IGI PUBLISHING
ITJ3905
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.igi-pub.com

This paper appears in the publication, **International Journal of Data Warehousing and Mining**, Volume 3, Issue 4
edited by **David Taniar© 2007, IGI Global**

Evolutionary Induction of Mixed Decision Trees[1]

Marek Kretowski, Bialystok Technical University, Poland

Marek Grzes, Bialystok Technical University, Poland

ABSTRACT

This article presents a new evolutionary algorithm (EA) for induction of mixed decision trees. In non-terminal nodes of a mixed tree, different types of tests can be placed, ranging from a typical inequality test up to an oblique test based on a splitting hyper-plane. In contrast to classical top-down methods, the proposed system searches for an optimal tree in a global manner, that is it learns a tree structure and finds tests in one run of the EA. Specialized genetic operators are developed, which allow the system to exchange parts of trees, generating new sub-trees, pruning existing ones as well as changing the node type and the tests. An informed mutation application scheme is introduced and the number of unprofitable modifications is reduced. The proposed approach is experimentally verified on both artificial and real-life data and the results are promising. Scaling of system performance with increasing training data size was also investigated.

Keywords: *decision trees; evolutionary algorithms; global induction; mixed decision trees*

INTRODUCTION

Decision trees (Murthy, 1998) are one of the most frequently applied data mining approaches. There exist many induction algorithms, which tackle the problem of building decision trees in a different way. Most frequently, they differ in the measure for the test assessment, but also in the type of search in solution space (i.e., top-down vs. global). From a user's point of view, one of the most important features of a decision tree is a test representation in the internal nodes. In typical univariate trees, two types of tests are usually permitted. For a nominal attribute, mutually exclusive sets of feature values are associated with each branch, whereas for a continuous valued feature inequality tests are applied. In the case of multivariate trees, more than one feature can be used to create a test. Oblique tests based on a splitting hyper-plane are the most widely used form of multivariate tests. Most of the DT-based systems are homogeneous, which means that they take advantage of only one type of test (i.e., univariate or oblique). *C4.5* (Quinlan, 1993) can be treated as one of the best-known representatives of the first type, whereas *OC1* (Murthy, Kasif, & Salzberg, 1994) is a good example of an oblique tree inducer.

These two systems belong to the group of decision tree algorithms, which represent the de-facto standard for empirical evaluations and are commonly used for comparisons.

The term *mixed decision trees* was proposed by Llora and Wilson (2004) to describe trees in which different types of tests can be exploited. One of the first and best-known examples of such an approach is the *CART* system (Breiman, Friedman, Olshen, & Stone, 1984). This system is able to search for a linear combination of non-nominal features in each node and it compares the obtained test with the best univariate test. However, it should be noted that *CART* has a strong preference for simpler tests; it rarely uses the more elaborate splits. Another form of a hybrid classifier is proposed by Brodley (1995). Her *MCS* system combines univariate tests, linear machines, and instance-based classifiers (*k-NN*) and during the top-down generation of a tree classifier it recursively applies automatic bias selection. Recently, a fine-grain parallel model *GALE* (Llora et al., 2004) was applied to generate decision trees, which employ inequality and oblique tests.

There are two main approaches to the decision tree induction: top-down and global. The first one is based on a greedy recursive procedure of test searching and sub-node creation until a stopping condition is met. The locally optimal tests according to the predefined criteria are chosen in each step, but such a procedure does not guarantee the global optimality of the final tree. This problem can be easily observed when there is a strong interaction between features. Only treating them together can lead to the optimal solution. Additionally, the post-pruning is usually applied after the actual top-down induction to avoid the problem of over-fitting the training data. It should be noted that post-pruning techniques have only limited ability to correct the tree structure. The *C4.5* and *OC1* systems apply the top-down approach and are used for the comparison in this article.

In contrast to the classical top-down approach, global algorithms try to simultaneously search for both the tree structure and all tests in non-terminal nodes. This process is obviously much more computationally complex but it can reveal hidden regularities, which are almost undetectable by greedy methods. The global induction is mainly represented by systems based on evolutionary approach.

Evolutionary computations (Michalewicz, 1996) are stochastic techniques, which have been inspired by the process of biological evolution. Their success is attributed to the ability to avoid local optima, which is their main advantage over greedy search methods. Evolutionary techniques are known to be useful in many data mining tasks (Freitas, 2002). They were successfully applied in the framework of both top-down and global systems to learning univariate (Fu, Golden, Lele, Raghavan, & Wasil, 2003; Koza, 1991; Nikolaev & Slavov, 1998; Papagelis & Kalles, 2001) and oblique trees (Bot & Langdon, 2000; Chai, Huang, Zhuang, Zhao, & Sklansky, 1996; Cantu-Paz & Kamath, 2003; Kretowski, 2004).

The global approach based on evolutionary algorithms for decision tree induction was investigated in our previous articles. We showed that homogeneous trees, univariate (Kretowski & Grzes, 2005a) or oblique (Kretowski & Grzes 2005b, 2006) can be effectively induced and we demonstrated that globally generated classifiers are generally less complex with at least comparable accuracy. In this article, we want to merge the two developed methods in one system, which will be able to induce mixed trees.

The rest of the article is organized as follows. In the next section our global system for induction of mixed decision trees is presented. Experimental validation of the approach on both artificial and real-life datasets is presented in the third section. The article finishes with our conclusion.

GLOBAL INDUCTION OF MIXED DECISION TREES

The algorithm proposed in this article applies a global approach to decision tree induction based on evolutionary computation. The general structure of the proposed solution follows a typical evolutionary framework (Michalewicz, 1996).

In our previous research, we applied such a technique to build homogenous decision trees. In each of investigated algorithms, one test type was used i.e. axis-parallel or oblique. Two separate systems to induce decision trees with these two kinds of tests were developed, tested, and presented in our previous articles Kretowski et al. (2005a, 2005b, 2006). *GDT-AP* is a version, which uses axis-parallel and *GDT-OB* oblique tests. For a detailed description of these solutions we refer the reader to (Kretowski et al., 2005a) for the description of the *GDT-AP* and (Kretowski et al., 2005b, 2006) *GDT-OB* system. Both systems produce decision trees using evolutionary computation of decision trees, which are encoded as individuals in a natural tree-like structure. The processed populations can contain a large variety of individuals of different structure and tests in internal nodes. In *GDT-AP,* we allow axis-parallel tests on numerical and nominal attributes. Special informed types of modifications of individuals are used to realize genetic operators because decision trees are extremely sensitive on any changes in upper parts of them especially. We try to apply genetic operators in the informed way. For example for nominal tests, we use regrouping of symbolic values. It allows obtaining different combinations of these values and finding better inner disjunction. In *GDT-OB,* multivariate tests based on a hyper-plane are used. These types of tests require different rationality in applied genetic operators. One of the most important things in this case is feature selection. It requires a specialised mutation operator, which eliminates features when corresponding weights in the hyper-plane are set to zero.

The evolutionary process incorporated in both *GDT-AP* and *GDT-OB* systems is steered by the fitness function, which seems to be the most sensitive part of the entire solution. In our case we use a penalised fitness function, which tries to balance the influence of the test set accuracy of the evaluated tree against its complexity. Such an approach requires defining a user specified parameter, which balances the influence of both these factors. Our previous analysis showed that it is possible to find relatively good values of this parameter which lead to the best or very good results on most of evaluated (for particular value of this parameter) datasets. Experiments showing the tuning of this parameter are presented in our previous work (Kretowski et al., 2005b).

In this article, we present a continuation and unification of our work and propose a combined solution, which can induce decision trees with both axis-parallel and oblique tests.

In this section, a relatively detailed description of this approach is presented especially with respect to issues that are specific to mixed trees.

Representation, Initialization, and Termination Condition

A mixed decision tree is a complicated tree structure, in which the number of nodes, test types, and even the number of test outcomes are not known in advance for a given learning set. Moreover additional information (e.g., about input feature vectors associated with each node should be accessible during the induction). As a result, decision trees are not specially encoded in individuals and they are represented in their actual form.

There are three possible test types in internal nodes: two univariate and one multivariate. In the case of univariate tests, a test representation depends on the considered attribute type. For nominal attributes, at least one attribute value is associated with each branch starting in the node, which means that an internal disjunction is implemented. For continuous-valued features, typical inequality tests with two outcomes are used. In order to speed up the search process only boundary thresholds (a boundary threshold for the given attribute is defined as a midpoint between such a successive pair of examples in the sequence sorted by the increasing value of the attribute, in which the examples belong to two different classes) are considered as potential splits and they are calculated before starting the EA. Finally, an oblique test with binary outcome can also be applied as a multivariate test. A splitting hyper-plane is represented by a fixed size table of real values corresponding

to the weight vector and the threshold. The inner product is calculated to decide where an example is routed.

Before starting the actual evolution, an initial population is created. All initial trees are homogeneous, but half of the population is initialized with univariate tests and the other part with oblique tests. A simple top-down algorithm is applied to generate all individuals. In each potential internal node it chooses randomly a pair of objects from different classes and searches for a test, which separates them to distinct sub-trees. In the case of a univariate tree, such a test can be directly constructed for any feature with different feature values. When an oblique test is necessary, the splitting hyper-plane is perpendicular to the segment connecting the two drawn objects and placed in a halfway position.

The recursive partitioning is finished when all training objects in a node belong to the same class or the number of objects in a node is lower than the predefined value (default value: 5).

Application of the above simple (but fast) algorithm to the full training data leads to the population of initial trees, which are generally diverse but too large compared to the final tree. It is especially visible in case of big datasets composed of thousands of feature vectors and it obviously slows down the evolution. In order to avoid the aforementioned problem, randomly chosen subsamples of the original training data (10% of data, but not more then 500 examples) are used to generate initial trees.

The evolutionary induction terminates when the fitness of the best individual does not improve during a fixed number of generations (default value is equal to 1000) or the maximum number of generations (default value: 10000) is reached.

Genetic Operators

There are two specialized genetic operators corresponding to the classical mutation and crossover. Application of both operators can result in changes of the tree structure and tests in non-terminal nodes.

A mutation-like operator is applied with a given probability to a tree (default value is 0.8) and it guarantees that at least one node of the selected individual is mutated. In this article we extended the operator application scheme proposed in Kretowski et al. (2006). Firstly, the type of the node (leaf or internal node) is randomly chosen with equal probability and if a mutation of a node of this type is not possible, the other node type is chosen. A ranked list of nodes of the selected type is created and a mechanism analogous to ranking linear selection (Michalewicz, 1996) is applied to decide which node will be affected. While concerning internal nodes, the location (the level) of the node in the tree and the quality of the subtree starting in the considered node are taken into account. It is evident that modification of the test in the root node affects whole tree and has a great impact, whereas mutation of an internal node in lower parts of the tree has only a local impact. In the proposed method, nodes on higher levels of the tree are mutated with lower probability and among nodes on the same level the number of misclassified objects by the subtree is used to sort them. Additionally, perfectly classifying nodes with only leaves as descendants and with a test composed of one feature are excluded from a ranking, because their mutation cannot improve the fitness. As for leaves, the number of objects from other classes than the decision assigned to the leaf is used to put them in order, but homogenous leaves are not included. As a result, leaves, which are worse in terms of classification accuracy, are mutated with higher probability. In Figure 1, an example of constructing the ranking lists of leaves and internal nodes is presented.

To specify how probabilities presented in this figure were calculated, the existence of the ordered list of internal nodes or leaves is assumed. The calculation of discussed probabilities is the same for these two types of lists. Figure 1 shows how to obtain these ordered lists. The probability $P(i)$ to select a node, which is at the position i in the given sorted list, is calculated according to Equation 1 (Michalewicz, 1996):

Figure 1. Preparation step before applying actual mutation: creation of two ranked lists of internal nodes and leaves

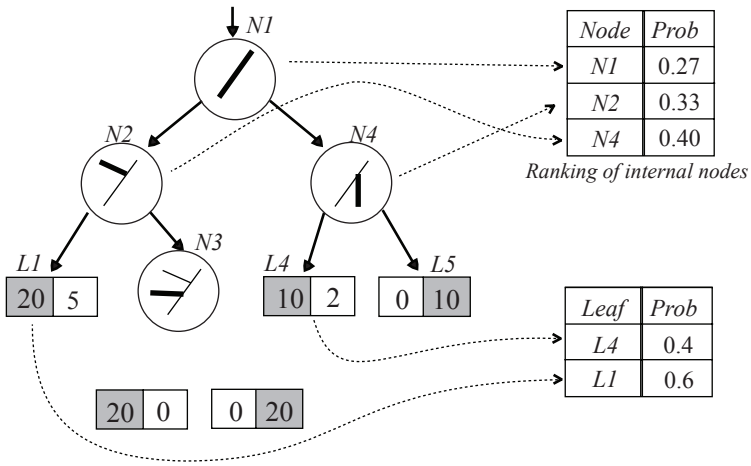
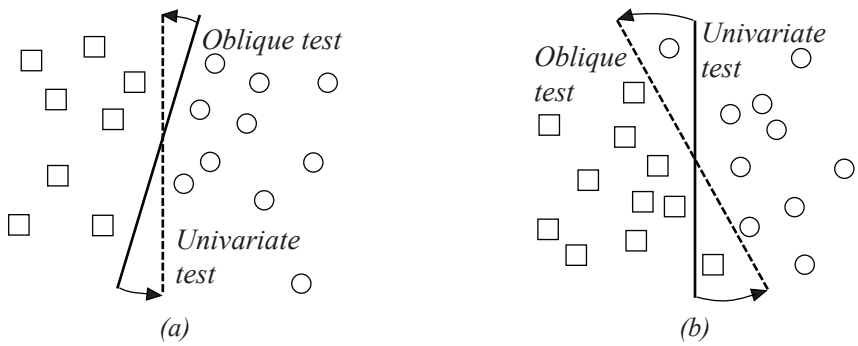


Figure 2. Changing the test type: a) simplifying oblique test to univariate one b) extending axis-parallel test to oblique one



$$P(i) = \frac{2 - SP + 2(SP-1)\dfrac{(i-1)}{(N-1)}}{\sum_{j=1..N}\left[2 - SP + 2(SP-1)\dfrac{(j-1)}{(N-1)}\right]} \quad (1)$$

where N represents the number of nodes in the list and SP stands for selective pressure. Variables i and j represent positions in the list and are in the range $[1,N]$. Linear ranking allows values of selective pressure in [1.0, 2.0]. In this case 1.2 was used. Values of probabilities given in Figure 1 were also obtained with this default value of SP parameter.

Modifications performed by the mutation operator depend on the node type (i.e., if the considered node is a leaf node or an internal node). For a non-terminal node, a few possibilities exist:

- A completely new test of the same or different type can be drawn; new tests are

created in the same way as described for the initialization,

- The existing test can be altered by shifting the splitting threshold (continuous-valued feature), by re-grouping feature values (nominal features) or by shifting the hyperplane (oblique test); these modifications can be purely random or can be performed according to the adapted dipolar operator (Kretowski, 2004),
- An oblique test can be simplified to the corresponding inequality test by eliminating (zeroing) the smallest coefficients or an axis-parallel test can be transformed into oblique one (see Figure 2),
- A test can be replaced by another test or tests can be interchanged,
- One sub-tree can be replaced by another sub-tree from the same node,
- A node can be transformed (pruned) into a leaf.

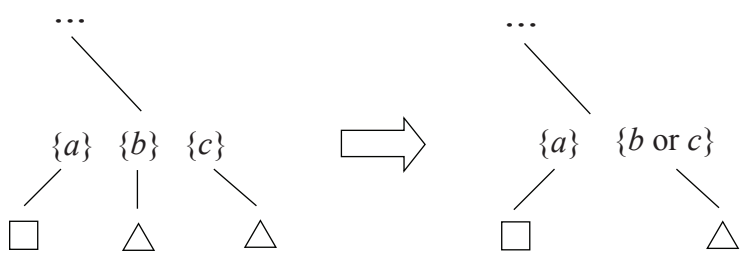
Modifying a leaf makes sense only if it contains objects from different classes. The leaf is transformed into an internal node and a new test is randomly chosen. The search for effective tests can be recursively repeated for all newly created descendants (leaves). As a result, the mutated leaf can be replaced by a subtree, which potentially accelerates the evolution.

There are also several variants of crossover operators (applied with a default probability 0.2). All of them start with selecting the cross-over positions in two affected individuals. One node is randomly chosen in each of two trees. In the most straightforward variant, the subtrees starting in the selected nodes are exchanged. This corresponds to the classical cross-over from genetic programming. In the second variant, which can be applied only when non-internal nodes are randomly chosen and the numbers of outcomes are equal, only tests associated with the nodes are exchanged. The last variant is also applicable only when non-internal nodes are drawn and the numbers of descendants are equal. In this case, branches, which start from the selected nodes, are exchanged in random order.

The application of any genetic operator can result in a necessity for relocation of the input vectors between parts of the tree rooted in the modified node. Additionally the local maximization of the fitness is performed by pruning lower parts of the sub-tree on the condition that it improves the value of the fitness.

It was observed by Bennett, Cristianini, Shave-Taylor, and Wu (2000) that enlarging the margin in oblique decision trees (the margin is defined as the distance between decision boundary and the closest input feature vectors) is profitable in term of classification accuracy. In the presented system, a simple mechanism called *centring* based on this observation is introduced and it is applied to the best decision tree found. In case of an oblique test, centring is performed as follows: the closest two objects

Figure 3. Merging leaves with the same decision, simplifying nominal tests using inner disjunction



to the splitting hyper-plane are determined on the opposite sides of it and if the objects found belong to different classes, the hyper-plane is shifted (centred) by modifying the threshold. In case of a univariate inequality test, the threshold can also be shifted to half distance between corresponding feature values. It should be noted that such a post-processing does not change the fitness corresponding to the final tree. Obviously, the centring cannot be applied to tests based on nominal features. For them, another kind of test improvement is used. If there is an internal node with nominal test, and there are descendant leaves, which have the same decision, then such leaves are merged and inner disjunction is used in the splitting node (see Figure 3).

Fitness Function

The evolutionary search process is very sensitive to its fitness function. When concerning a classification task it is well known that the direct optimization of the classifier accuracy measured on the learning set leads to an over-fitting problem. In a typical top-down induction of decision trees, the over-specialization problem is mitigated by defining a stopping condition and by applying a post-pruning (Esposito, Malerba, & Semeraro, 1997). In our approach, the search for an optimal structure is embedded into the evolutionary algorithm by incorporating a complexity term in the fitness function. A similar idea is used in cost complexity pruning in the CART system. The fitness function is maximized and has the following form:

$$Fitness(T) = Q_{\mathrm{Re}class}(T) - \alpha\,(Comp(T) - 1.0) \tag{2}$$

where $Q_{Reclass}(T)$ is the training accuracy (reclassification quality) of the tree T and α is the relative importance of the classifier complexity (default value is 0.005). In the simplest form the tree complexity $Comp(T)$ can be defined as the classifier size, which is usually equal to the number of nodes. The penalty associated with the classifier complexity increases proportionally with the tree size and prevents classifier over-specialization. Subtracting 1.0 eliminates the penalty when the tree is composed of only one leaf (in majority voting).

This simple complexity definition is adequate for a homogeneous tree composed of only univariate tests. However, when oblique tests are also considered, it seems that a more elaborate solution is necessary. It is rather straightforward that an oblique split based on a few features is more complex than a univariate test and that we should apply preference to simpler tests as an inductive bias. As a consequence the tree complexity should also reflect the complexity of the tests. However, it is not easy to definitely decide how to balance different test complexities because it depends on the problem solved and user preferences. In such a situation we decided to define the tree complexity $Comp(T)$ in a flexible way and allow the user to tune its final form:

$$Comp(T) = \left|N_{leaf}(T)\right| + \sum_{n \in N_{\mathrm{int}}(T)} \left[1 + \beta\,(F(n) - 1)\right] \tag{3}$$

where $N_{leaf}(T)$ and $N_{int}(T)$ are sets of leaves and internal nodes correspondingly, $F(n)$ is the number of features used in the test associated with the node n and $\beta \in [0, 1]$ is the relative importance of the test complexity (default value 0.2). The complexity of the tree is defined as a sum of the complexities of the nodes and it is assumed that for leaves and internal nodes with univariate tests the node complexity is always equal to 1.0. It can be also observed that when $\beta=1$ the number of features included in the test is used as the node complexity, and if $\beta=0$, then the node complexity is 1.0.

EXPERIMENTAL RESULTS

The proposed approach to learning mixed decision trees is assessed on both artificial and real life datasets and is compared to the well-known top-down univariate (*C*4.5 Quinlan, 1993) and oblique (*OC*1 Murthy et al., 1994) decision tree systems. It is also compared to two homogenous versions of our global *GDT* system: univariate

GDT-AP (Kretowski et al., 2005a) and oblique *GDT-OB* (Kretowski et al., 2006). All prepared artificial datasets comprise training and testing parts. In the case of data from a UCI repository (Blake, Keogh, & Merz, 1998) for which testing data is not provided, 10-fold stratified cross-validation was employed. Each experiment on all stochastic algorithms (i.e., all except *C*4.5) was performed 10 times and the average result of such an evaluation was presented. The number of leaves is the complexity measure (size) of compared trees. The *OC*1 system was run with different values of the seed that initializes the random number generator. Our system is initialized by the system time.

A statistical analysis of the obtained results was done by the Friedman test with the corresponding Dunn's multiple comparison test (significance level equal to 0.05) as recommended by Demsar (2006).

We also perform an experiment to quantify the scalability of our evolutionary algorithm. This experiment, which is performed on larger artificial datasets is presented and discussed at the end of this section.

Artificial Datasets

Two types of artificial datasets are used. Some of them (see e.g., *normchessap* in Figure 4) require axis-parallel and others (*ls2* in Figure 4) oblique tests. Such a methodology is used to assess the flexibility of the proposed approach.

The prefix *chess* in suggested names indicates the so-called chessboard domain and the given numbers represent the number of partitions at each axis. For example *chess3x3* (Figure 6a) means that it is the two dimensional domain with three intervals in each dimension. *Chess2x2x2* has three dimensions and two intervals at each of them. The abbreviation *ob* given after prefix *chess* means that it is a rotated (i.e. oblique) chessboard like *chessob4cl* in Figure 4. *Chessob2cl* is analogous except it has two classes.

The prefix *norm* reflects not clearly defined borders between classes and normal distribution

Figure 4. Examples of artificial datasets

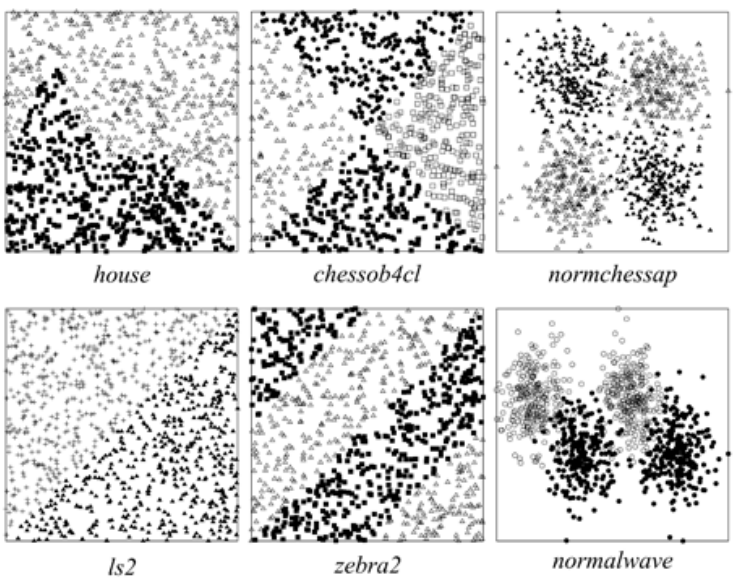


Figure 5. Results on artificial data

Dataset	C4.5		OC1		GDT-Mix		GDT-AP		GDT-OB	
	size	quality	size	quality	size	quality	size	quality	size	quality
chess2x2	1	50	10.1	89.3	4	99.8	4	99.8	4	99.3
chess2x2x2	1	50	23.8	71	8	99.7	8	99.7	8.2	97
chess3x3	9	99.7	21.1	73.7	9	99.3	9	99.7	9.9	97.1
chessob2cl	33	95.6	7	77.3	4.1	99.1	17.9	92.6	4.7	99.1
chessob4cl	35	94.6	4.3	49.8	4	98.9	18	92.1	4.4	98.4
house	21	97.4	8.2	92.8	3.8	96.9	13.3	96.6	4	96.7
ls10	284	77.3	7.3	95.3	2	97.6	18.8	70.7	2	97.2
ls2	22	97	2	99.7	2	99.9	14	95.7	2	99.9
normal	5	90	7.3	87.9	3.8	89.6	25.7	86.9	4	90
normchessap	1	50	11.2	85.5	4	95.5	4.2	95.5	4	95.4
normchessob	19	93	11	83.3	4	93.6	9.3	92.6	4	93.6
normwave	15	94	8.4	90.3	4	94.4	9.1	93.5	4	94.9
zebra1	25	95.3	3	83.5	3	99.6	15.3	94.6	3	99.3
zebra2	2	59.5	4.8	94.1	4	98.2	21.4	91.6	4.4	98.7
zebra3	57	91.2	8.2	24.3	8.4	97	31.5	88.8	8.8	96.8

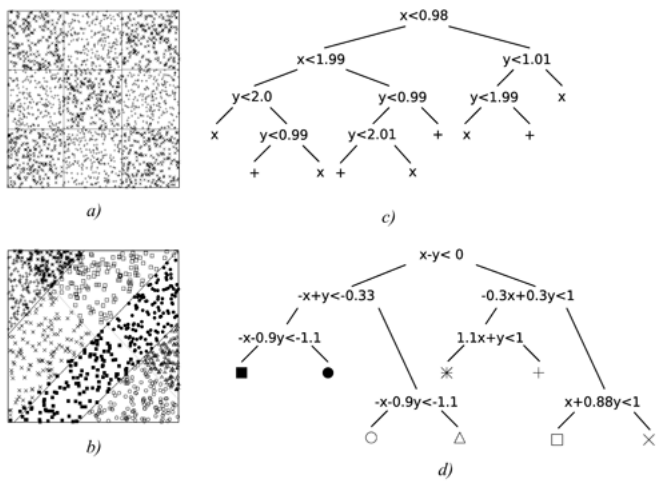
(rnorm in the R system was used) of generated observations (e.g., *normchessap,* which has overlapping borders in contrast to *ls2*). Observations in other datasets were generated with uniform distribution. *Normchessap* represents axis-parallel chessboard with overlapping borders and *normchessob* its rotated, oblique version. Dataset *synth* was taken from the collection of datasets used in this book (Ripley, 1996).

Two datasets with prefix *ls* represent linearly separable domain where classes are divided by a hyper-plane: $x_1+...+x_n<x_{n+1}+...+x_m$, where $n = 0.5m$ and $m = 2$ in *ls2* and $m = 10$ in *ls10*. The datasets with prefix *zebra* contain oblique stripes. *Zebra2* is presented in Figure 4 and *zebra1* is similar except it has three regions. All datasets with prefix *zebra* (including *zebra3* presented in Figure 6b) were generated using Figure 6 from (Cantu-Paz et al., 2003) as a template.

Most of the datasets have two continuous-valued features and only the *ls10* (linearly separable) dataset has 10 features and *chess2x2x2* has three features. The number of examples was varied and depends on the number of distinct regions. In the training part it ranged from 1000 (for simple 2-dimensional problems) to 4000 (for *ls10*). The testing part is twofold larger in each case.

The results on the range of datasets designed for this investigation are collected in Figure 5. Because we analyze artificial data in this experiment, we know how the optimal solution can be represented (in terms of used tests). There are certain classification tasks, like for instance the classical *chessboard* problem, that suit very well univariate decision trees. There are also linearly separable datasets (like *ls10*) for which splits based on hyper-planes are highly recommended to avoid a staircase-like structure. The main aim of our endeavour in this

Figure 6. Decision trees obtained by GDT-Mix system for chess3x3 and zebra3 datasets (b) and (d) and the corresponding dataset scatterplots with drawn splits (a) and (c)



work is to show that *GDT-Mix* can easily adjust to the specific problem. The analysis of Figure 5 proved that the *GDT-Mix* inducer performs better on axis parallel data when compared to oblique systems and on linearly separable data when compared to axis parallel systems. It is also important that statistical analysis does not show significant differences between the *GDT-Mix* algorithm and the systems specialized for certain problems when a comparison is made on such problems. Our universal system performs as well as the specialized systems. This is its very strong point. The statistical test on all artificial datasets indicates that *GDT-Mix* is significantly better in terms of quality than *C4.5*, *OC1* and *GDT-AP*. As for *GDT-OB*, it is statistically better than *OC1*. The comparisons based on the second measure (the tree size) are also favourable. Statistical analysis reveals that *GDT-Mix* produces significantly smaller trees than *C4.5*, *GDT-AP* and *OC1*. This score is easily justified for typical univariate inducers, because in the case of problems, which require oblique splits our GDT-Mix, system takes advantage of such splits.

Discussed results show that the proposed algorithm is more flexible in terms of representation, which can be modified during the induction. For that reason, more detailed analysis of these results aims at investigating the obtained decision trees. Such trees for relatively complex axis parallel and linearly separable classification tasks are presented in Figure 6. These trees present a promising result because the *GDT-Mix* system managed to find the type of tests that suit the data in the best way and was able to apply them to build trees that perform very competitively while comparing to results of specialized systems. In Figure 6a and 6b, splits from decision trees are additionally drawn to present how the input space is partitioned by the global inducer. These splits show that trees obtained in this experiment have an optimal structure (the empirical superiority of global induction).

Real-Life Data

Results on real-life data are divided into two groups. In the first one, *GDT−Mix* is compared with all evaluated algorithms on datasets where

the instances have only numeric (continuous) feature values (Figure 7). In the second group, the proposed system is compared with univariate tree induction algorithms on datasets that have both nominal and continuous attributes (Figure 8).

The analysis of these results shows that there are no statistically significant differences in the quality between the compared algorithms on all datasets. This is a favourable result. It means that the *GDT-Mix* system, which is designed to be universal to different kinds of tasks, performs as well as specialized counterparts. In the case of the size of the tree, the same statistical analysis of the Figure 8 indicates that *GDT-Mix* produces significantly smaller trees than *C4.5* and *GDT-AP*. A detailed inspection of this table shows that there are some datasets (e.g., *heart*) for which there is evident difference in the tree size, which shows the superiority of the *GDT-Mix* algorithm. This is a very useful feature of mixed trees. As for the Figure 7, there is a statistically significant difference in terms of the size. *GDT-Mix* and *GDT-OB* produced smaller trees than *C4.5* and *GDT-AP*.

One of the decision trees obtained for the real-life *heart* data is presented in Figure 9. This dataset was chosen for investigation because it contains both nominal and continuous attributes and represents a quite easily understood problem (at least in terms of outcomes of the classifier). Figure 9 presents one of the decision trees (there were 10 runs of the algorithm on each dataset) that were obtained in our experiment for the *heart* data. In the presented tree, all three types of possible tests are used. This example underlines the advantage of decision trees of being self explanatory and easy to understand. In mixed decision trees, we can have tests both on nominal and continuous attributes, which is a desirable feature of this system.

Evaluation of Algorithm Performance on Large Datasets

One of the most significant deficiencies of solutions based on evolutionary techniques is that they are supposedly slow, especially on large, real-life problems. In order to check how well our solution can deal with large datasets, a performance test is presented in Figure 10. The experiment was conducted on two artificial datasets: *ls5* and *chess3x3* with different numbers of generated observations ($10^2 \div 10^5$).

The promising outcome of this experiment is that it shows that our algorithm can deal with relatively large datasets (100000 observations)

Figure 7. Results on real datasets with only continuous attributes

Dataset	C4.5		OC1		GDT-Mix		GDT-AP		GDT-OB	
	size	quality	size	quality	size	quality	size	quality	size	quality
balance-sc.	57	77.5	5.4	90	2.6	89.5	32.8	78.2	3.2	89.1
bcw	22.8	94.7	4.7	91.2	2	96.7	6.6	95.8	2	96.9
bupa	44.6	64.7	5.8	65.6	3.5	68.6	69.3	62.8	3	71.3
glass	39	62.5	4.5	55.7	11.6	66.4	40.4	63.6	11.6	68.8
page-blocks	82.8	97	15.6	96.6	3	94.9	7.5	96.4	3	95.3
pima	40.6	74.6	6.5	69.6	2.2	75.4	14.3	73.8	2.1	75.3
sat	435	85.5	58.3	78.9	6.4	81.5	19.2	83	7	83.1
vehicle	138.6	72.7	21.6	66.4	8.8	65.4	45.1	70.3	7.7	65.7
waveform	107	73.5	10.5	77.4	4.2	80.5	36.2	72.3	4.2	82.2
wine	9	85	3.2	87	4.2	91.5	5.2	86.3	4.8	90.9

Figure 8. Results on real datasets with both continuous and nominal attributes

Dataset	C4.5		GDT-Mix		GDT-AP	
	size	quality	size	quality	size	quality
australian	39	87	2.4	87.5	22.8	84.6
cars	31	97.7	3	97.9	4	98.7
cmc	136.8	52.2	4	55.4	13.1	53.8
german	77	73.3	3.8	72.4	16.5	73.4
heart	22	77.1	6.1	78.4	44.9	74.2
solar	20	73.1	4	71.3	33.7	73.6
vote	5	97	2	97	13.5	95.6

Figure 9. The decision tree for heart data found by GDT-Mix

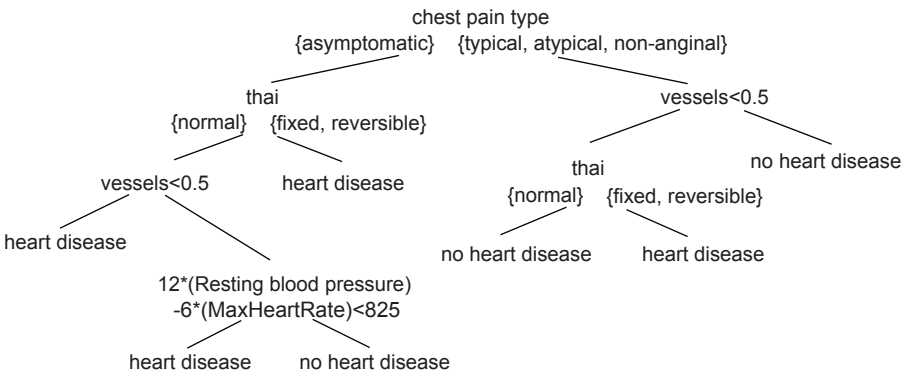
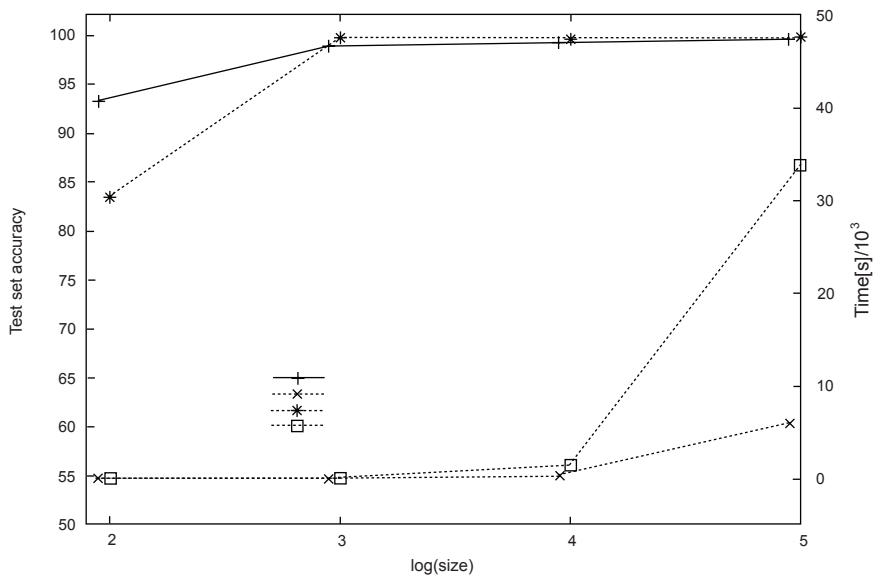


Figure 10. Performance of the algorithm on scaled up datasets (the number of observations in the range of 100÷10000)



in acceptable time. This time varies between about 2 (*chess3x3*) and 11 (*ls5*) hours as measured on a typical machine (Xeon 3.2GHz, 2GB RAM). There is an interesting (in the context of mixed decision trees) explanation for this discrepancy. The *chess3x3* dataset belongs to the class of tested problems for which the best solution contains only axis parallel splits. *ls5* on the other hand requires only one test which is based on hyper-plane. In our previous work on evolutionary induction of decision trees we have noticed that induction of oblique decision trees usually requires more computation time (because of, for example, the calculation of the inner product to determine location of observations and more time consuming evolutionary operators). On the tested datasets, *GDT-Mix* produces the optimal (at least in terms of representation) outcome. That is, an axis parallel decision tree for the *chess3x3* dataset and oblique (with one expected internal node) decision tree for the *ls5* dataset. Because the returned trees are of the desired type, it is highly probable that most of the population in especially later generations was dominated by trees of this type. It is a favourable observation for mixed decision trees algorithm which can flexibly choose the type of tests used in induced decision trees and by doing this make the induction process more efficient. The evaluation of the *chess3x3* dataset on oblique decision tree algorithm would lead to increase of experiment time but probably not up to the value obtained for *ls5* because *chess3x3* has two whereas *ls5* has five attributes.

CONCLUSION AND FUTURE WORKS

In the article, a new evolutionary algorithm for global induction of mixed decision trees is proposed. In the unified framework, both univariate and oblique tests are searched and applied in non-terminal nodes for optimal data splitting. The defined fitness function enables the controlling of the inductive biases.

Experimental validation shows that the algorithm is able to adapt to the problem being solved and to locally choose the most suitable test representation. Even though the evolutionary induction is computationally more complex than classical top-down approaches, we proved that it can be effectively applied to large datasets.

The presented approach is still under development. We are considering introducing additional test types, especially multivariate tests. Furthermore, the fitness function and especially the impact of the definition of the complexity term on the resulting decision tree will be studied in more detail. Obtained results are promising. But further analysis of the fitness function may lead to interesting results. In this problem especially, where the complexity of the decision tree comprises at least two factors: the number of nodes and the number of features used in these nodes. Minimum description length might be considered as one of the first-line choices.

Our approach applies mutation operators in an informed way (informed selection of individuals or parts of individuals to mutate). It uses ranking of nodes to decide which nodes should or should not be modified in a particular way. One possible improvement to our system is to use a similar informed approach to implement the crossover operator. At the moment this operator is uniformed. Used in this way, might be destructive, because the meaning of the sub-tree of a decision tree is *very* context dependent. When we move one sub-tree to a different location it might become useless given the new context and have to be pruned out.

Finally, is it a well-known fact that evolutionary techniques are well-suited for parallel architectures. We plan to speed up our system by re-implementing it in a distributed environment. It is especially important in the context of modern data mining applications, where huge learning sets are analyzed.

ACKNOWLEDGMENT

This work was supported by the grant W/WI/5/05 from Bialystok Technical University.

REFERENCES

Blake, C., Keogh, E., & Merz, C. (1998). *UCI repository of machine learning databases*. Irvine, CA: University of California, Department of Computer Science.

Bennett, K., Cristianini, N., Shave-Taylor, J., & Wu, D. (2000). Enlarging the margins in perceptron decision trees. *Machine Learning*, *41*, 295-313.

Bot, M., & Langdon, W. (2000). Application of genetic programming to induction of linear classification trees. *Proceedings of EuroGP'2000* (pp. 247.258). Springer, LNCS 1802.

Breiman, L., Friedman, J., Olshen, R., & Stone C. (1984). Classification and regression trees. Wadsworth International Group.

Brodley, C. (1995). Recursive automatic bias selection for classifier construction. *Machine Learning*, *20*, 63-94.

Cantu-Paz, E., & Kamath, C. (2003). Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *7*(1), 54-68.

Chai, B., Huang, T., Zhuang, X., Zhao, Y., & Sklansky, J. (1996). Piecewise-linear classifiers using binary tree structure and genetic algorithm. *Pattern Recognition*, *29*(11), 1905-1917.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1-30.

Esposito, F., Malerba, D., & Semeraro, G. (1997). A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(5), 476-491.

Freitas A. (2002). *Data mining and knowledge discovery with evolutionary algorithms*. Springer.

Fu, Z., Golden, B., Lele, S., Raghavan, S., & Wasil, E. (2003). A genetic algorithm based approach for building accurate decision trees. *INFORMS Journal of Computing*, *15*(1), 3-22.

Koza, J. (1991). Concept formation and decision tree induction using genetic programming paradigm. *Proceedings of International Conference on Parallel Problem Solving from Nature--Proceedings of 1st Workshop* (pp. 124-128). Springer, LNCS 496.

Kretowski, M. (2004). An evolutionary algorithm for oblique decision tree induction. *Proceedings of International Conference on Artificial Intelligence and Soft Computing* (pp. 423-437), Springer, LNCS 3070.

Kretowski, M., & Grzes, M. (2005a). Global learning of decision trees by an evolutionary Algorithm. In: Information Processing and Security Systems, Springer, 401–410.

Kretowski, M., & Grzes, M. (2005b). Global induction of oblique decision trees: an evolutionary approach. *Proceedings of Intelligent Information Processing And Web Mining. Springer* (pp. 309-318).

Kretowski, M., & Grzes, M. (2006). Evolutionary learning of linear trees with embedded feature selection. *Proceedings of International Conference on Artificial Intelligence and Soft Computing*, Springer, LNCS 4029.

Llora, X., & Wilson, S. (2004). Mixed decision trees: Minimizing knowledge representation bias in LCS. *Proceedings of Genetic and Evolutionary Computation Conference* (pp. 797-809), Springer, LNCS 3103.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs* (3rd ed.). Springer.

Murthy, S. (1998). Automatic construction of decision trees from data: A multidisciplinary survey. *Data Mining and Knowledge Discovery*, *2*, 345-389.

Murthy, S., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, *2*, 1-33.

Nikolaev, N., & Slavov, V. (1998). Inductive genetic programming with decision trees. *Intelligent Data Analysis*, *2*, 31-44.

Papagelis, A., & Kalles, D. (2001). Breeding decision trees using evolutionary techniques. *Proceedings of International Conference on Machine Learning* (pp. 393-400), Morgan Kaufmann.

Ripley, B. D. (1996). *Pattern recognition and neural networks*. Cambridge University Press.

Quinlan, J. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann.

ENDNOTE

[1] This article is an extended version of the paper presented at DaWaK'06

Marek Grzes received an MSc degree in computer science from Bialystok Technical University (Poland). After graduation in 2003, he has been working there as a teaching and research assistant. Since 2006, he has been studying for a PhD degree at The University of York (United Kingdom).

Marek Kretowski received an MSc degree in computer science in 1996 from Bialystok Technical University, Poland. His PhD thesis, defended in 2002, was prepared in the framework of collaboration between Laboratory of Signal and Image Processing (LTSI), University of Rennes 1, INSERM, France and faculty of computer science, Bialystok Technical University, Poland. From 2000 to 2002 M. Kretowski was a scholar of the French Government. Now he works as an assistant professor in the faculty of computer science, Bialystok Technical University. His current research focuses on data mining methods and biomedical applications of computer modeling.