances for a non-terminal node: same as above, but instead of the actual number of instances, we set the minimum percentage of instances. The parameter is thus relative to the total number of instances in a data set. Range: $[1\%, 10\%]$ the total number of instances ;
- Reaching an accuracy threshold within a node: a parameter *accuracy reached* can be specified for halting the growth of the tree when the accuracy within a node (majority of instances) has reached a given threshold. Range: $\{70\%, 75\%, 80\%, 85\%, 90\%, 95\%, 99\%\}$.

The first gene, **criterion**, selects among the five different strategies for stopping the tree growth. The second gene,

parameter, dynamically adjusts a value in the range $[0, 100]$ to the corresponding strategy. For example, if the strategy selected by gene **criterion** is *reaching an accuracy threshold within a node*, the following mapping function is executed:

$$param = ((value \bmod 7) \times 5) + 70 \quad (1)$$

This function maps from $[0, 100]$ to $\{70, 75, 80, 85, 90, 95, 100\}$, which is almost what was defined as the range of strategy *reaching an accuracy threshold*. The final step subtracts 1 if the resulting value of the parameter is 100. Similar mapping functions are executed dynamically to adjust the ranges of gene **parameter**.

2.3 Missing Values Genes

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

During the split criterion evaluation in node t based on attribute a_i, we implemented the strategies: (i) ignore all instances whose value of a_i is missing [5]; (ii) imputation of missing values with either the mode (nominal attributes) or the mean/median (numeric attributes) of all instances in t [12]; (iii) weight the splitting criterion value (calculated in node t with regard to a_i) by the proportion of missing values [31]; and (iv) imputation of missing values with either the mode (nominal attributes) or the mean/median (numeric attributes) of all instances in t whose class attribute is the same of the instance whose a_i value is being imputed [24].

For deciding which child node training instance x_j should go to, considering a split in node t over a_i, we adopted the options: (i) ignore instance x_j [29]; (ii) treat instance x_j as if it has the most common value of a_i (mode or mean), regardless of the class [31]; (iii) treat instance x_j as if it has the most common value of a_i (mode or mean) considering the instances that belong to the same class than x_j; (iv) assign instance x_j to all partitions; (v) assign instance x_j to the partition with largest number of instances [31]; (vi) weight instance x_j according to the partition probability [32, 23]; and (vii) assign instance x_j to the most probable partition, considering the class of x_j [24].

Finally, for classifying an unseen test instance x_j, considering a split in node t over a_i, we used the strategies: (i) explore all branches of t combining the results [30]; (ii) take the route to the most probable partition (largest subset); (iii) halt the classification process and assign instance x_j to the majority class of node t [31].

2.4 Pruning Genes

Pruning is usually performed in decision trees for enhancing tree comprehensibility by reducing its size while maintaining (or even improving) accuracy. It was originally conceived as a strategy for tolerating noisy data, though it was found to improve decision tree accuracy in many noisy data sets [5, 29, 30]. We have implemented the following well-known pruning strategies: i) reduced-error pruning; ii) pessimistic error pruning; iii) minimum error pruning; iv) cost-complexity pruning; and v) error-based pruning.

Reduced-error pruning (REP) is a conceptually simple strategy proposed by Quinlan [30]. It uses a pruning set (a part of the training set) to evaluate the goodness of a given subtree from T. The idea is to evaluate each non-terminal node t with regard to the classification error in the pruning set. If such an error decreases when we replace the subtree $T^{(t)}$ rooted on t by a leaf node, then $T^{(t)}$ must be pruned. Quinlan imposes a constraint: a node t cannot be pruned if it contains a subtree that yields a lower classification error in the pruning set. The practical consequence of this constraint is that REP should be performed in a bottom-up fashion. The REP pruned tree T' presents an interesting optimality property: it is the smallest most accurate tree resulting from pruning original tree T [30]. Besides this optimality property, another advantage of REP is its linear complexity, since each node is visited only once in T. An obvious disadvantage is the need of using a pruning set, which means one has to divide the original training set, resulting in fewer instances to grow the tree. This disadvantage is particularly serious for small data sets.

Also proposed by Quinlan [30], the pessimistic error pruning (PEP) uses the training set for both growing and pruning the tree. The apparent error rate, i.e., the error rate calculated over the training set, is optimistically biased and cannot be used to decide whether pruning should be performed or not. Quinlan thus proposes adjusting the apparent error according to the continuity correction for the binomial distribution in order to provide a more realistic error rate. PEP is computed in a top-down fashion, and if a given node t is pruned, its descendants are not examined, which makes this pruning strategy quite efficient in terms of computational effort. As a point of criticism, Esposito et al. [16] point out that the introduction of the continuity correction in the estimation of the error rate has no theoretical justification, since it was never applied to correct over-optimistic estimates of error rates in statistics.

Originally proposed in [28] and further extended in [8], minimum error pruning (MEP) is a bottom-up approach that seeks to minimize the *expected error rate* for unseen cases. It uses an ad-hoc parameter m for controlling the level of pruning. Usually, the higher the value of m, the more severe the pruning. Cestnik and Bratko [8] suggest that a domain expert should set m according to the level of noise in the data. Alternatively, a set of trees pruned with different values of m could be offered to the domain expert, so he/she can choose the best one according to his/her experience.

Cost-complexity pruning (CCP) is the post-pruning strategy of the CART system, detailed in [5]. It consists of two steps: (i) generate a sequence of increasingly smaller trees, beginning with T and ending with the root node of T, by successively pruning the subtree yielding the lowest *cost complexity*, in a bottom-up fashion; (ii) choose the best tree among the sequence based on its relative size and accuracy (either on a pruning set, or provided by a cross-validation procedure in the training set). The idea within step 1 is that pruned tree T_{i+1} is obtained by pruning the subtrees that show the lowest increase in the apparent error (error in the training set) per pruned leaf. Regarding step 2, CCP chooses the smallest tree whose error (either on the pruning set or on cross-validation) is not more than one standard error (SE) greater than the lowest error observed in the sequence of trees.

Finally, error-based pruning (EBP) was proposed by

Table 1: Summary of the data sets used in the experiments.

data set	# instances	# attributes	# numeric attributes	# nominal attributes	% missing	# classes
abalone	4177	8	7	1	0.00	30
anneal	898	38	6	32	0.00	6
arrhythmia	452	279	206	73	0.32	16
audiology	226	69	0	69	2.03	24
bridges_version1	107	12	3	9	5.53	6
car	1728	6	0	6	0.00	4
cylinder_bands	540	39	18	21	4.74	2
glass	214	9	9	0	0.00	7
hepatitis	155	19	6	13	5.67	2
iris	150	4	4	0	0.00	3
kdd_synthetic	600	61	60	1	0.00	6
segment	2310	19	19	0	0.00	7
semeion	1593	265	265	0	0.00	2
shuttle_landing	15	6	0	6	28.89	2
sick	3772	30	6	22	5.54	2
tempdiag	120	7	1	6	0.00	2
tep.fea	3572	7	7	0	0.00	3
vowel	990	13	10	3	0.00	11
winequality_red	1599	11	11	0	0.00	10
winequality_white	4898	11	11	0	0.00	10

Quinlan and it is implemented as the default pruning strategy of C4.5 [32]. It is an improvement over PEP, based on a far more pessimistic estimate of the expected error. Unlike PEP, EBP performs a bottom-up search, and it carries out not only the replacement of non-terminal nodes by leaves but also *grafting* of subtree $T^{(t)}$ onto the place of parent t. For deciding whether to replace a non-terminal node by a leaf (subtree replacement), to graft a subtree onto the place of its parent (subtree raising) or not to prune at all, a pessimistic estimate of the expected error is calculated by using an upper confidence bound. An advantage of EBP is the new *grafting* operation that allows pruning useless branches without ignoring interesting lower branches. A drawback of the method is the presence of an ad-hoc parameter, CF. Smaller values of CF result in more pruning.

We designed two genes in HEAD-DT for pruning. The first gene, **method**, indexes one of the five approaches for pruning a DT (and also the option of not pruning at all). The second gene, **parameter**, is in the range $[0, 100]$ and its value is dynamically mapped by a function, according to the pruning method selected (similar to the stopping criteria **parameter** gene). For REP, the parameter is the percentage of training data to be used in the pruning set (varying within the interval $[10\%, 50\%]$). For PEP, the parameter is the number of standard errors (SEs) to adjust the apparent error, in the set $\{0.5, 1, 1.5, 2\}$. For MEP, the parameter m may range within $[0, 100]$. For CCP, there are two parameters: the number of SEs (in the same range than PEP) and the pruning set size (in the same range than REP). Finally, for EBP, the parameter CF may vary within $[1\%, 50\%]$.

2.5 Evolutionary Process

HEAD-DT is a regular generational EA configured with the following parameters:

- Population size: 100;
- Maximum number of generations: 100;
- Selection: tournament selection with size $t = 2$;
- Elitism rate: 10 individuals;
- Crossover: one-point crossover with 90% probability;
- Mutation: random uniform gene mutation with 10% probability.

The initial population is created by randomly choosing values within the range of each gene. For guiding the search for the near-optimal solution, we have defined the fitness function of HEAD-DT to be the accuracy of the decision tree generated by the evolved algorithms. In order to avoid overfitting issues, we have divided the training data set in two subsets: sub-training (75%) and validation (25%) sets. The sub-training set is used so the evolved algorithm can generate a decision-tree at each generation of the EA. The validation set is used to assess the relative performance of the algorithm in the training data, and hence the accuracy obtained in the validation set is used as fitness function for guiding the evolutionary search. At the end of each run, the best individual (algorithm) is selected for inducing a decision tree from the test data set.

3. EXPERIMENTAL ANALYSIS

3.1 Methodology

To assess the relative performance of the algorithms automatically designed by HEAD-DT, we use different data sets (see Table 1) from the UCI machine-learning repository[1] [18] and compare the resulting DT algorithms with the two most well-known and widely-used DT induction algorithms: C4.5 [32] and CART [5]. We report the classification accuracy of the DTs generated for each data set, as well as the F-Measure (performance measure which is not biased towards the majority class) and the size of the decision tree (total number of nodes). All results are based on the average of 10-fold cross-validation runs. Additionally, since HEAD-DT is a non-deterministic method, we averaged the results of 5 different runs (varying the random seed).

In order to provide some reassurance about the validity and non-randomness of the obtained results, we present the results of statistical tests by following the approach proposed by Demšar [15]. In brief, this approach seeks to compare

[1] `http://archive.ics.uci.edu/ml/`

multiple algorithms on multiple data sets, and it is based on the use of the Friedman test with a corresponding post-hoc test. The Friedman test is a non-parametric counterpart of the well-known ANOVA, as follows. Let R_i^j be the rank of the j^{th} of k algorithms on the i^{th} of N data sets. The Friedman test compares the average ranks of algorithms, $R_j = \frac{1}{N}\sum_i R_i^j$. The Friedman statistic is given by:

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \quad (2)$$

is distributed according to χ_F^2 with $k-1$ degrees of freedom, when N and k are big enough.

Iman and Davenport [20] showed that Friedman's χ_F^2 is undesirably conservative and derived an adjusted statistic:

$$F_f = \frac{(N-1)\times\chi_F^2}{N\times(k-1)-\chi_F^2} \quad (3)$$

which is distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

If the null hypothesis of similar performances is rejected, then we proceed with the Nemenyi post-hoc test for pairwise comparisons. The performance of two classifiers is significantly different if their corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha\sqrt{\frac{k(k+1)}{6N}} \quad (4)$$

where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$.

3.2 Results

Table 2 shows the classification accuracy of C4.5, CART and HEAD-DT. It illustrates the average accuracy over the 10-fold cross-validation runs ± the standard deviation of the accuracy obtained in those runs (best absolute values in bold). It is possible to see that HEAD-DT generates more accurate trees in 13 out of the 20 data sets. CART provides more accurate trees in two data sets, and C4.5 in none. In the remaining 5 data sets, no method was superior to the others.

To evaluate the statistical significance of the accuracy results, we calculated the average rank for CART, C4.5 and HEAD-DT: 2.375, 2.2 and 1.425, respectively. The average rank suggest that HEAD-DT is the best performing method regarding accuracy. The calculation of Friedman's χ_F^2 is given by:

$$\chi_F^2 = \frac{12\times20}{3\times4}\left[2.375^2+2.2^2+1.425^2-\frac{3\times4^2}{4}\right] = 10.225 \quad (5)$$

Iman's F statistic is given by:

$$F_f = \frac{(20-1)\times10.225}{20\times(3-1)-10.22} = 6.52 \quad (6)$$

Critical value of $F(k-1,(k-1)(n-1)) = F(2,38)$ for $\alpha = 0.05$ is 3.25. Since $F_f > F_{0.05}(2,38)$ $(6.52 > 3.25)$, the null-hypothesis is rejected. We proceed with a post-hoc Nemenyi test to find which method provides better results. The critical difference CD is given by:

Table 2: Classification accuracy of CART, C4.5 and HEAD-DT.

	CART	C4.5	HEAD-DT
abalone	**0.26 ± 0.02**	0.22 ± 0.02	0.20 ± 0.02
anneal	0.98 ± 0.01	0.99 ± 0.01	0.99 ± 0.01
arrhythmia	**0.71 ± 0.05**	0.65 ± 0.04	0.65 ± 0.04
audiology	0.74 ± 0.05	0.78 ± 0.07	**0.80 ± 0.06**
bridges_version1	0.41 ± 0.07	0.57 ± 0.10	**0.60 ± 0.12**
car	0.97 ± 0.02	0.93 ± 0.02	**0.98 ± 0.01**
cylinder_bands	0.60 ± 0.05	0.58 ± 0.01	**0.72 ± 0.04**
glass	0.70 ± 0.11	0.69 ± 0.04	**0.73 ± 0.10**
hepatitis	0.79 ± 0.05	0.79 ± 0.06	**0.81 ± 0.08**
iris	0.93 ± 0.05	0.94 ± 0.07	**0.95 ± 0.04**
kdd_synthetic	0.88 ± 0.00	0.91 ± 0.04	**0.97 ± 0.03**
segment	0.96 ± 0.01	0.97 ± 0.01	0.97 ± 0.01
semeion	0.94 ± 0.01	0.95 ± 0.02	**1.00 ± 0.00**
shuttle_landing	0.95 ± 0.16	0.95 ± 0.16	**0.95 ± 0.15**
sick	0.99 ± 0.01	0.99 ± 0.00	0.99 ± 0.00
tempdiag	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
tep.fea	0.65 ± 0.02	0.65 ± 0.02	0.65 ± 0.02
vowel	0.82 ± 0.04	0.83 ± 0.03	**0.89 ± 0.04**
winequality_red	0.63 ± 0.02	0.61 ± 0.03	**0.64 ± 0.03**
winequality_white	0.58 ± 0.02	0.61 ± 0.03	**0.63 ± 0.03**

$$CD = 2.343\times\sqrt{\frac{3\times4}{6\times20}} = 0.74 \quad (7)$$

The difference between the average rank of HEAD-DT and C4.5 is 0.775 and that between HEAD-DT and CART is 0.95. Since both the differences are greater than CD, the performance of HEAD-DT is significantly better than both C4.5 and CART regarding accuracy.

Table 3 shows the classification F-Measure of C4.5, CART and HEAD-DT. Results show that HEAD-DT generates better trees (regardless of the class imbalance problem) in 16 out of the 20 data sets. CART generates the best tree in two data sets, while C4.5 does not provide the best tree for any data set.

Table 3: Classification F-Measure of CART, C4.5 and HEAD-DT.

	CART	C4.5	HEAD-DT
abalone	**0.22 ± 0.02**	0.21 ± 0.02	0.20 ± 0.02
anneal	0.98 ± 0.01	0.98 ± 0.01	**0.99 ± 0.01**
arrhythmia	**0.67 ± 0.05**	0.64 ± 0.05	0.63 ± 0.06
audiology	0.70 ± 0.04	0.75 ± 0.08	**0.79 ± 0.07**
bridges_version1	0.44 ± 0.06	0.52 ± 0.10	**0.56 ± 0.12**
car	0.93 ± 0.97	0.93 ± 0.02	**0.98 ± 0.01**
cylinder_bands	0.54 ± 0.07	0.42 ± 0.00	**0.72 ± 0.04**
glass	0.67 ± 0.10	0.67 ± 0.05	**0.72 ± 0.09**
hepatitis	0.74 ± 0.07	0.77 ± 0.06	**0.80 ± 0.08**
iris	0.93 ± 0.05	0.93 ± 0.06	**0.95 ± 0.05**
kdd_synthetic	0.88 ± 0.03	0.90 ± 0.04	**0.97 ± 0.03**
segment	0.95 ± 0.01	0.96 ± 0.09	**0.97 ± 0.01**
semeion	0.93 ± 0.01	0.95 ± 0.02	**1.00 ± 0.00**
shuttle_landing	0.56 ± 0.03	0.56 ± 0.38	**0.93 ± 0.20**
sick	0.98 ± 0.00	0.98 ± 0.00	**0.99 ± 0.00**
tempdiag	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
tep.fea	0.60 ± 0.02	0.61 ± 0.02	0.61 ± 0.02
vowel	0.81 ± 0.03	0.82 ± 0.03	**0.89 ± 0.03**
winequality_red	0.61 ± 0.02	0.60 ± 0.03	**0.63 ± 0.03**
winequality_white	0.57 ± 0.02	0.60 ± 0.02	**0.63 ± 0.03**

We calculated the average rank for CART, C4.5 and HEAD-DT: 2.5, 2.225 and 1.275, respectively. The average rank suggest that HEAD-DT is the best performing method regarding F-Measure. The calculation of Friedman's χ_F^2 is given by:

$$\chi_F^2 = \frac{12 \times 20}{3 \times 4}\left[2.5^2 + 2.225^2 + 1.275^2 - \frac{3 \times 4^2}{4}\right] = 16.525 \quad (8)$$

Iman's F statistic is given by:

$$F_f = \frac{(20-1) \times 16.525}{20 \times (3-1) - 16.525} = 13.375 \quad (9)$$

Since $F_f > F_{0.05}(2, 38)$ $(13.375 > 3.25)$, the null-hypothesis is rejected. The difference between the average rank of HEAD-DT and C4.5 is 0.95 and that between HEAD-DT and CART is 1.225. Since both the differences are greater than CD (0.74), the performance of HEAD-DT is significantly better than both C4.5 and CART regarding F-Measure.

Table 4 shows the size of trees generated by C4.5, CART and HEAD-DT. Results show that CART generates smaller trees in 15 out of the 20 data sets. C4.5 generates smaller trees in 2 data sets, and HEAD-DT in only one data set. The statistical analysis is given below:

$$\chi_F^2 = \frac{12 \times 20}{3 \times 4}\left[1.2^2 + 2^2 + 2.8^2 - \frac{3 \times 4^2}{4}\right] = 25.6 \quad (10)$$

$$F_f = \frac{(20-1) \times 33.78}{20 \times (3-1) - 25.6} = 33.78 \quad (11)$$

Since $F_f > F_{0.05}(2, 38)$ $(33.78 > 3.25)$, the null-hypothesis is rejected. The difference between the average rank of HEAD-DT and C4.5 is 0.8 and that between HEAD-DT and CART is 1.6. Since both the differences are greater than CD (0.74), HEAD-DT generates trees which are significantly larger than both C4.5 and CART. This should not be a concern, since smaller trees are only preferable in scenarios where the predictive performance of the algorithms is similar. The statistical analysis previously presented clearly indicate that HEAD-DT generates algorithms whose trees outperform C4.5 and CART regarding predictive performance. The Occam's razor assumption that among competitive hypotheses, the simpler is preferred, does not apply in this case.

4. DISCUSSION

In this section, we discuss three important topics to support the empirical analysis presented in Section 3: (i) the accuracy dilemma and when preferring F-Measure for evaluating DT algorithms; (ii) the empirical (and theoretic) time complexity of HEAD-DT, and also of the algorithms it generates; and (iii) an example of algorithm automatically designed by HEAD-DT.

4.1 Accuracy vs F-Measure

The fact that HEAD-DT designs algorithms whose trees are significantly more accurate than those by C4.5 and CART is quite encouraging. Notwithstanding, we must point out that accuracy may be a misleading performance measure. For instance, suppose we have a data set whose class distribution is very skewed: 90% of the instances belong to class A and 10% to class B. An algorithm that always classifies instances as belonging to class A would achieve 90% of accuracy, even though it never predicts a class-B instance. In this case, assuming that class B is equally important (or even more so) than class A, we would prefer an algorithm with lower accuracy, but that could eventually predict instances as belonging to the rare class B.

The example above illustrates the importance of not relying only in accuracy when designing an algorithm. F-Measure is the harmonic mean of *precision* and *recall*, and thus rewards solutions that present a good trade-off between these two measures. Hence, for imbalanced-class problems, F-Measure should be preferred over accuracy.

Even though HEAD-DT explicitly optimizes its solutions to achieve the highest possible accuracy, we can notice that HEAD-DT algorithms provide decision trees whose F-Measure are significantly higher than C4.5 and CART. In other words, HEAD-DT usually generates DT algorithms which are more robust to skewed-class problems, and that is yet another advantage of the automatic algorithm's generation by HEAD-DT.

4.2 Time Complexity

Regarding execution time, it is clear that HEAD-DT is slower than either C4.5 and CART. Considering that there are 100 individuals executed for 100 generations, there is a maximum (worst case) of 10000 fitness evaluations of decision trees.

We recorded the execution time of both breeding operations and fitness evaluation (one thread was used for breeding and other for evaluation). Total time of breeding is absolutely negligible (a few milliseconds in a full evolutionary cycle), regardless of the data set being used (breeding does not consider any domain-specific information). Indeed, breeding individuals in the form of an integer string is known to be quite efficient in the EA research field.

Fitness evaluation, on the other hand, is the bottleneck of HEAD-DT. The largest data set (winequality_white) takes 2.5 hours to be fully executed (one full evolutionary cycle of 100 generations). The smallest data set (shuttle_landing) takes only 0.72 seconds to be fully executed, which means the fitness evaluation time can vary quite a lot according to the data set size.

The computational complexity of algorithms such as C4.5 and CART is $O(m \times n \log n)$ (m is the number of attributes and n the data set size), plus a term regarding the specific pruning method. Considering that breeding takes negligible time, we can say that in the worst case scenario, HEAD-DT time complexity is $O(i \times g \times m \times n \log n)$, where i is the number of individuals and g is the number of generations. In practice, the number of evaluations is much smaller than $i \times g$, due to the fact that repeated individuals are not re-evaluated. In addition, individuals selected by elitism and by reproduction (instead of crossover) are also not re-evaluated, saving computational time.

4.3 Example of an Evolved Algorithm

For illustrating a novel algorithm designed by HEAD-DT, let us consider the Semeion data set, in which HEAD-DT managed to achieve maximum accuracy and F-Measure (which was not the case of CART and C4.5). The algorithm designed by HEAD-DT is presented in Algorithm 1. It is indeed novel, since no algorithm in the literature combines components such as the Chandra-Varghese criterion with MEP pruning. Furthermore, it chooses MEP as its pruning

Table 4: Tree size of CART, C4.5 and HEAD-DT trees.

	CART	C4.5	HEAD-DT
abalone	**44.40 ± 16.00**	2088.90 ± 37.63	4068.12 ± 13.90
anneal	**21.00 ± 3.13**	48.30 ± 6.48	55.72 ± 3.66
arrhythmia	**23.20 ± 2.90**	82.60 ± 5.80	171.84 ± 5.18
audiology	**35.80 ± 11.75**	50.40 ± 4.01	118.60 ± 3.81
bridges_version1	**1.00 ± 0.00**	24.90 ± 20.72	156.88 ± 14.34
car	**108.20 ± 16.09**	173.10 ± 6.51	171.92 ± 4.45
cylinder_bands	4.20 ± 1.03	**1.00 ± 0.00**	211.44 ± 9.39
glass	**23.20 ± 10.56**	44.80 ± 5.20	86.44 ± 3.14
hepatitis	**6.60 ± 8.58**	15.40 ± 4.40	71.80 ± 4.77
iris	**6.20 ± 1.69**	8.00 ± 1.41	20.36 ± 1.81
kdd_synthetic	**1.00 ± 0.00**	37.80 ± 4.34	26.16 ± 2.45
segment	**78.00 ± 8.18**	80.60 ± 4.97	132.76 ± 3.48
semeion	34.00 ± 12.30	55.00 ± 8.27	**19.00 ± 0.00**
shuttle_landing	1.00 ± 0.00	1.00 ± 0.00	5.64 ± 1.69
sick	**45.20 ± 11.33**	46.90 ± 9.41	153.70 ± 8.89
tempdiag	5.00 ± 0.00	5.00 ± 0.00	5.32 ± 1.04
tep.fea	13.00 ± 2.83	**8.20 ± 1.69**	18.84 ± 1.97
vowel	**175.80 ± 23.72**	220.70 ± 20.73	361.42 ± 5.54
winequality_red	**151.80 ± 54.58**	387.00 ± 26.55	796.00 ± 11.22
winequality_white	**843.80 ± 309.01**	1367.20 ± 58.44	2525.88 ± 13.17

method, which is a surprise considering that MEP is usually a neglected pruning method in the DT literature.

The main advantage of HEAD-DT is that it automatically searches for the suitable components (with their own biases) according to the data set being investigated. It is hard to believe that a researcher would combine such a distinct set of components like those in Algorithm 1 to achieve 100% accuracy in a particular data set.

Algorithm 1: Algorithm designed by HEAD-DT for the Semeion data set.

```
1 Recursively split nodes with the Chandra-Varghese criterion;
2 Aggregate nominal splits in binary subsets;
3 Perform step 1 until class-homogeneity or the minimum number
  of 5 instances is reached;
4 Perform MEP pruning with m = 10;

  When dealing with missing values:
1 Calculate the split of missing values by performing
  unsupervised imputation;
2 Distribute missing values by assigning the instance to all
  partitions;
3 For classifying an instance with missing values, explore all
  branches and combine the results.
```

5. RELATED WORK

To the best of our knowledge, no work to date attempts to automatically design full DT induction algorithms. The most related approach to this work is HHDT (Hyper-Heuristic Decision Tree) [34]. It proposes an EA for evolving heuristic rules in order to determine the best splitting criterion to be used in non-terminal nodes. Whilst this approach is a first step to automate decision-tree induction algorithms, it evolves a single component of the algorithm (the choice of splitting criterion), and thus should be further extended for being able to generate full DT induction algorithms.

A somewhat related approach is the one presented by Delibasic et al. [14]. The authors propose a framework for combining DT components, and test 80 different combination of design components on 15 benchmark data sets. This approach is not a hyper-heuristic, since it does not present an heuristic to choose among different heuristics. It simply selects a fixed number of component combinations and test them all against traditional DT algorithms (C4.5, CART, ID3 and CHAID). We believe our strategy is more robust, since by using an evolutionary algorithm, we can search for solutions in a much larger search space. Currently, HEAD-DT can search for more than 127 million different algorithms.

6. CONCLUSIONS AND FUTURE WORK

In this work, we presented HEAD-DT, a hyper-heuristic algorithm to automatically design top-down decision-tree induction algorithms. These algorithms have been manually improved for the last 40 years, resulting in a great number of approaches for each of their design components. Since the human manual approach for testing all possible modifications in the design components of decision-tree algorithms would be unfeasible, we believe the evolutionary search of HEAD-DT constitutes a robust and efficient solution for the problem.

We performed a thorough experimental analysis in which the algorithms automatically designed by HEAD-DT were compared to state-of-the-art decision-tree induction algorithms CART [5] and C4.5 [32] in 20 public UCI data sets. We assessed the performance of HEAD-DT through 2 different performance measures, accuracy and F-Measure, and also a complexity measure, tree size. The experimental analysis suggested that HEAD-DT can generate algorithms which perform significantly better than CART and C4.5, though generating significantly larger trees. Bearing in mind that an accurate prediction system is widely preferred over a less-accurate (but simpler) system, we believe that HEAD-DT arises as an effective algorithm for future applications of decision trees.

As future work, we intend to develop a multi-objective fitness function, allowing the trade-off between predictive performance and parsimony. In addition, we plan to investigate whether a more sophisticated search system, such as grammar-based genetic programming, can outperform our current HEAD-DT implementation. Optimizing the evolutionary parameters of HEAD-DT is also a topic left for future research.

Acknowledgements

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.

7. REFERENCES

[1] R. C. Barros, M. P. Basgalupp, A. C. P. L. F. de Carvalho, and A. A. Freitas. A survey of evolutionary algorithms for decision tree induction. *IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and Reviews*, In press., 2011.

[2] R. C. Barros, R. Cerri, P. A. Jaskowiak, and A. C. P. L. F. de Carvalho. A Bottom-Up Oblique Decision Tree Induction Algorithm. In *11th International Conference on Intelligent Systems Design and Applications*, pages 450 –456, 2011.

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

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

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

[6] L. Breslow and D. Aha. Simplifying decision trees: A survey. *The Knowledge Engineering Review*, 12(01):1–40, 1997.

[7] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward. Exploring Hyper-heuristic Methodologies with Genetic Programming. In *Colaborative Computational Intelligence*, pages 177–201. Springer, 2009.

[8] B. Cestnik and I. Bratko. On estimating probabilities in tree pruning. In *EWSL'91*, pages 138–150. Springer, 1991.

[9] B. Chandra, R. Kothari, and P. Paul. A new node splitting measure for decision tree construction. *Pattern Recognition*, 43(8):2725–2731, 2010.

[10] B. Chandra and P. P. Varghese. Moving towards efficient decision tree construction. *Information Sciences*, 179(8):1059–1069, 2009.

[11] J. Ching, A. Wong, and K. Chan. Class-dependent discretization for inductive learning from continuous and mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):641–651, 1995.

[12] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

[13] R. L. De Mántaras. A Distance-Based Attribute Selection Measure for Decision Tree Induction. *Machine Learning*, 6(1):81–92, 1991.

[14] B. Delibasic, M. Jovanovic, M. Vukicevic, M. Suknovic, and Z. Obradovic. Component-based decision trees for classification. *Intelligent Data Analysis*, 15:1–38, Aug. 2011.

[15] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[16] F. Esposito, D. Malerba, and G. Semeraro. A Comparative Analysis of Methods for Pruning Decision Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

[17] U. Fayyad and K. Irani. The attribute selection problem in decision tree generation. In *National Conference on Artificial Intelligence*, pages 104–110, 1992.

[18] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[19] M. Gleser and M. Collen. Towards automated medical decisions. *Computers and Biomedical Research*, 5(2):180–189, 1972.

[20] R. Iman and J. Davenport. Approximations of the critical region of the friedman statistic. *Communications in Statistics*, pages 571–595, 1980.

[21] B. Jun, C. Kim, Y.-Y. Song, and J. Kim. A New Criterion in Selection and Discretization of Attributes for the Generation of Decision Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):1371–1375, 1997.

[22] B. Kim and D. Landgrebe. Hierarchical classifier design in high-dimensional numerous class cases. *IEEE Transactions on Geoscience and Remote Sensing*, 29(4):518–528, 1991.

[23] I. Kononenko, I. Bratko, and E. Roskar. Experiments in automatic learning of medical diagnostic rules. Technical report, Jozef Stefan Institute, Ljubljana, Yugoslavia, 1984.

[24] W. Loh and Y. Shih. Split selection methods for classification trees. *Statistica Sinica*, 7:815–840, 1997.

[25] J. Martin. An exact probability metric for decision tree splitting and stopping. *Machine Learning*, 28(2):257–291, 1997.

[26] J. Mingers. Expert systems - rule induction with statistical data. *Journal of the Operational Research Society*, 38:39–47, 1987.

[27] J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3(4):319–342, 1989.

[28] T. Niblett and I. Bratko. Learning decision rules in noisy domains. In *6th Annual Technical Conference on Expert Systems*, pages 25–34, 1986.

[29] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[30] J. R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[31] J. R. Quinlan. Unknown attribute values in induction. In *6th International Workshop on Machine Learning*, pages 164–168, 1989.

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

[33] P. C. Taylor and B. W. Silverman. Block diagrams and splitting criteria for classification trees. *Statistics and Computing*, 3:147–161, 1993.

[34] A. Vella, D. Corne, and C. Murphy. Hyper-heuristic decision tree induction. *World Congress on Nature & Biologically Inspired Computing*, pages 409–414, 2009.