Constructing Binary Decision Trees using Genetic Algorithms

Sung-Hyuk Cha, and Charles Tappert
Computer Science Department, Pace University
Pleasantville, NY, USA
Email: {scha,ctappert}@pace.edu

Abstract - *Decision trees have been well studied and widely used in knowledge discovery and decision support systems. Although the problem of finding an optimal decision tree has received attention, it is a hard optimization problem. Here we propose utilizing a genetic algorithm to improve on the finding of ap-propriate decision trees. We present a method to encode/decode a decision tree to/from a chromo-some where genetic operators can be applied. Theo-retical properties of decision trees, encoded chro-mosomes, and fitness functions are presented.*

Keywords: Binary decision tree, Genetic Algorithm.

1 Introduction

Decision trees approximate discrete-valued target functions as trees and are a widely used practical method for inductive inference [1]. Decision trees have prospered in knowledge discovery and decision support systems because of their natural and intuitive paradigm to classify a pattern through a sequence of questions. Algorithms for constructing decision trees such as ID3 [1,2,3] often use heuristics to find a shorter tree. Finding the shortest decision tree is a hard optimization problem [4,5]. Here we propose a method using genetic algorithms which use an optimization technique based on natural evolution [1,2,6,7].

Several attempts to construct near-optimal decision trees appear in the literature (see the extensive survey [8]). Gehrke et al developed a bootstrapped optimistic algorithm for decision tree construction [9]. For continuous attribute data, Zhao and Shirasaka [10] suggested an evolutionary design, Bennett and Blue [11] proposed an *extreme point tabu search* algorithm, and Pajunen and girolami [12] exploited *linear ICA* to construct binary decision trees. Here we consider constructing binary attribute decision trees.

Genetic algorithms (GAs) have been used to find near optimal decision trees in several ways. GAs were used to select attributes to be used to construct decision trees in a hybrid or preprocessing manner [13,14, 15]. Other recent work applied GAs directly to decision trees [16,17], where an attribute can occur more than once in the path of a decision tree. In this paper, we propose an alternate method.

The rest of the paper is organized as follows. Section 2 reviews decision trees. Section 3 presents the encoding/decoding decision trees to/from chromosomes, genetic operators like mutation and crossover, fitness functions and their analysis. Finally, section 4 concludes this work.

2 Preliminary: Decision Trees

A decision tree is a rooted tree T that consists of internal nodes representing attributes, leaf nodes representing labels, and edges representing the attributes' possible values. Decision trees classify instances by traversing from root node to leaf node. The classification process starts from the root node of a decision tree, tests the attribute specified by this node, and then moves down the tree branch according to the attribute value given.

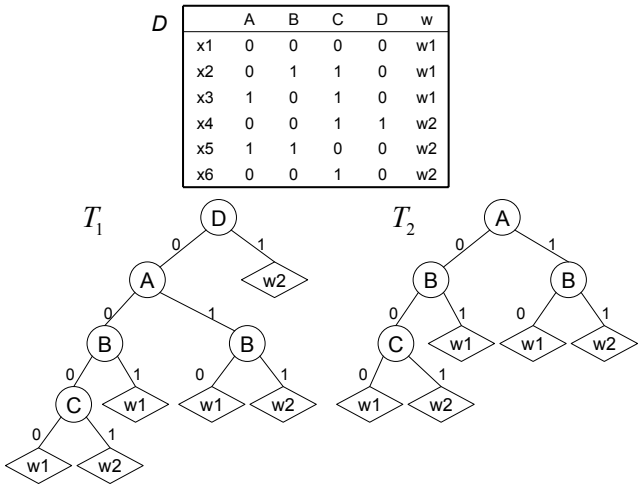

Figure 1. Decision trees consistent with D: T_1 by ID-3 and T_2 by GA.

Algorithms for decision tree construction take a set of training instances D as input and output a learned discrete-valued target function in the form of a tree. Instances are represented by attribute-value pairs where each attribute can have a small number of possible disjoint values. Here we consider only binary attributes. Hence, the input set has n instances where each instance x_i consists of d ordered binary

attributes and a target value which is one of c states of nature, w. Consider a sample input database D where $n = 6$, $d = 4$, $c = 2$, and $w = \{w_1, w_2\}$.

Among numerous decision trees that are consistent with the training database of instances, Fig 1 shows two of them. All instances $x = \{x_1, \ldots, x_6\}$ are classified correctly by both decision trees T_1 and T_2. However, an unknown instance <0, 0, 0, 1, ?>, which is not in the training set, is classified differently by the two decision trees; T_1 classifies the instance as w_2 whereas T_2 classifies it as w_1. This inductive inference is a fundamental problem in machine learning. Albeit controversial, many decision-tree building algorithms such as ID3 [3] prefer smaller trees (*Occam's razor)*. The shorter the tree, the fewer the number of questions required. Finding a smallest decision tree is an *NP-complete* problem [4,5] and we present a near optimal method using GAs.
Here are the important properties of a binary decision tree:

Property 1: The size of a decision tree with l leaves is $2l - 1$.

Property 2: The lower and upper bounds of l are c and n;

$$c \leq l \leq n.$$

If D represents a c-class classification problem, the number of leaves in the corresponding trees must be at least c in the best cases. In the worst cases, the number of leaves will be the size of D with each instance corresponding to a leaf. Consider a full binary decision tree T^f_3 where each path from the root to a leaf contains all the attributes exactly once as exemplified in Fig 2. There are 2^d leaves where d is the height of the full decision tree. Note that real decision trees can be sparse because some internal nodes can be leaves as long as they are homogeneous.

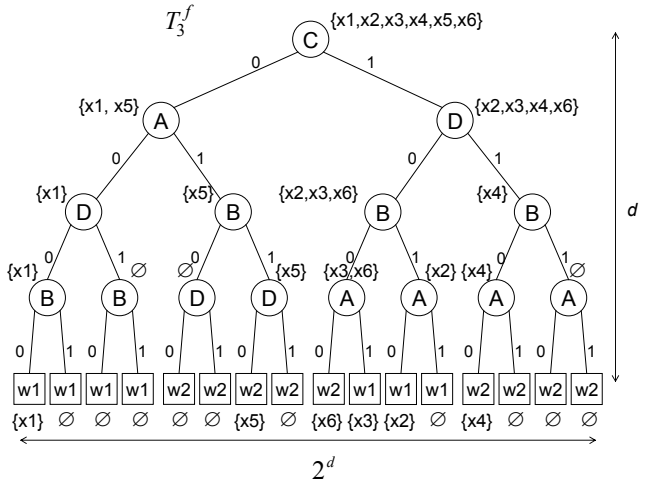

Figure 2. A sample full binary decision tree structure.

Branches represent the attributes' possible values, left branches having values of 0 and right branch values of 1. For simplicity we omit the value labels in later figures.

3 Algorithm

Genetic algorithms can provide good solutions to many optimization problems [6,7]. They are based on natural processes of evolution and the survival-of-the-fittest idea. In order to use the genetic algorithm process, one must define at least the following four steps: encoding, genetic operators such as mutation and crossover, decoding, and fitness function.

3.1 Encoding

For genetic algorithms to construct decision trees the decision trees must be encoded so that the genetic operators, such as mutation and crossover, can be applied. We illustrate and describe the process by considering the full binary decision tree in Fig 2.

Let $A = \{a_1, a_2, \ldots, a_d\}$ be the attribute list. Consider the full binary decision tree T^f_3, $A = \{A, B, C, D\}$. Graphically speaking, the encoding process converts the attribute names in T^f_3 to the index of the attribute according to A recursively starting from the root as illustrated in Fig. 3. For example, the root is C and its index in A is 3. Recursively, for each subtree, update A to A-{C} attribute list. The possible integer values at a node in the i-th level in the encoded decision tree T^e are from 1 to $d - i + 1$. Finally, take the breadth-first traversal to generate the chromosome string. For T^f_3 the string is given in Fig 3 (b).

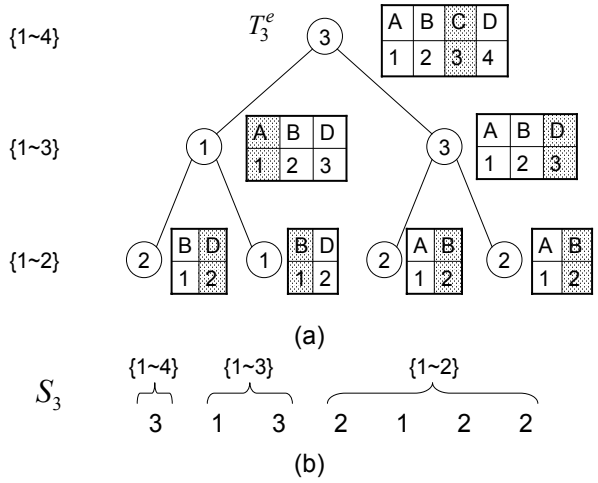

Figure 3. A Encoding and decoding schema: (a) encoded tree for in Fig 2 and (b) its chromosome attribute-selection scheduling string.

$T_1$and T_2 in Fig 1 are encoded into <4, 1, *, 1, *, *, *> and <1, 1, 1, 1, *, *, *>, respectively, where * can be any number within the restricted bounds. Let us call this a chromosome attribute-selection scheduling string, S, where genetic operators can be applied. Properties of S include:

Property 3: The parent position of position i is $\lfloor i/2 \rfloor$, except for $i = 1$, the root.

Property 4: The left and right child positions of position i are $2i$ and $2i + 1$, respectively, if $i \leq 2^{d-2} - 1$; otherwise, there are no children.

Property 5: The length of the S's is exponential in d:

$$|S| = 2^{d-1} - 1 .$$

Property 6: Possible integer values at position i are 1 to $d - \lceil \log(i+1) \rceil - 1$: $s_i \in \{1,..,d - \lceil \log(i+1) \rceil - 1\}$.

Property 7: The number of possible S, $|\pi(S)|$, is exponential: $|\pi(S)| = \prod_{i=1}^{|S|} \left(d - \lceil \log(i+1) \rceil - 1\right)^i$ or equivalently $|\pi(S)| = \prod_{i=1}^{d-1} (d - x + 1)^{2^{i-1}}$.

The lower and upper bounds of $|\pi(S)|$ are $o(2^{|S|})$ and $\Omega(d^{|S|})$, respectively.

3.2 Genetic Operators

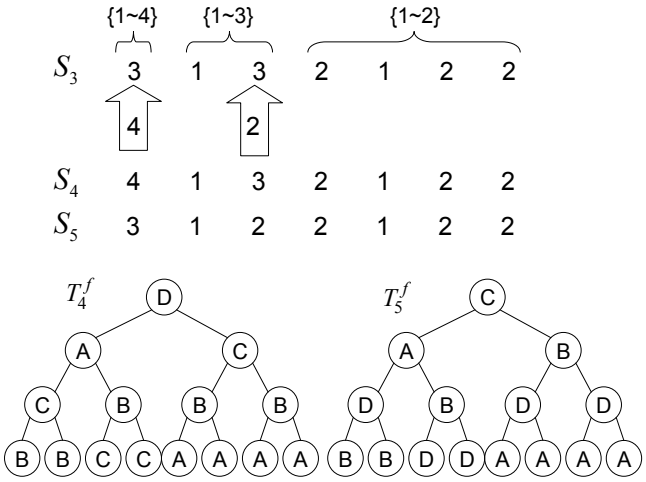

Figure 4. Mutation Operator.

Two of the most common genetic operators are mutation and crossover. The mutation operator is defined as changing the value of a certain position in a string to one of the possible values in the range. We illustrate the mutation process on the attribute selection scheduling string s_3 <3, 1, 3, 2, 1, 2, 2> in Fig 4. If a mutation occurs in the first position and changes the value to 4, which is in the range {1,..,4}, T^f_4 is generated. If a mutation happens in the third position and changes the value to 2, which is in the range {1,.. ,3}, then T^f_5 is generated. As long as the changed value is within the allowed range, the resulting new string always generates a valid full binary decision tree.

Consider S_1 <4, 1, 3, 1, 2, 2, 1>. A full binary decision tree is built according to this schedule. The final decision tree for S_1 will be T_1 <4, 1, *, 1, *, *, *> in Fig 1. There are $3 \times 2 \times 2 \times 2 = 24$ equivalent schedules that produce T_1. Therefore, for the mutation to be effective we apply the mutation operators only to those positions that are not *.

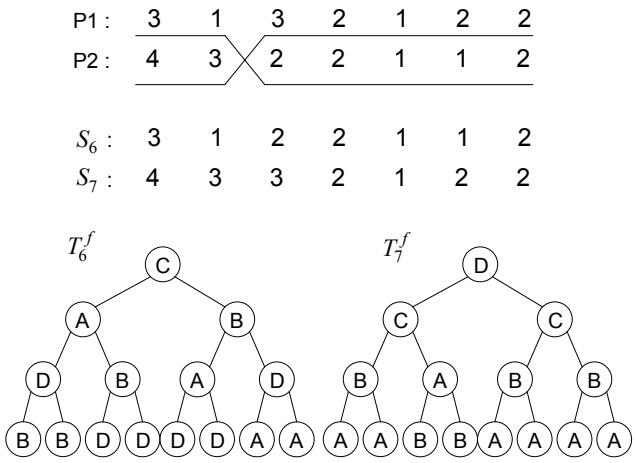

Figure 5. Crossover Operator**.**

We illustrate the crossover operator in Fig 5. Consider the two parent attribute selection scheduling strings, P1 and P2, in Fig 5. After randomly selecting a split point, the first part of P1 and the last part of P2 contribute to yield a child string s_6. Reversing the crossover produces a second Child s_7. The resulting full binary decision trees for these two children are T_6 and T_7, respectively.

3.3 Decoding

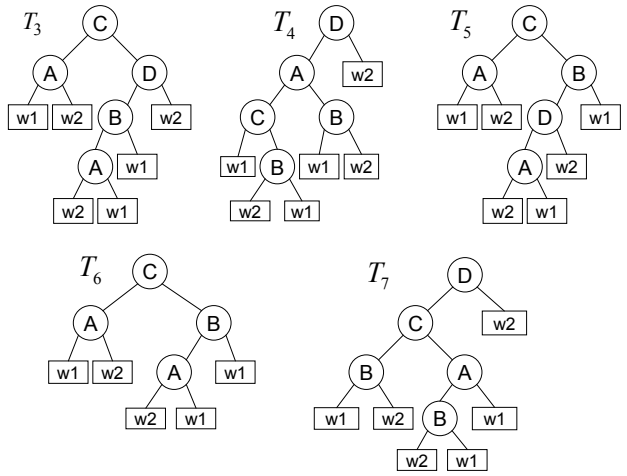

Figure 6. Decoded decision trees.

Decoding is the reverse of the encoding process in Fig 3. Starting from the root node, we place the attribute according to the chromosome schedule S which contains the index values of attribute list A. When an attribute a is selected, D is divided into left and right branches D_L and D_R. D_L consists of all the x_i having a value of 0 and D_R consists of

all the x_i having a value of 1. For each pair of sub-trees we repeat the process recursively with the new attribute list $A = A - \{a\}$. When a node becomes homogeneous, i.e., all class values in D are the same, we label the leaf. Fig 7 displays the decision trees from s_3, s_4, s_5 , s_6, and s_7, respectively.

Sometimes a chromosome introduces mutants. For instance, consider a chromosome s_8 <3, 3, 2, 1, 1, 1, 2> which results T_8, in Fig 7. The ⊗ occurs when D at the node attribute a has non-homogeneous labels but the a column in D has either all 0's or all 1's. In other words, the a attribute provides no *information gain*. We refer to such a decision tree as a mutant tree for two reasons. First, what values should we put in ⊗? The label may be chosen at random but D provides no clue. Second, if we allow entering a value for ⊗, it may violate property 2; the number of leaves l may exceed n. Indeed, T_8 behaves identically to T_6 with respect to D. Thus, mutant decision trees will not be chosen as the fittest trees according to the fitness functions.

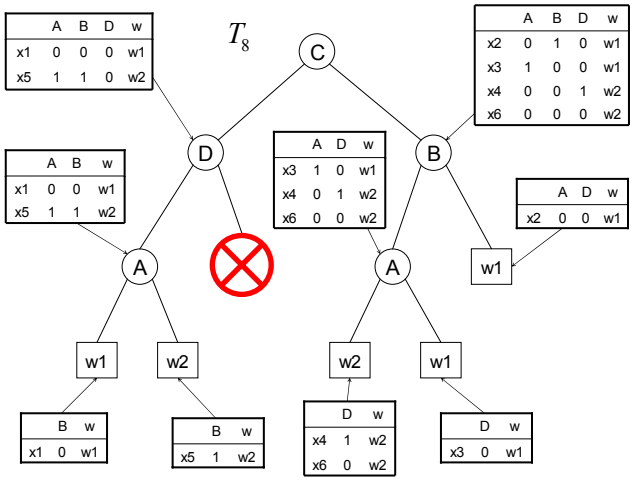

Figure 7. Mutant binary decision tree.

3.4 Fitness functions

Each attribute selection scheduling string S must be evaluated according to a fitness function. We consider two cases: in the first case D contains no contradicting instances and d is a small finite number, in the other case d is very large. Contradicting instances are those whose attribute values are identical but their target values are different. The case with small d and no contradicting instances has application to network function representation [18].

Theorem 1: Every attribute selection scheduling string S produces a consistent full binary decision tree as long as the set of instances contains no contradicting instances.

Proof: Suppose one adds nodes at the $d - l$ level with the remaining one attribute to the full binary decision tree. One also adds the leaves with target values in the set of instances accordingly. If the target value is not available in the training set, add a random target value. This becomes a complete truth table as the number of leaves are 2^d. The predicted value by the decision tree is always consistent with the actual target value. ■

Since the databases we consider contain no contradicting instances, one obvious fitness function is the size of the tree; the fitness function f_s is the size, the total number of nodes, e.g., $f_s(T_1) = 11$ and $f_s(T_2) = 9$. Here, however, we use a different fitness function f_d, i.e., the sum of the depth of the leaf nodes for each instance. This is equivalent to the sum of questions to be asked for each instance, e.g., $f_d(T_1) = 18$ and $f_d(T_2) = f_d(T_6) = 15$ as shown in Table 1. This requires the assumption that each instance in D has equal prior probability. Both of these evaluation functions suggest that T_2 and T_6 are better than T_1 which is produced by the *ID3* algorithm.

Table 1. Training instances and their depths in decision trees.

X	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
x1	4	3	2	3	2	2	3	3
x2	3	2	3	4	2	2	4	2
x3	3	2	4	3	4	3	3	3
x4	1	3	2	1	3	3	1	3
x5	3	2	2	3	2	2	3	3
x6	4	3	4	4	4	3	4	3
sum	18	15	17	18	17	15	18	17

With two decision trees of the same size ($2l - 1$) where l is the number of leaves, the number of questions to be asked could be different by theorem 2.

Theorem 2: There exist T_x and T_y such that $f_s(T_x) = f_s(T_y)$ and $f_d(T_x) < f_d(T_y)$.

Proof: Let T_x be a balanced binary tree and T_y be a skewed or linear binary tree. Then $f_d(T_x) = \Theta(n\log n)$ whereas $f_d(T_y) = \Theta(n^2) = \Theta(\sum_{i=1}^{n} i)$. ■

Corollary 1: $n\log c \le f_d(T_x) \le \sum_{i=1}^{n} i - 1$ where n is the number of instances.

Proof: By property 2, in the best case the decision tree will have $l = c$ leaves. In the best case, the tree is completely balanced with height log c. Thus the lower bound for $f_d(T_x)$ is $n$log c. In the worst case, the number of leaves is n by property 2 and the decision tree forms a skewed or linear binary tree. Thus the upper bound is $\sum_{i=1}^{n}(i-1)$. ■

If $f_d(T_x) < n\log c$, T_x classifies only a subset of classes, never classifying one or more classes. Regardless of the size, this tree should not be selected. If $f_d(T_x) > \sum_{i=1}^{n} i - 1$, there is a mutant node and T_x can be shortened.

When d is large, the size of the chromosome will explode by the property 3. In this case we must limit the height of the decision tree. The chromosome has a finite length and guides to select attributes up to a certain height. Since $n \ll 2^d$ typically, a good choice of the height of the decision tree is $\log n$, i.e., $|s| \approx n$.

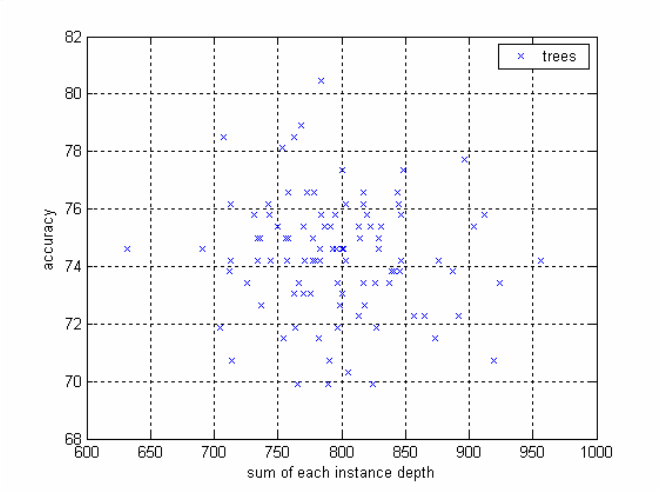

Figure 8. Decision trees with respect to f_d and accuracy f_a.

We randomly generated a 26 binary attribute training database and limited the tree height to 8. Fig 8 shows 100 genetically generated decision trees after 100 generations in respect to their $f_d(T_x)$ and accuracy $f_a(T_x)$ on the training examples.

4 Conclusions

In this paper, we reviewed binary decision trees and utilized genetic algorithms to build them. By limiting the tree's height the presented method guarantees finding a better or equal decision tree than the best known algorithms since such trees can be put in the initial population.

Methods that apply GAs directly to decision trees [16,17] can yield subtrees that are never visited as shown in Fig 9. After mutation operator in 'O' node in T_9, T_{11} has a dashed subtree that is never visited. After crossover between T_9 and T_{10}, the child trees T_{12} and T_{13} also have dashed subtrees. These unnecessary subtrees occur whenever an attribute occurs more than once in the path of a decision tree. However, by encoding the decision tree, this problem never occurs as in Fig 10.

When d is large, limiting the height is recommended. However, pre-order depth first traversal rather than the breadth first traversal is an answer as decision trees are often sparse as shown in Fig 10. Mutation shown in this paper is still valid in pre-order depth first traversal representation but crossover may cause a mutant when two subtrees to be switched are at different levels. The index number of a node may exceed the limit due to a crossover. Thus mutation after crossover is inevitable. Analyzing and implementing the depth first traversal of encoded decision tree remains ongoing work. Encoding and decoding non-binary decision trees where different attributes have different possible values is an open problem.

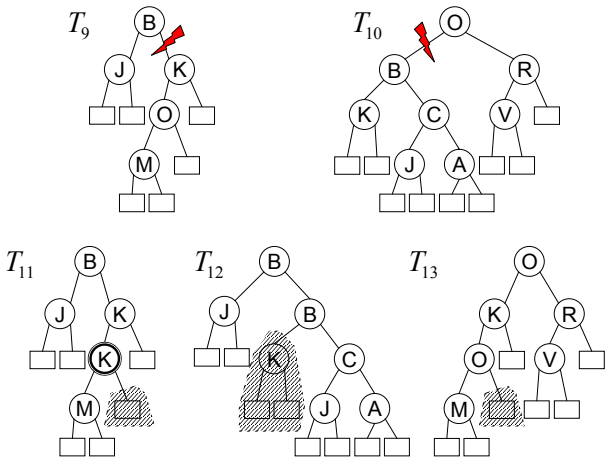

Figure 9. Direct GA on decision trees.

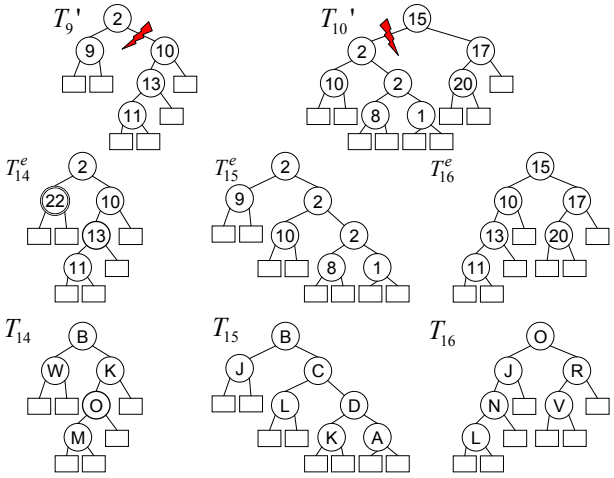

Figure 10. Pre-order depth first traversal for decision trees.

5 References

[1] Mitchell, T. M., "Machine Learning", McGraw-hill, 1997

[2] Duda, R. O., Hart, P. E., and Stork, D. G., Pattern Classification, 2nd ed. Wiley interscience, 2001

[3] Quinlan, J. R., Induction of decision trees, Machine Learning, 1(1), 81-106

[4] L. Hyafil and R. L. Rivest, " Constructing optimal binary decision trees is NP-complete ," Information Processing Letters, vol. 5, No. 1, 15-17, 1976.

[5] Bodlaender, L.H. and Zantema H., Finding Small Equivalent Decision Trees is Hard, International Journal

of Foundations of Computer Science, Vol. 11, No. 2 World Scientific Publishing, 2000, pp 343-354

[6] Goldberg D. L., "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, 1989

[7] Mitchell, M., "An Introduction to Genetic Algorithms", Massachusetts Institute of Technology, 1996.

[8] Safavian, S.R. and Landgrebe, D., A survey of decision tree classifier methodology, IEEE Transactions on Systems, Man and Cybernetics, vol 21, num 3, pp 660-674, 1991

[9] Gehrke, J., Ganti, V., Ramakrishnan R., and Loh, W., BOAT-Optimistic Decision Tree Construction, in Proc. of the ACM SIGMOD Conference on Management of Data, 1999, p169-180

[10] Zhao, Q. and Shirasaka, M., A Study on Evolutionary Design of Binary Decision Trees, in Proceedings of the Congress on Evolutionary Computation, vol 3, IEEE, 1999, pp 1988-1993

[11] Bennett , K. and Blue, J., Optimal decision trees, Tech. Rpt. No. 214 Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, New York., 1996

[12] Pajunen, P. and Girolami, M., "Implementing decisions in binary decision trees using independent component analysis", in Proc. of ICA, 2000, pages 477-481.

[13] Kyoung Min Kim, Joong Jo Park, Myung Hyun Song, In Cheol Kim, and Ching Y. Suen, Binary Decision Tree Using Genetic Algorithm for Recognizing Defect Patterns of Cold Mill Strip, LNCS vol 3060, Springer, 2004, pp1611-3349

[14] Teeuwsen, S.P., Erlich, I., El-Sharkawi, M.A., and Bachmann, U., Genetic algorithm and decision tree-based oscillatory stability assessment, IEEE Transactions on Power Systems, Vol 21, Issue 2, May 2006 p 746-753

[15] Bala, J., Huang, J., Vafaie, H., DeJong, K., and Wechsler, H., Hybrid learning using genetic algorithms and decision tress for pattern classification. in Proc. of the 14th International Joint Conference on Artificial Intelligence, Montreal, Canada, 1995, pp. 719-724.

[16] Papagelis, A. and Kalles, D., GA Tree: genetically evolved decision trees, in Procc of 12th IEEE International Conference on Tools with Artificial Intelligence, 2000, p 203-206

[17] Fu, Z., An Innovative GA-Based Decision Tree Classifier in Large Scale Data Mining, LNCS Vol. 1704, Springer, 1999, pp 348 – 353.

[18] Martinez, T. R. and Campbell, D. M., A Self-Organizing Binary Decision Tree For Incrementally Defined Rule Based Systems, Systems, IEEE Systems, Man, and Cybernetics, vol. 21, No. 5, pp.1231-1238, 1991.