JOURNAL OF PATTERN RECOGNITION RESEARCH 1 (2009) 1-13

Received Jul 27, 2008. Accepted Feb 6, 2009.

A Genetic Algorithm for Constructing Compact Binary Decision Trees

Sung-Hyuk Cha *scha@pace.edu*
Charles Tappert *ctappert@pace.edu*
Computer Science Department, Pace University
861 Bedford Road, Pleasantville, New York, 10570, USA

Abstract

Tree-based classifiers are important in pattern recognition and have been well studied. 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 compact, near-optimal decision trees. We present a method to encode and decode a decision tree to and from a chromosome where genetic operators such as mutation and crossover can be applied. Theoretical properties of decision trees, encoded chromosomes, and fitness functions are presented.

Keywords: Binary Decision Tree, Genetic Algorithm.

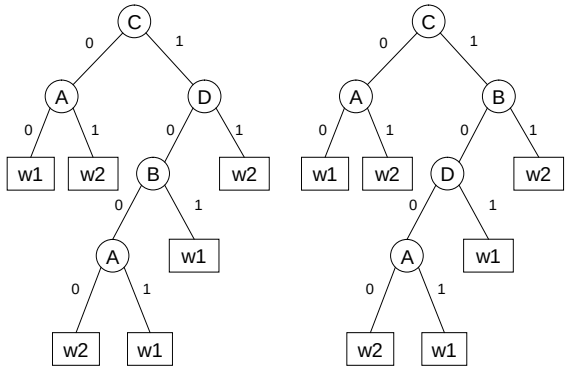
1. Introduction

Decision trees have been well studied and widely used in knowledge discovery and decision support systems. Here we are concerned with decision trees for classification where the leaves represent classifications and the branches represent feature-based splits that lead to the classifications. These 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 [1-3], often use heuristics that tend to find short trees. Finding the shortest decision tree is a hard optimization problem [4, 5].

Attempts to construct short, near-optimal decision trees have been described (see the extensive survey [6]). Gehrke et al developed a bootstrapped optimistic algorithm for decision tree construction [7]. For continuous attribute data, Zhao and Shirasaka [8] suggested an evolutionary design, Bennett and Blue [9] proposed an extreme *point tabu search* algorithm, and Pajunen and Girolami [10] exploited linear *independent component analysis* (ICA) to construct binary decision trees.

Genetic algorithms (GAs) use an optimization technique based on natural evolution [1, 2, 11, 12]. GAs have been used to find near-optimal decision trees in twofold. On the one hand, they were used to select attributes to be used to construct decision trees in a hybrid or preprocessing manner [13-15]. On the other hand, they were applied directly to decision trees [16, 17]. A problem that arises with this approach is that an attribute may appear more than once in the path of the tree. In this paper, we describe an alternate method of constructing near-optimal binary decision trees proposed succinctly in [18].

In order to utilize genetic algorithms, decision trees must be represented as chromosomes where genetic operators such as mutation and crossover can be applied. The main contribution of this paper is proposing a new scheme to encode and decode a decision tree to and from a chromosome. The remainder of the paper is organized as follows. Section 2 reviews decision trees and defines a new function denoted as $d\Delta$ to describe the complexity. Section 3 presents the encoding/decoding



(a) T_1 consistent to D (b) T_x inconsistent to D

Fig. 1: Decision trees consistent and inconsistent with D.

decision trees to/from chromosomes which stems from $d\Delta$, genetic operators like mutation and crossover, fitness functions and their analysis. Finally, Section 4 concludes this work.

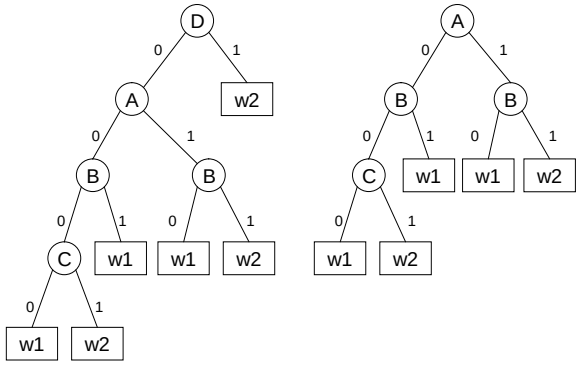
2. Preliminary

Let D be a set of labeled training data, a database of instances represented by attribute-value pairs where each attribute can have a small number of possible disjoint values. Here we consider only binary attributes. Hence, D 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. The following sample database D where $n = 6$, $d = 4$, $c = 2$, and $w = \{w1, w2\}$ will be used for illustration throughout the rest of this paper.

$$D = \left| \begin{array}{cccccc} & A & B & C & D & w \\ x_1 & 0 & 0 & 0 & 0 & w_1 \\ x_2 & 0 & 1 & 1 & 0 & w_1 \\ x_3 & 1 & 0 & 1 & 0 & w_1 \\ x_4 & 0 & 0 & 1 & 1 & w_2 \\ x_5 & 1 & 1 & 0 & 0 & w_2 \\ x_6 & 0 & 0 & 1 & 0 & w_2 \end{array} \right|$$

Algorithms to construct a decision tree take a set of training instances D as input and output a learned discrete-valued target function in the form of a tree. 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. Branches represent the attributes possible values and in binary decision trees, left branches have values of 0 and right branches have values of 1 as shown in Fig. 1. For simplicity we omit the value labels in some later figures. A decision tree represents a disjunction of conjunctions. In Fig. 1(a), for example, T_1 represents the w_1 and w_2 states as $(\neg C \wedge \neg A) \vee (C \wedge \neg D \wedge \neg B \wedge A) \vee (C \wedge \neg D \wedge B)$ and $(\neg C \wedge A) \vee (C \wedge \neg D \wedge \neg B \wedge \neg A) \vee (C \wedge D)$, respectively. Each conjunction corresponds to a path from the root to a leaf.

A decision tree based on a database D with c number of classes is a c-class classification problem. 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 at this node, and then moves down the tree branch according to the attribute value given. Fig. 1 shows two decision trees, T_1 and T_x. The decision tree T_1 is said to be a *consistent* decision tree because it is consistent



(a) T_2 with 11 nodes (b) T_3 with 9 nodes

Fig. 2: Two decision trees consistent with D: (a) by ID-3 and (b) by GA.

with all instances in D. However, the decision tree T_x is *inconsistent* with D because x_2's class is actually $w1$ in D whereas T_x classifies it as $w2$.

There are two 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 for a consistent binary decision tree are c and n: $c \leq l \leq n$.*

The number of leaves in a consistent decision tree 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 unique leaf, e.g., T_1 and T_2.

2.1 Occams Razor and ID3

Among numerous decision trees that are consistent with the training database of instances, Fig. 2 shows two of them. All instances $x = \{x_1, \ldots, x_6\}$ are classified correctly by both decision trees T_2 and T_3. However, an unknown instance $\langle 0, 0, 0, 1, ? \rangle$, which is not in the training set, D is classified differently by the two decision trees; T_2 classifies the instance as w_2 whereas T_3 classifies it as w_1. This inductive inference is a fundamental problem in machine learning. The *minimum description length principle* formalized from *Occam's Razor* [19] is a very important concept in machine learning theory [1, 2]. Albeit controversial, many decision-tree building algorithms such as ID3 [3] prefer smaller (more compact, shorter depth, fewer nodes) trees and thus the instance $\langle 0, 0, 0, 1, ? \rangle$ is preferably classified as w_2 by T_3 because T_3 is shorter than T_2. In other words, T_3 has a simpler description than T_2. The shorter the tree, the fewer the number of questions required to classify instances.

Based on *Occam's Razor*, Ross Quinlan proposed a heuristic that tends to find smaller decision trees [3]. The algorithm is called ID3 (*Iterative Dichotomizer 3*) and it utilizes the *Entropy* which is a measure of homogeneity of examples as defined in the equation 1.

$$Entropy(D) = -\sum_{x \in w = \{w1, w2\}} P(x) \log P(x) \tag{1}$$

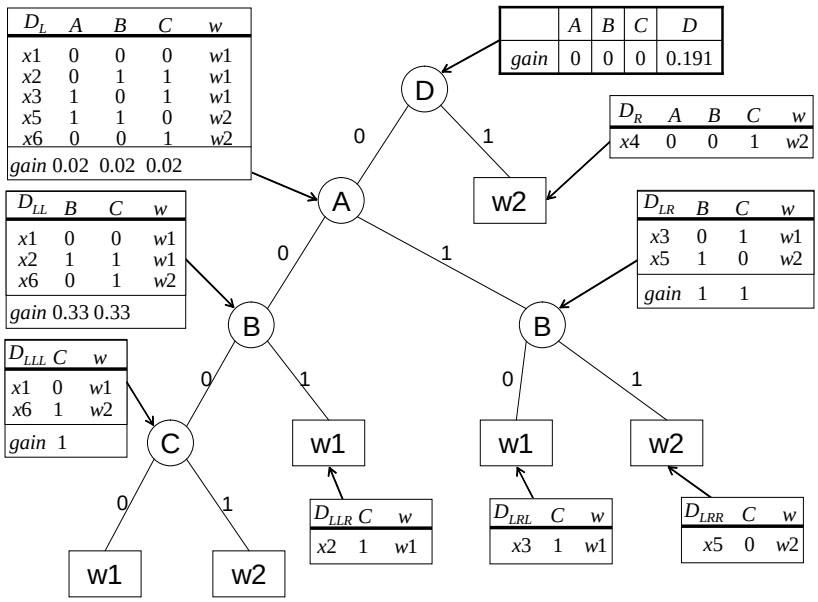


Fig. 3: Illustration of the ID3 algorithm.

Information gain or simply *gain* is defined in terms of *Entropy* where X is one of attributes in D. When all attributes are binary type, the gain can be defined as in the equation 2.

$$gain(D, X) = Entropy(D) - \left(\frac{|D_L|}{|D|} Entropy(D_L) + \frac{|D_R|}{|D|} Entropy(D_R) \right) \tag{2}$$

The ID3 algorithm first selects the attribute whose *gain* is the maximum as a root node. For all subtrees of the root node, it finds the next attribute whose gain is the maximum iteratively. Fig. 3 illustrates the ID3 algorithm. Starting with the root node, it evaluates all attributes in the database D. Since the attribute D has the highest gain, the attribute D is selected as a root node. Then it partitions the database D into two sub databases: D_L and D_R. For each sub-database, it calculates the gain. As a result, T_2 decision tree is built. However, as is apparent from Fig. 2 the ID3 algorithm does not necessarily find the smallest decision tree.

2.2 Complexity of $d\Delta$ Function

To the extent that smaller trees are preferred, it becomes interesting to find a smallest decision tree. Finding a smallest decision tree is an NP-complete problem though [4, 5]. So as to comprehend the complexity of finding a smallest decision tree, consider a full binary decision tree, T_1^f (the superscript f denotes full), where each path from the root to a leaf contains all the attributes exactly once as exemplified in Fig. 4 . There are $2d$ leaves where $d+1$ is the height of the full decision tree.

In order to build a consistent full binary tree, one may choose any attribute as a root node, e.g., there are four choices in Fig. 4 . In the second level nodes, one can choose any attribute that is not in the root node. For example, in Fig. 4 there are $2 \times 2 \times 2 \times 2 \times 3 \times 3 \times 4$ possible full binary trees. We denote the number of possible full binary tree with d attributes as $d\Delta$.

Definition 1 *The number of possible full binary tree with d attributes is formally defined as*

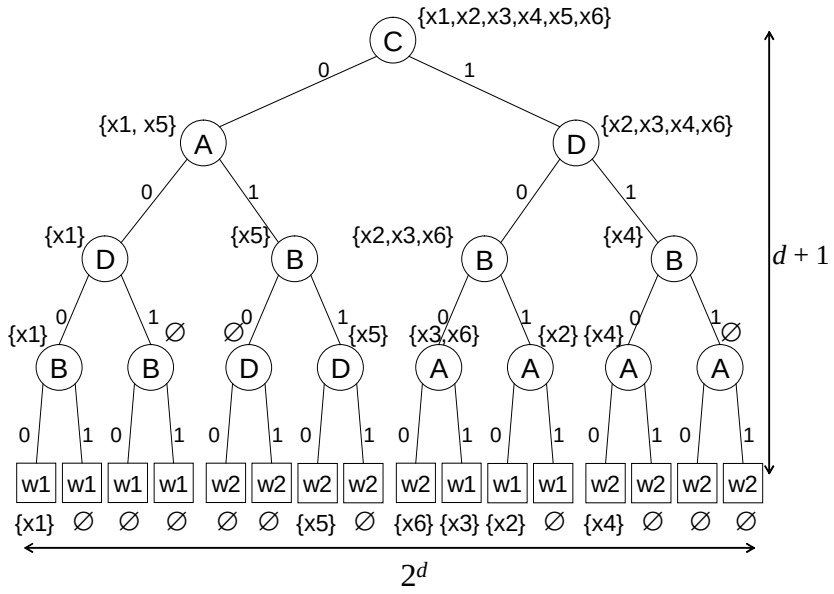


Fig. 4: A sample full binary decision tree structure T_1^f.

Table 1: Comparison of $d\Delta$ and $d!$ functions.

d	$d\Delta$		$d!$
1	$1\Delta = 1$	$= 1$	$1! = 1$
2	$2\Delta = 1 \times 1 \times 2$	$= 2$	$2! = 2$
3	$3\Delta = 1 \times 1 \times 1 \times 1 \times 2 \times 2 \times 3$	$= 12$	$3! = 6$
4	$4\Delta = 2 \times 2 \times 2 \times 2 \times 3 \times 3 \times 4$	$= 576$	$4! = 24$
5	$5\Delta = 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 3 \times 3 \times 3 \times 3 \times 4 \times 4 \times 5$	$= 1658880$	$5! = 120$
6	$6\Delta = 2^{16} \times 3^8 \times 4^4 \times 5^2 \times 6$	$= 16511297126400$	$6! = 720$
7	$7\Delta = 2^{32} \times 3^{16} \times 4^8 \times 5^4 \times 6^2 \times 7$	$= 1.9084e + 027$	$7! = 5040$
8	$8\Delta = 2^{64} \times 3^{32} \times 4^{16} \times 5^8 \times 6^4 \times 7^2 \times 8$	$= 2.9135e + 055$	$8! = 40320$
9	$9\Delta = 2^{128} \times 3^{64} \times 4^{32} \times 5^{16} \times 6^8 \times 7^4 \times 8^2 \times 9$	$= 7.6395e + 111$	$9! = 362880$
10	$10\Delta = 2^{256} \times 3^{128} \times 4^{64} \times 5^{32} \times 6^{16} \times 7^8 \times 84 \times 9^2 \times 10$	$= 5.8362e + 224$	$10! = 3628800$

$$d\Delta = \prod_{i=1}^{d} i^{2^{d-i}}. \tag{3}$$

As illustrated in Table 1, as d grows, the function $d\Delta$ grows faster than all polynomial, exponential, and even factorial functions. The *factorial* function $d!$ is the product of all positive integers less than or equal to d and many *combinatorial optimization* problems have the complexity of $O(d!)$ search space. The search space of full binary decision trees is much larger, i.e., $d\Delta = \Omega(d!)$. Lower and upper bounds of $d\Delta$ are $\Omega(2^{2^d})$ and $o(d^{2^d})$. Note that real decision trees, such as T_1 in Fig. 1 (a), can be sparse because some internal nodes can be leaves as long as they are homogeneous. Hence, the search space for finding a shortest binary decision tree can be smaller than $d\Delta$.

3. Genetic Algorithm for Binary Decision Tree Construction

Genetic algorithms can provide good solutions to many optimization problems [11, 12]. They are based on natural processes of evolution and the survival-of-the-fittest concept. In order to use the

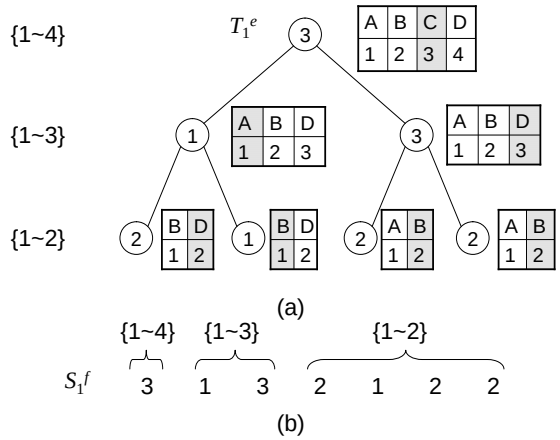


Fig. 5: Encoding and decoding schema: (a) encoded tree for T_1^f in Fig. 4 and (b) its chromosome attribute-selection scheduling string.

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 genetic operators, such as mutation and crossover, can be applied. Let $A = \{a_1, a_2, \ldots, a_d\}$ be the ordered attribute list. We illustrate and describe the process by considering the full binary decision tree T_1^f in Fig. 4, where $A = \{A, B, C, D\}$.

Graphically speaking, the encoding process converts the attribute names in T_1^f into the index of the attribute according to the ordered attribute list, A, recursively, starting from the root as depicted in Fig. 5. For example, the root is C and its index in A is 3. Recursively, for each sub-tree, update A to $A - \{C\} = \{A, B, D\}$ 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 S, which stems from $d\Delta$ function. For T_1^f the chromosome string S_1 is given in Fig. 5 (b).

T_1 and T_3 in Fig. 1 is encoded into $S_1 = \langle 3, 1, 3, *, *, 2, * \rangle$ where $*$ can be any number within the restricted bounds. T_2 and T_3 in Fig. 2 are encoded into $S_2 = \langle 4, 1, *, 1, 1, *, * \rangle$ and $S_3 = \langle 1, 1, 1, 1, *, *, * \rangle$, respectively. 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 : $|S| = 2^{d-2} - 1$.*

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

Property 6 provides the restricted bounds for each position of S.

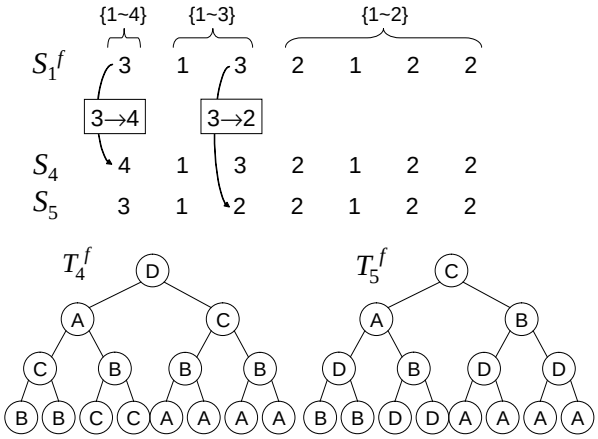


Fig. 6: Mutation operator.

Property 7 *The number of possible S, $|\pi(S)|$, is $d\Delta$ or* $|\pi(S)| = \prod_{i=1}^{|S|}(d - \lceil \log(i+1) \rceil - 1)^i$.

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

3.2 Genetic Operators

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 $S_1^f = \langle 3, 1, 3, 2, 1, 2, 2 \rangle$ in Fig. 6. If a mutation occurs in the first position and changes the value to 4, which is in the range $\{1, .., 4\}$, T_4^f 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_5^f 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 $S_x = \langle 4, 1, 3, 1, 1, 2, 1 \rangle$. A full binary decision tree is built according to this schedule. The final decision tree for S_x will be T_2 Fig. 2, i.e., $S_x = \langle 4, 1, *, 1, 1, *, * \rangle$. There are $3 \times 2 \times 2 = 12$ equivalent schedules that produce T_2. Therefore, for the mutation to be effective we apply the mutation operators only to those positions that are not $*$.

To illustrate the crossover operator, consider the two parent attribute selection scheduling strings, $P1$ and $P2$, in Fig. 7 . 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^f and T_7^f, respectively.

3.3 Decoding

Decoding is the reverse of the encoding process in Fig. 5. 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. 8 displays the decision trees from S_4, S_5, S_6, and S_7, respectively.

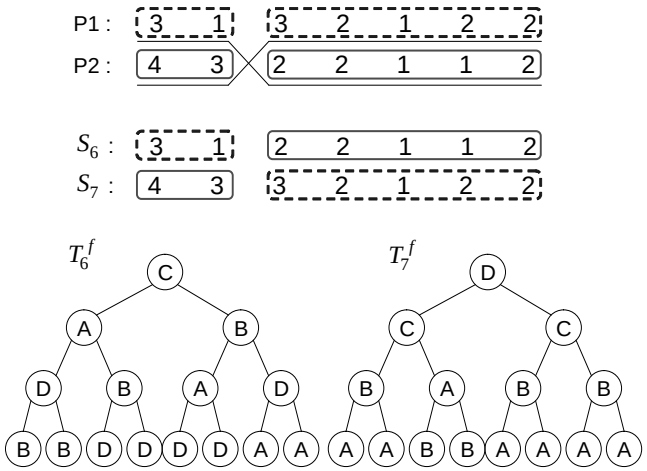


Fig. 7: Crossover operator.

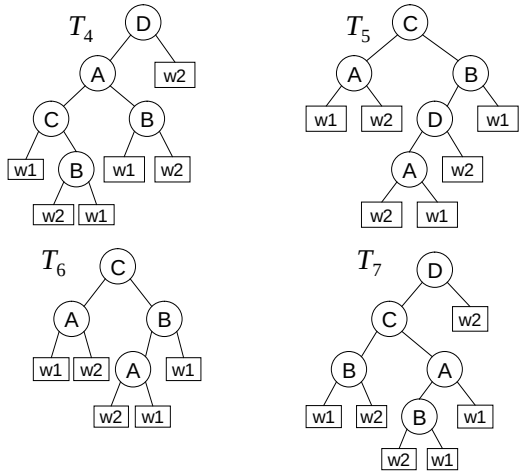


Fig. 8: Decoded decision trees.

Sometimes a chromosome introduces mutants. For instance, consider a chromosome S_8 $\langle 3, 3, 2, 1, 1, 1, 2\rangle$ which results T_8 in Fig. 9. The $\otimes$ 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 $\otimes$? The label may be chosen at random but D_L provides no clue. Second, if we allow entering a value for $\otimes$, it may violate the 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 presented in the later section.

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

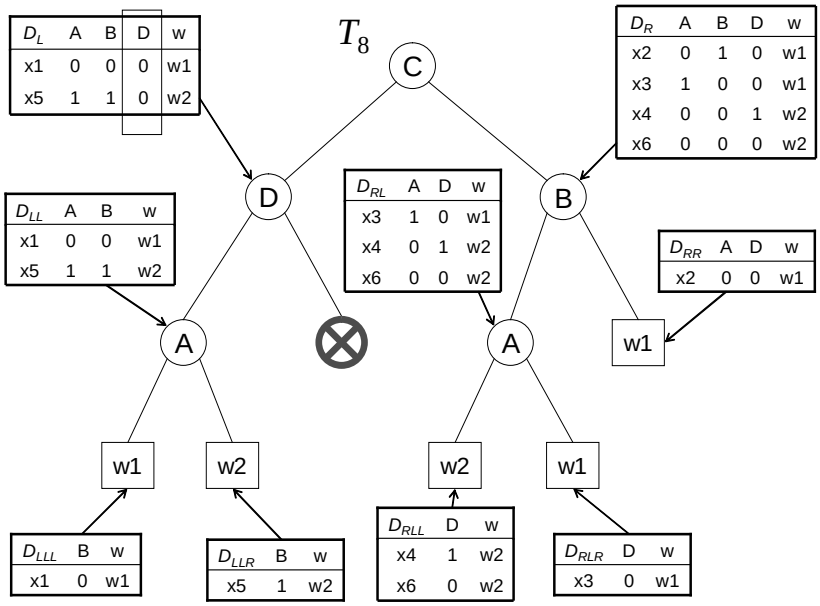


Fig. 9: Mutant binary decision tree.

identical but their target values are different. The case with small d and no contradicting instances has application to network function representation [20].

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 Every attribute selection scheduling string S produces a full binary decision tree, e.g., in Fig. 4. Since each instance in D corresponds to a certain branch, add 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 is 2^d. The predicted value by the decision tree is always consistent with the actual target value in D. If there are two instances with same attribute values but different classes, no consistent binary decision tree exists. □

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., $f_s(T_2) = 11$ and $f_s(T_3) = 9$. 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., $f_d(T_2) = 18$ and $f_d(T_3) = f_d(T_6) = 15$ as shown in Table 2. This requires the assumption that each instance in D has equal prior probability. Both of these evaluation functions suggest that T_3 and T_6 are better than T_2 which is produced by the ID3 algorithm.

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 $F_d(T_y) = \Theta(n^2) = \Theta(\sum_{i=1}^{n} i)$. □

Table 2: Training instances and their depths in decision trees.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
x_1	2	4	3	3	2	2	3	3
x_2	3	3	2	4	2	2	4	2
x_3	4	3	2	3	4	3	3	3
x_4	2	1	3	1	3	3	1	3
x_5	2	3	2	3	2	2	3	3
x_6	4	4	3	4	4	3	4	3
sum	17	18	15	18	17	15	18	17

The f_d fitness function prefers not only shorter decision trees to larger ones, but also balanced decision trees to skewed ones.

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

Proof By the 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 the 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. 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, $|S|$ will explode by the Property 5. 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 $h = \log n$, i.e., $|S| \approx n$. When a branch reaches a height h, the node is assigned to be a leaf with the class whose prior probability is the highest instead of choosing another attribute. When the height is limited, decision trees may be no longer consistent to D.

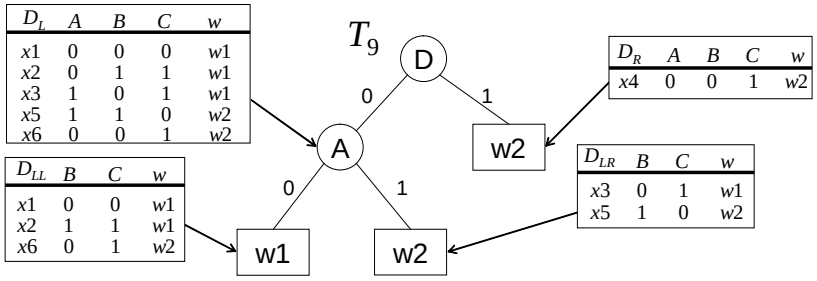
Suppose that the height is limited to 3; $h = 3$ in Fig. 3 case. Fig. 10 shows the pruned binary decision tree, T_9. In the 4th node in breadth first traversal order, instead of choosing B attribute in Fig. 3, $w1$ is labeled because the prior probability of $w1$ is higher than $w2$ in D_{LL}. When the prior probabilities are the same, either one can be chosen. Let f_a be the fitness function that represents the percentage of correctly classified instances by the decision tree. In Fig. 10, $\{x_1, x_2, x_4, x_5\}$ are correctly predicted by T_9 and hence $f_a(T_9) = 4/6$.

We randomly generated a 26 binary attribute training database and limited the tree height to 8. Fig. 11 shows the initial population of 100 binary decision trees generated by random chromosomes in respect to their $f_d(T_x)$ and accuracy $f_a(T_x)$ on the training examples. The size of decision trees and their accuracies on the training examples have no correlations.

4. Discussion

In this paper, we reviewed binary decision trees and demonstrated how to utilize genetic algorithms to find compact binary decision trees. 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. 12. After mutation operator in 'O' node in T_a, T_c has a dashed subtree that



D	A	B	C	D	w	Predicted by T_9
$x1$	0	0	0	0	$w1$	$w1$
$x2$	0	1	1	0	$w1$	$w1$
$x3$	1	0	1	0	$w1$	$w2$
$x4$	0	0	1	1	$w2$	$w2$
$x5$	1	1	0	0	$w2$	$w2$
$x6$	0	0	1	0	$w2$	$w1$

Fig. 10: Pruned decision tree at the height of 3.

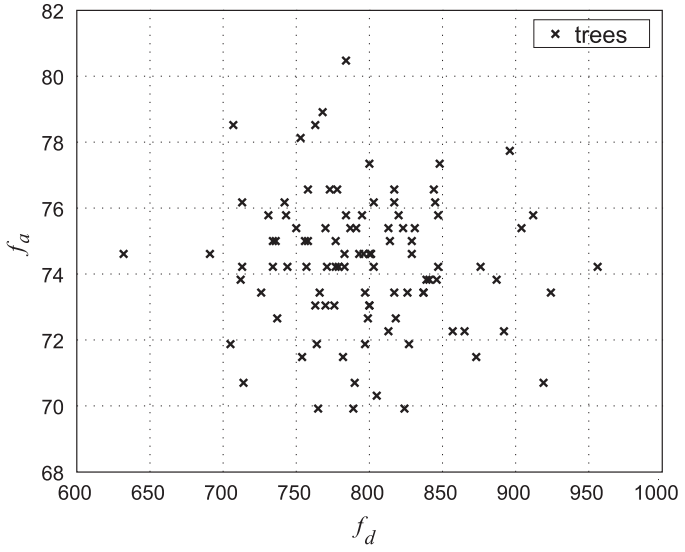


Fig. 11: Binary decision trees with respect to f_d and accuracy f_a.

is never visited. After crossover between T_a and T_b, the child trees T_d and T_e also have dashed subtrees. These unnecessary subtrees occur whenever an attribute occurs more than once in a path of a decision tree. However, by encoding the decision tree, this problem never occurs as illustrated in Fig. 13.

When d is large, limiting the height was recommended. Encoding the binary decision tree in this paper utilized the breadth first traversal. However, pre-order depth first traversal can be utilized as shown in Fig. 13 . 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 immediately 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.

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] Safavian, S.R. and Landgrebe, D., A survey of decision tree classifier methodology,*IEEE Transactions on Systems, Man and Cybernetics*, Vol 21, No. 3, pp 660-674, 1991.

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

[8] 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.

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

[10] Pajunen, P. and Girolami, M., "Implementing decisions in binary decision trees using independent component analysis", in *Proceedings of ICA*, 2000, pp. 477-481.

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

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

[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, pp. 1611-3349.

[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 pp. 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 *Proceedings 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 *Proceedings of 12th IEEE International Conference on Tools with Artificial Intelligence*, 2000, pp. 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] S. Cha and C. C. Tappert, Constructing Binary Decision Trees using Genetic Algorithms, in *Proceedings of International Conference on Genetic and Evolutionary Methods*, July 14-17, 2008, Las Vegas, Nevada.

[19] P. Grnwald, *The Minimum Description Length principle*, MIT Press, June 2007.

[20] 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.

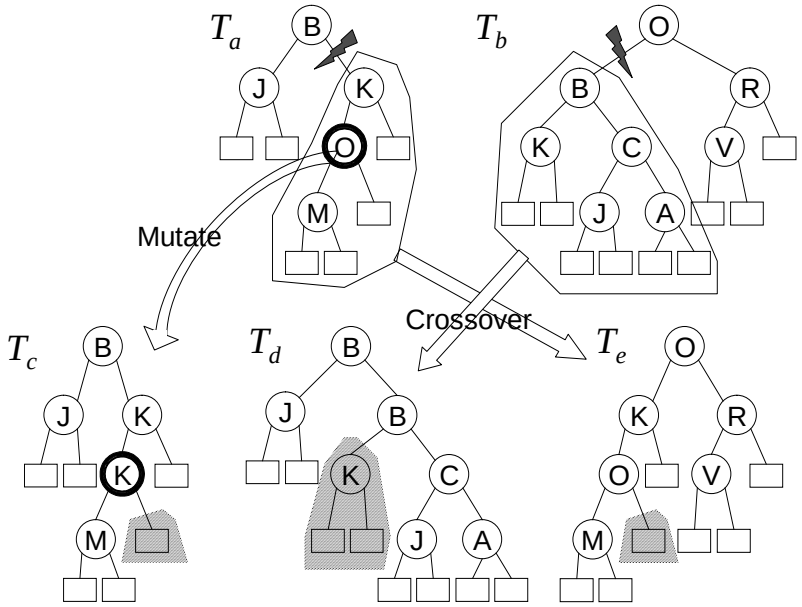


Fig. 12: Direct GA on decision trees.

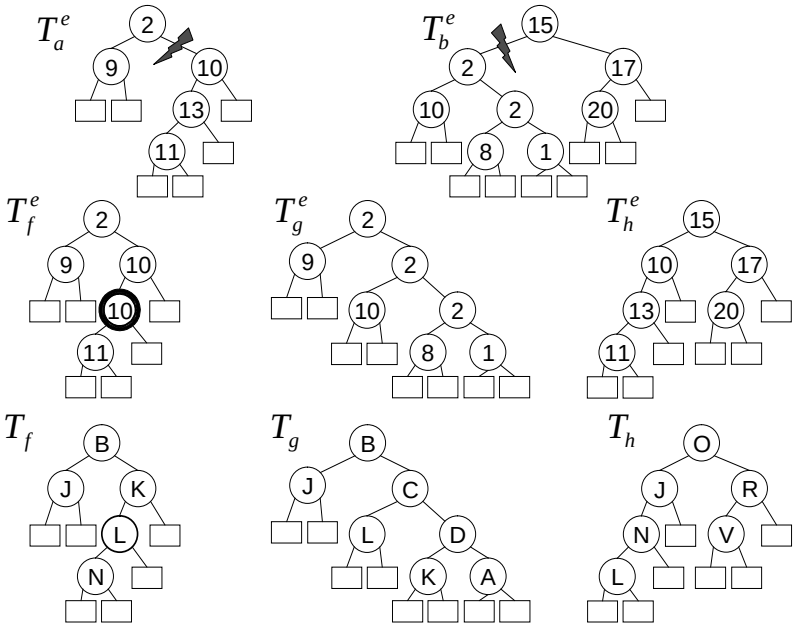


Fig. 13: Pre-order depth first traversal for decision trees.