IEICE TRANS. FUNDAMENTALS, VOL. E75-A, NO. 5 MAY 1992

PAPER

An Approximate Algorithm for Decision Tree Design

Satoru OHTA†, *Member*

SUMMARY Efficient probabilistic decision trees are required in various application areas such as character recognition. This paper presents a polynomial-time approximate algorithm for designing a probabilistic decision tree. The obtained tree is near-optimal for the cost, defined as the weighted sum of the expected test execution time and expected loss. The algorithm is advantageous over other reported heuristics from the viewpoint that the goodness of the solution is theoretically guaranteed. That is, the relative deviation of the obtained tree cost from the exact optimum is not more than a positive constant ε, which can be set arbitrarily small. When the given loss function is Hamming metric, the time efficiency is further improved by using the information theoretical lower bound on the tree cost. The time efficiency of the algorithm and the accuracy of the solutions were evaluated through computational experiments. The results show that the computing time increases very slowly with an increase in problem size and the relative error of the obtained solution is much less than the upper bound ε for most problems.

key words: *decision tree, pattern recognition, optimization, approximation algorithm, statistical decision problem, source coding, polynomial time algorithm*

1. Introduction

The decision tree design problem has long been of great importance. This is because efficient decision trees are required in widely spread areas such as fault diagnosis[1],[2], computer programming[3], medical diagnosis[4],[5], character recognition[6],[7], remote sensing[8],[9], and machine learning[10]. Decision trees represent sequential decision processes commonly found in these applications. In such a process, several tests are sequentially executed to gather information about the occurring unknown state, and then an appropriate action is taken depending on the outcome of tests. Depending on the characteristics of the test outcomes, decision trees are categorized into two types: deterministic and probabilistic. With the former, a test applied to a state always produces a fixed outcome while in the latter a test yields different outcomes with non-zero probability. Probabilistic decision trees are particularly significant in applications with various pattern recognition problems[7],[14].

Reference (11) presented an algorithm to construct efficient probabilistic decision trees. This algorithm strictly optimizes a decision tree for the cost, defined as the weighted sum of the expected test execution time and expected loss. Empirical results have shown that the algorithm is relatively time-efficient among exact algorithms. In spite of this, the worst case computing time with this method is not polynomial in problem size. Thus, the time possibly becomes too large if extremely hard problems are given. Likewise, the algorithm may not be applicable when the problem size is very large. To solve these hard or large problems in practical computing time, a faster algorithm should be developed.

If exact solutions are not strictly required, heuristics are more advantageous than the exact algorithms from a viewpoint of running time. Until now, a number of heuristics have been proposed for the probabilistic decision tree design[7],[12]–[15]. These heuristics are divided into the greedy method and the pruning method. In the greedy method, the tree is constructed in a top-down manner from the root to leaves. At each step of the construction, a tree node is labeled with a test, which is selected on the basis of a locally evaluated measure, for example, the amount of mutual information to be obtained. If a certain stopping rule is satisfied, this test selection procedure is finished by turning the node into a leaf. The greedy method is also used to obtain an initial solution for the pruning method, with which the tree cost is improved by pruning off some of the branches from the initial tree.

The disadvantage of these heuristics is that no performance bounds have been theoretically clarified on accuracy of solutions. Therefore, we must expect that solutions attained using these methods are conditionally far from the strict optimum. For a deterministic decision tree case, Garey and Graham[16] have analyzed the accuracy of solutions obtained using the greedy method. When the tree cost is the expected test execution time, they examined the ratio of the cost achieved by the greedy method to the exact optimum. Needless to say, the accuracy is better for a smaller ratio. However, they have shown that this ratio can become arbitrarily large in the worst case. For the probabilistic decision tree design, no similar analysis has been attempted. Nevertheless, the above result

Manuscript received August 23, 1991.

† The author is with NTT Transmission Systems Laboratories, Yokosuka-shi, 238-03 Japan.

suggests that the worst case performance is not very good in the same way as in the deterministic decision tree case.

For some optimization problems, methods referred to as "ε-approximate algorithms" have been proposed[17],[18]. These algorithms run in polynomial time and yield solutions that have their relative deviation from the optimum upper bounded by a certain constant ε. Such a method is obviously advantageous because the algorithm is guaranteed to always yield good solutions for any problems. The purpose of this paper is to establish such an ε-approximate algorithm for the probabilistic decision tree design.

As in Ref. (11), the tree cost used here is defined as the weighed sum of the expected test execution time and expected loss. Several characteristics are derived for decision trees associated with this cost definition. On the basis of these characteristics, an approximate algorithm is established for designing probabilistic decision trees. These characteristics are also used in the analysis on the proposed algorithm. The cost of the tree obtained by this algorithm is not larger than $1+\varepsilon$ times the optimal value for a given arbitrary constant $\varepsilon(\varepsilon>0)$. The computing time of the proposed method is polynomial in test outcome set size, test set size, state space size, and action set size. It is also shown that the efficiency of the algorithm is further improved by utilizing the lower bound on the cost if the loss function is commonly used Hamming metric. Finally, computational experimental results are depicted to clarify the effectiveness of the proposed method in computing time and solution accuracy.

2. Preliminaries

Suppose that a certain state u occurs from the set $\mathcal{U}=\{u_1, u_2, \cdots, u_K\}$ with a prior probability $Q(u)$ and an action a is taken from the set $\mathcal{A}=\{a_1, a_2, \cdots, a_L\}$. Here, the loss function $d: \mathcal{U}\times\mathcal{A}\rightarrow[0, \infty)$ is defined, and the action taken should be one achieving a small value of the loss. While a prior state probability is known for each state $u\in\mathcal{U}$, the occurring state is unknown. Tests can be carried out to obtain information about the present state. These tests belong to the set $\mathcal{T}=\{t_1, t_2, \cdots, t_N\}$. A test $t\in\mathcal{T}$ produces an outcome x, which is an element of the set $\mathcal{X}=\{0, \cdots, b-1\}$. When a test t is applied to a state u, an outcome x is observed with a conditional probability $P_t(x|u)$.

A decision tree is a rooted tree that represents a sequential decision process, which takes place in the above situation. In a decision tree, an internal node is associated with a test $t\in\mathcal{T}$, and a leaf is labeled with an action $a\in\mathcal{A}$ as shown in Fig. 1. The decision process starts at the root and ends at a leaf. At an internal node, the associated test is executed and the process proceeds to one of its b sons depending on the outcome x. When the process arrives at a leaf by repeating this, the associated action is taken. If the conditional test outcome probability takes a value such that $0<P_t(x|u)<1$, the tree is called a probabilistic decision tree.

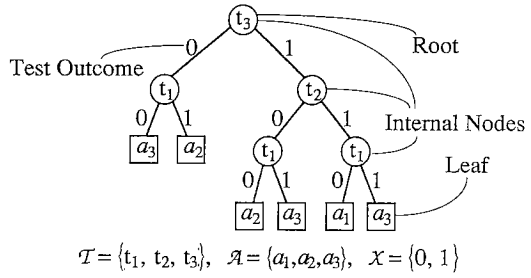


Fig. 1 A decision tree, illustrating terminology.

The decision tree design problem is described as follows:

Problem: From given state probabilities $Q(u)$'s, given conditional test outcome probabilities $P_t(x|u)$'s, and a given loss function d, construct an optimal or near-optimal b-ary decision tree for a specified tree cost.

The input length of this problem is determined by state set size K, action set size L, test set size N, and test outcome set size b. Therefore, it is reasonable to evaluate the efficiency of the algorithm in terms of K, L, N, and b.

The tree cost is defined in terms of the expected test execution time and expected loss. The expected test execution time is the expected value of test numbers from the root to a leaf. The expected loss is the sum of the losses $d(u; a)$'s weighted by the joint probabilities of the state u and the action a. Then, the tree cost C is formulated as,

$$C=W+\rho D, \tag{1}$$

where W is the expected test execution time and D is the expected loss. The constant $\rho(0<\rho<\infty)$ is the weight parameter, which is larger if the expected loss is more strictly required to be less than the test execution time, and smaller otherwise.

The above definition of the tree cost seems adequate to obtain time-efficient and low-loss decision trees. Thus, similar tree cost definitions have been employed in literature[5],[12]. One benefit of this cost definition is that an optimization algorithm can be simplified by utilizing the next characteristic[11].

Characteristic 1: Let C denote the cost of the tree such that the root is labeled with a test. Let $v_0, \cdots, v_{b-1}$ be sons of its root and C_x be the cost of the subtree rooted at v_x $(x\in\mathcal{X})$. Suppose that v_x corresponds to outcome x, which is observed at the root with the probability P_x. Then,

$$C=1+\sum_{x=0}^{b-1} P_x C_x. \tag{2}$$

u_k	u_1		u_2		u_3	
$Q(u_k)$	0.3		0.4		0.3	
	$P_t(0\|u_1)$	$P_t(1\|u_1)$	$P_t(0\|u_2)$	$P_t(1\|u_2)$	$P_t(0\|u_3)$	$P_t(1\|u_3)$
t_1	0.1	0.9	0.2	0.8	0.6	0.4
t_2	0.2	0.8	0.8	0.2	1.0	0.0
t_3	0.9	0.1	0.1	0.9	0.9	0.1

$\mathcal{U}=\{u_1,u_2,u_3\}$, $\mathcal{T}=\{t_1, t_2, t_3\}$, $\mathcal{X}=\{0, 1\}$

Fig. 2 A probabilistic decision table.

The proof for this characteristic is seen in Ref. (11). The algorithm developed in this paper is based on this characteristic as well as several newly revealed characteristics described in Sect. 3.

This paper assumes the following.

Assumption 1: The outcome of different tests are statistically independent. That is, the conditional probability $P_t(x|u)$ does not change before and after other tests are executed.

Assumption 2: Repeated execution of the same test always gives the same outcome. Thus, reapplying identical tests is useless.

Assumption 3: The loss function satisfies that $0 \leqq d(u\,;\,a) \leqq 1$ for any $u \in \mathcal{U}$ and $a \in \mathcal{A}$.

The first two assumptions are often acceptable for practical applications such as character recognition[7],[14]. Nevertheless, these assumptions can be omitted in the algorithm described later. If Assumption 1 is made, probabilities $Q(u)$'s and $P_t(x|u)$'s are indicated in a table as shown in Fig. 2. This table is called a probabilistic decision table following the terminology in Ref. (14). Assumption 3 can be made without loss of generality. If the given loss function is such that $d(u\,;\,a) > 1$ for certain u and a, then consider

$$\rho' = \rho \max_{u,a}[d(u\,;\,a)], \text{ and}$$

$$d'(u\,;\,a) = \frac{d(u\,;\,a)}{\max_{u,a}[d(u\,;\,a)]}$$

as the new weight parameter and new loss function. The new loss function and weight parameter satisfy $0 \leqq d(u\,;\,a) \leqq 1$ without changing the value of the tree cost specified above.

3. Characteristics for Approximate Tree Design

The most trivial solution for the decision tree design problem is a tree consisting of only a leaf. With this tree, the decision process is completed by executing no tests and taking the action that minimizes the expected loss. From Eq. (1), the cost of such a tree is ρD. If the weight parameter ρ is not larger than 1, the optimal solution is a tree consisting of only a leaf. This is verified as follows. Since Assumption 3 asserts that $0 \leqq D \leqq 1$, the cost of the tree consisting of only a leaf, ρD, is not larger than 1 for $0 < \rho \leqq 1$. On the other hand, the cost of a tree executing a test at the root is not less than 1 because $W \geqq 1$. Thus, the tree consisting of only a leaf is optimal. Since the design problem is trivial for $\rho \leqq 1$ in this way, let us assume $1 < \rho < \infty$ in what follows.

Obviously, the cost of a tree consisting of only a leaf is an upper bound on the optimal cost. There is the next relation between these costs.

Lemma 1: Let C_{leaf} denote the cost of a tree consisting of only a leaf and let C_{opt} denote the cost of the optimal tree. Then,

$$C_{\text{leaf}} \leqq \rho C_{\text{opt}}. \tag{3}$$

Proof: First, consider a case where the optimal tree has the root labeled with a test $t_{\text{opt}} \in \mathcal{T}$. Since a test is executed at least once with this tree and the expected loss is non-negative,

$$1 \leqq C_{\text{opt}}. \tag{4}$$

Here, it is clear from Assumption 3 that

$$C_{\text{leaf}} = \rho D_{\text{leaf}} \leqq \rho, \tag{5}$$

where D_{leaf} is the expected loss of the tree consisting of only a leaf. From Eqs. (4) and (5), Eq. (3) is derived.

Next, if the optimal tree consists of only a leaf, then obviously $C_{\text{leaf}} = C_{\text{opt}}$. This satisfies Eq. (3) for $\rho > 1$ and thus the proof is completed. □

Assume that the optimal tree has the root labeled with a certain test t_{opt}. In this tree, let us define v_x and P_x in the same way as in Characteristic 1. The following two lemmas are obtained for the cost of the optimal tree and that of its subtree rooted at v_x.

Lemma 2: In the optimal tree with the root labeled with t_{opt}, the subtree rooted at v_x is optimal for the state probabilities $Q(u_1|x), \cdots, Q(u_K|x)$, where x is an arbitrary element in $\mathcal{X}$. Conversely, in a tree executing t_{opt} at the root, if the subtree rooted at v_x is optimal for every $x \in \mathcal{X}$, then the tree is optimal.

Proof: Both statements are immediately proved from Characteristic 1. □

Lemma 3: In the optimal tree with the root labeled with t_{opt}, let $C_{\text{opt},x}$ denote the cost of its subtree rooted at v_x. Also, let C_t be the cost of a certain tree executing t_{opt} at the root, and let $C_{t,x}$ be that of its subtree rooted at v_x. If $C_{t,x}$ satisfies that

$$C_{t,x} \leqq (1+\delta)\, C_{\text{opt},x}, \text{ for every } x \in \mathcal{X}, \tag{6}$$

for a positive constant δ, then

$$C_t \leqq (1+\eta\delta)\, C_{\text{opt}}, \tag{7}$$

where η is a constant defined by

$$\eta=\frac{\rho-1}{\rho}. \tag{8}$$

Proof: The following two equations are obtained from Eq. (2).

$$C_{\text{opt}}=1+\sum_{x=0}^{b-1} P_x C_{\text{opt},x}, \text{ and} \tag{9}$$

$$C_t=1+\sum_{x=0}^{b-1} P_x C_{t,x}. \tag{10}$$

If Eq. (6) is satisfied, then

$$C_t\leqq 1+\sum_{x=0}^{b-1} P_x C_{\text{opt},x}(1+\delta). \tag{11}$$

Substituting Eq. (9) into Eq. (11), we have

$$C_t\leqq\left(1+\delta-\frac{\delta}{C_{\text{opt}}}\right)C_{\text{opt}}. \tag{12}$$

From Assumption 3, it is easy to show that

$$0< C_{\text{opt}}\leqq C_{\text{leaf}}\leqq\rho. \tag{13}$$

Therefore,

$$C_t\leqq\left(1+\delta-\frac{\delta}{\rho}\right)C_{\text{opt}}=\left(1+\frac{\rho-1}{\rho}\delta\right)C_{\text{opt}}. \tag{14}$$

This completes the proof. □

Here, let us define some terms.

Definition: Consider a decision tree constructed by an approximate algorithm. Let C denote the cost of this tree. Then, the *relative error* E_r for the tree is defined as

$$E_r=\frac{C-C_{\text{opt}}}{C_{\text{opt}}}, \tag{15}$$

where C_{opt} is the optimal cost. If $E_r\leqq\varepsilon$ for a certain positive constant ε, the tree is called *ε-suboptimal*.

(end of definition)

Clearly by this definition, the optimal tree is ε-suboptimal for all $\varepsilon>0$.

The above lemmas derive the next theorem concerning an ε-suboptimal decision tree.

Theorem 1: Let C_{opt} denote the optimal tree cost for given $Q(u)$'s, $P_t(x|u)$'s and $\rho(\rho>1)$. Then, there is an ε-suboptimal tree whose height is not larger than the constant h, defined as

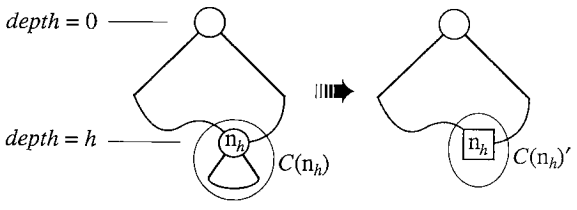


Fig. 3 Explanation for the proof of Theorem 2. Left: the original optimal tree. Right: the modified tree.

$$h=\max\left(\left\lceil\log_\eta\frac{\varepsilon}{\rho-1}\right\rceil, 0\right), \tag{16}$$

where η is given in Eq. (8).

Proof: When the height of the optimal tree does not exceed h defined in Eq. (16), the optimal tree obviously satisfies the theorem statement. Therefore, the proof is shown assuming that the height of the optimal tree is greater than h.

If the height of a tree is greater than h, there is at least one internal node whose depth is h. Here, modify the optimal tree by changing every internal node placed at depth h into a leaf (see Fig. 3). Thus, the height of the tree becomes h by this modification. In the following, the theorem will be proved by showing that the modified tree is ε-suboptimal.

Let n_h be a node whose depth is h. Also, let $C(n_h)$ denote the cost of the subtree rooted at n_h in the original optimal tree and $C(n_h)'$ denote that of the modified tree. Then, Lemma 1 implies that

$$C(n_h)'\leqq\rho C(n_h). \tag{17}$$

If $h=0$, n_h must be the root. Thus, the values $C(n_h)$ and $C(n_h)'$ are the cost of the optimal tree and that of the modified tree. Meanwhile, the definition of h implies that $\varepsilon>\rho-1$ for $h=0$. Therefore, the cost $C(n_h)'$ satisfies the condition that the modified tree is ε-suboptimal.

When $h\geqq 1$, suppose that n_{h-1} is a node sited at depth $h-1$ in the optimal tree. Additionally, let $C(n_{h-1})$ denote the cost of the subtree rooted at n_{h-1} in the original optimal tree and let $C(n_{h-1})'$ denote that of the modified tree. By Lemma 2, every subtree in the original tree is optimal. This characteristic and Eq. (17) suffice the condition of Lemma 3. Therefore,

$$C(n_{h-1})'\leqq\{1+\eta(\rho-1)\}C(n_{h-1}). \tag{18}$$

Let C' be the cost of the modified tree. Then, by repeating the above consideration, we get

$$C'\leqq\{1+\eta^h(\rho-1)\}C_{\text{opt}}. \tag{19}$$

Meanwhile, from the definition of h,

$$h\geqq\log_\eta\frac{\varepsilon}{\rho-1}. \tag{20}$$

Since the constant η is less than 1,

$$\eta^h\leqq\frac{\varepsilon}{\rho-1}. \tag{21}$$

By substituting Eq. (21) into Eq. (19), it is verified that the modified tree cost satisfies $E_r\leqq\varepsilon$. Thus, the proof is completed. □

4. Approximate Algorithm

4.1 Basic Algorithm

The basic algorithm yielding approximate solutions is described as follows.

Algorithm 1: Execute the procedure Approx shown in Fig. 4 to construct an ε-suboptimal decision tree from given $Q(u)$'s, $P_t(x|u)$'s, and ρ.

In the procedure Approx, $\mathbf{Q}$ means the vector $(Q(u_1), \cdots, Q(u_K))$ and $\mathbf{Q}|x$ is the vector $(Q(u_1|x), \cdots, Q(u_K|x))$. Also, P_x is the probability of observing test outcome x at the root. The vector $\mathbf{Q}|x$ and the probability P_x are computed from the augment $\mathbf{Q}$ and the given constants $P_t(x|u)$'s.

Theorem 2: The procedure Approx returns an ε-suboptimal decision tree DT_ε and its cost C_ε for given $Q(u)$'s, $P_t(x|u)$'s, ρ and ε. The height of the obtained tree is not greater than h defined by substituting the given values of ε and ρ into Eq. (16). The procedure uses $O(b^h N^h KL)$ time for given test outcome set size b, test set size N, state set size K, and action set size L.

Proof: First, let us prove the first and second statements by using induction on h and N.

If $N=0$, then the only solution is a tree consisting of only a leaf. In this case, lines 4-8 are not executed and the procedure returns the only solution. Since the height of the constructed tree is 0, the statements are correct. For ε and ρ giving $h=0$, Eq. (16) implies that $\varepsilon \geqq \rho - 1$. This means that lines 3-8 are not executed because of the if-statement of line 2. Therefore, the procedure returns a tree consisting of only a leaf and its cost C_{leaf}. Meanwhile, Lemma 1 asserts that this solution is ε-suboptimal for $\varepsilon \geqq \rho - 1$. Furthermore,

```
Procedure Approx(Q, T, ε, C_ε, DT_ε);
  begin
1.  DT_ε := a tree consisting of only a leaf; C_ε := C_leaf;
2.  if (ε < ρ − 1) and (T ≠ ∅) then
3.    for every t ∈ T do begin
4.      for x := 0 to b − 1 do
            Approx(Q|x, T − {t}, ερ/(ρ − 1), C_{ε,x}, DT_{ε,x});
5.      C := 1 + Σ_{x=0}^{b−1} P_x C_{ε,x};
6.      if C < C_ε then begin
7.        C_ε := C;
8.        DT_ε := a tree consisting of the root labeled with t
                and subtrees DT_{ε,0}, ..., DT_{ε,b−1}
          end
        end
  end.
```

Fig. 4 The procedure Approx describing the basic approximate algorithm.

the height of this tree is equal to $h(=0)$. Hence, the statements are correct also if $h=0$.

Let us examine the case of $N>0$ and $h>0$ assuming the validity of the statements for $N-1$ and $h-1$. For the parameters ε and ρ yielding $h>0$, suppose that ε is changed to $\varepsilon\rho/(\rho-1)$. By changing ε into $\varepsilon\rho/(\rho-1)$ in Eq. (16), it is verified that this replacement decreases h by 1. This fact and the assumption imply that line 4 correctly constructs subtrees whose heights are upper bounded by $h-1$. Therefore, the height of a tree stored by line 1 or 8 does not exceed h and thus the second statement is proved. Now let us turn to the first statement. It is easily seen from lines 1, 5, and 7 that C_ε is exactly the cost of a tree represented by DT_ε. Additionally, line 6 indicates that C_ε decreases whenever updated. Thus, we have to show that C_ε takes a value not exceeding $(1+\varepsilon)\,C_{\mathrm{opt}}$ at least once during the computation. Consider a case where the optimal tree has the root labeled with a certain test t_{opt}. The assumption of the induction implies that the costs of subtrees obtained in line 4 have the relative errors bounded by $\varepsilon\rho/(\rho-1)$. Lemma 3 asserts that these subtree costs yield the cost satisfying $C \leqq (1+\varepsilon)\,C_{\mathrm{opt}}$ when the variable t is equal to t_{opt} in the loop of lines 3-8. This means that C_ε certainly becomes less than or equal to $(1+\varepsilon)\,C_{\mathrm{opt}}$. On the other hand, if the optimal tree is a tree consisting of only a leaf, C_ε is set to be C_{opt}. That is, C_ε satisfies $C_\varepsilon \leqq (1+\varepsilon)\,C_{\mathrm{opt}}$ also for this case. This completes the proof of the first statement.

Next, let us examine the computing time. Let $T(h, N, K, L)$ denote the time for the constant h, test set size N, state set size K, and action set size L. Then,

$$T(h,N,K,L) \leqq \begin{cases} c_1, & \text{if } h=0 \text{ or } N=0 \\ c_1 + N\{c_2 + t(h) \\ \quad + bT(h-1, N-1, K, L)\}, \\ \text{otherwise} \end{cases} \tag{22}$$

where c_1 is a constant representing time to execute lines 1-2, and c_2 is a constant for the computation of $\mathbf{Q}|x$ and P_x, as well as testing the conditions in lines 3-4. Also, time for the execution of lines 5-8 is expressed by the term $t(h)$, which is a function of the height of the constructed tree. By expanding Eq. (22) for $N>h$, we have

$$T(h,N,K,L) \leqq \sum_{i=0}^{h} N^i b^i c_1 + \sum_{i=0}^{h-1} (c_2 + t(h-i))\, N^{i+1} b^i. \tag{23}$$

Since the heights of the subtrees constructed in line 4 is bounded by $h-1$, line 8 uses $O(b^h)$ time. Lines 5-7 use $O(b)$ time and thus $t(h)$ is bounded by cb^h, where c is a certain constant. Substituting this value into Eq. (23),

$$T(h, N, K, L) \leq 2c_1 N^h b^h + 2c_2 N^h b^{h-1} + 2cN^h b^h. \tag{24}$$

Here, the constant c_1 is $O(KL)$ for most loss functions. This is because it takes $O(KL)$ time to select the action labeled at the leaf in line 1. Meanwhile, the computation of $\mathbf{Q}|x$ and P_x uses $O(bK)$ time and the conditions in lines 3-4 are tested in $O(b)$ time. Consequently, c_2 is $O(bK)$. Replacing c_1 and c_2 in Eq. (24) with these values, we obtain $T(h, N, K, L) = O(b^h N^h KL)$. □

Next, let us see the necessity for the assumptions mentioned in Sect. 2. If Assumption 1 does not hold, $P_t(x|u)$ is not constant any more. However, if $P_t(x|u)$ can be known from the sequence of the executed tests and their outcomes, the procedure becomes applicable by making a slight change. To do this, define a list whose element is a pair of an executed test and its outcome. Then, by adding the list as a parameter of the procedure, $P_t(x|u)$ and $\mathbf{Q}|x$ becomes computable because $P_t(x|u)$ is known from the list. Thus, the procedure correctly works even if the assumption is not satisfied. However, the problem description becomes too complex and impractical if one specifies $P_t(x|u)$'s for all the possible sequences. For this reason, let us employ Assumption 1 throughout the whole paper.

In many applications, Assumption 2 is required to hold[14]. Thus, it is proper to make this assumption. However, there may be an application such that the repeated execution of the same test is allowed. For such an application, the parameter $\mathcal{T}-\{t\}$ in the recursive call of line 4 must be modified into $\mathcal{T}$ because the available test set does not change at every node in a decision tree. This modification does not affect the correctness of the procedure. This is proved in almost the same way as above except that the induction is carried out only on h. Therefore, the algorithm can be used likewise if Assumption 2 is not satisfied. In this paper, Assumption 2 is made also because it is easy to compare the algorithm with the method of Ref. (11), in which the assumption is indispensable for correct computation.

4.2 Efficient Method for the Hamming Metric Loss Function

The computing time shown in Theorem 2 infers that the described algorithm is efficient only for ε and ρ giving a small value of h. However, we can improve the time efficiency of the algorithm if the loss function is Hamming metric. This characteristic is favorable because the Hamming metric is the most commonly used loss function.

The Hamming metric is defined as

```
Procedure FastApprox(Q, 𝒯, ε, C_ε, DT_ε);
  begin
1.  DT_ε:= a tree consisting of only a leaf; C_ε:= C_leaf;
2.  if (ε < ρ − 1) and (𝒯 ≠ ∅) then
3.   if C_ε > (1 + ε)G_ρ(Q) then
4.    for every t ∈ 𝒯 do begin
5.      g:= 1 + Σ_{x=0}^{b−1} P_x G_ρ(Q|x);
6.      if C_ε > (1 + ε)g then begin
7.          for x:= 0 to b − 1 do
              FastApprox(Q|x, 𝒯 − {t}, ερ/(ρ − 1), C_{ε,x}, DT_{ε,x});
8.          C:= 1 + Σ_{x=0}^{b−1} P_x C_{ε,x};
9.          if C < C_ε then begin
10.             C_ε:= C;
11.             DT_ε:= a tree consisting of the root labeled with
                        t and subtrees DT_{ε,0}, ..., DT_{ε,b−1}
                end
            end
        end
  end.
```

Fig. 5 The procedure FastApprox describing the improved approximate algorithm, which is built for the Hamming metric loss function.

$$d(u\,;\,a) = \begin{cases} 0, & \text{if } u = a \\ 1, & \text{otherwise} \end{cases} \tag{25}$$

for $\mathcal{U} = \mathcal{A}$. When $d(u\,;\,a)$ is defined as above, we obtain the next characteristic, which originates from the source coding theorem in information theory.

Characteristic 2: It can be assumed that states are expressed by numbers $1, \cdots, K$ satisfying $Q(1) \geqq Q(2) \geqq \cdots \geqq Q(K)$ without loss of generality. Let $\mathbf{Q}$ represent the vector $(Q(1), \cdots, Q(K))$. Then, for given $\mathbf{Q}$, ρ and the Hamming metric loss function, the tree cost C is lower bounded by the quantity $G_\rho(\mathbf{Q})$ defined as follows.

$$G_\rho(\mathbf{Q}) = \min_{1 \leq m \leq K} \left[-\sum_{k=1}^{m} Q_m(k) \log_b Q_m(k) + \rho \sum_{k=m+1}^{K} Q(k) \right], \tag{26}$$

where

$$Q_m(k) = \begin{cases} Q(1) + Q(m+1) + \cdots + Q(K), \\ \quad \text{for} \quad k = 1 \\ Q(k), \\ \quad \text{for} \quad 2 \leqq k \leqq m. \end{cases} \tag{27}$$

The proof for this characteristic is found in Ref. (11).

By utilizing the lower bound $G_\rho(\mathbf{Q})$ defined in Eq. (26), the time efficiency of the described algorithm can be improved. This acceleration is achieved by introducing the suboptimization technique[19] based on a branch-and-bound method. The accelerated algorithm is as follows.

Algorithm 2: Execute the procedure FastApprox shown in Fig. 5 to construct an ε-suboptimal decision tree from given $Q(u)$'s, $P_t(x|u)$'s, and ρ.

Theorem 3: The procedure FastApprox constructs an ε-suboptimal decision tree DT_ε and its cost C_ε for given $Q(u)$'s, $P_t(x|u)$'s, ρ, ε, and the Hamming metric loss function. The computing time is $O(b^h N^h K \log K)$ for given b, N, and K.

Proof: If $N=0$ or $h=0$, or if the optimal solution is a tree consisting of only a leaf, the first statement is proved by the same discussion as in Theorem 2. Next, suppose that $N>0$ and $h>0$. When the optimal tree has the root labeled with a test t_{opt}, assume that lines 7-11 are executed for $t=t_{\mathrm{opt}}$. Then, in the similar way as the proof of Theorem 2, the value C becomes the cost of an ε-suboptimal tree. Thus, an ε-suboptimal solution is certainly stored in DT_ε and C_ε. If lines 7-11 are not executed for $t=t_{\mathrm{opt}}$, then the conditional expression in line 3 or 6 is not met. When the conditional expression in line 3 is not satisfied, $C_\varepsilon \leqq (1+\varepsilon) C_{\mathrm{opt}}$ because $G_\rho(\mathbf{Q}) \leqq C_{\mathrm{opt}}$. This means that an ε-suboptimal solution is already found. On the other hand, when the conditional expression in line 6 does not hold, the discussion is the same as above for the reason that the value g for t_{opt} is the lower bound on C_{opt}. Therefore, the first statement of the theorem is verified.

The computing time of FastApprox is also formulated as the recurrence of Eq. (22). Here, the constant c_1 represents time to execute lines 1-3. The constant c_2 describes time for the execution of lines 5-6, and testing the conditions of lines 4 and 7 as well as the computation of $\mathbf{Q}|x$, P_x, and $G_\rho(\mathbf{Q}|x)$. For Hamming metric, line 1 uses $O(K)$ time because the optimal action for a leaf can be chosen by finding $u_{\max}$ such that $Q(u_{\max}) \geqq Q(u)$ for all $u \neq u_{\max}$. The lower bound $G_\rho(\mathbf{Q})$ is computed in $O(K \log K)$ time[20], and thus c_1 is $O(K \log K)$. The constant c_2 is $O(bK \log K)$. This is because the computation of $G_\rho(\mathbf{Q}|x)$ uses $O(K \log K)$ time for each value of x while other operations use $O(bK)$ time. The term $t(h)$ is bounded by cb^h in the same way as in Theorem 2. By substituting the values of c_1, c_2 and $t(h)$ into Eq. (23), the time is proved to be $O(b^h N^h K \log K)$. □

As shown in Theorem 3, the worst case computing time of the procedure FastApprox does not look much better than that of Approx. However, the subtree construction step of line 7 is not necessarily executed for some t's$\in \mathcal{T}$ in the former while the latter constructs subtrees for every t in $\mathcal{T}$. If this elimination of subtree construction in FastApprox is carried out for a large subset of $\mathcal{T}$, the time will greatly decrease. The next section demonstrates the effect of eliminating unnecessary subtree construction through computational experiments.

5. Experimental Results

To evaluate the practical performance of the proposed approximate algorithm, a computational experiment was carried out. The procedure FastApprox was programmed in the turbo-Pascal language, and executed for randomly generated problems on an NEC PC-9801EX4 computer. The performance of the algorithm was tested by measuring the computing time and the accuracy of solutions. The exact optimal algorithm described in Ref. (11) was also examined to clarify the effectiveness of the proposed method in computing time and to obtain the exact optimal cost, which is necessary to estimate the relative error of a solution.

The problems fed to the algorithms were generated in the same way as those used in Ref. (21). That is, the constant b was 2, the state space size was 8, and the state probabilities $Q(u)$'s were computed by Zipf's law[22]. The conditional test outcome probability $P_t(1|u)$ was randomly determined from the uniform distribution over the interval that was selected from [0, 0.1] and [0.9, 1] with the probability of 0.5. The value of $P_t(0|u)$ was simply $1-P_t(1|u)$. The test set size was 5, 6, 7, 10, 20, 30, 50, or 100. For each test set size, 10 problems were generated. The weight parameter ρ was 8. The accuracy parameter ε for the proposed method was set to 0.1 and 0.2.

The experimental values of ε and ρ yield that $h=32$ for $\varepsilon=0.1$ and $h=27$ for $\varepsilon=0.2$. The algorithm

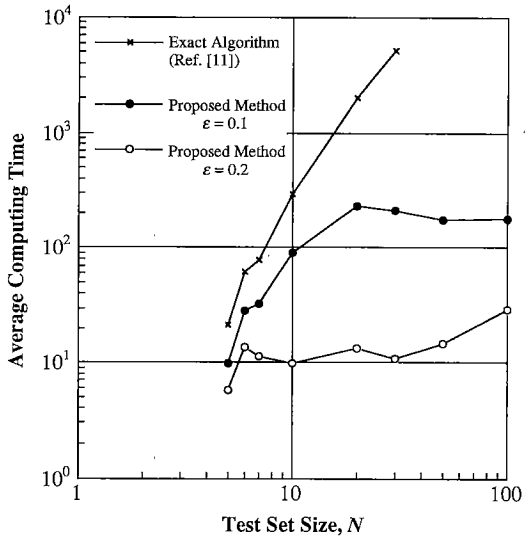


Fig. 6 Experimental result (1). Average computing time vs. test set size.

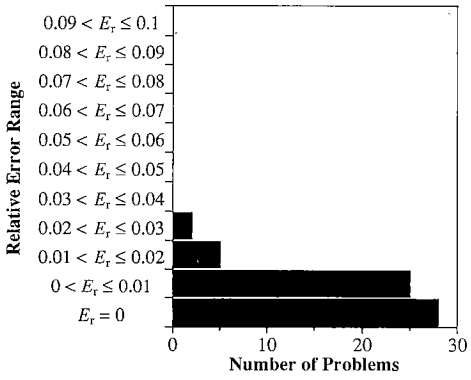


Fig. 7 Experimental result (2). Number of problems versus range of relative error for $\varepsilon=0.1$.

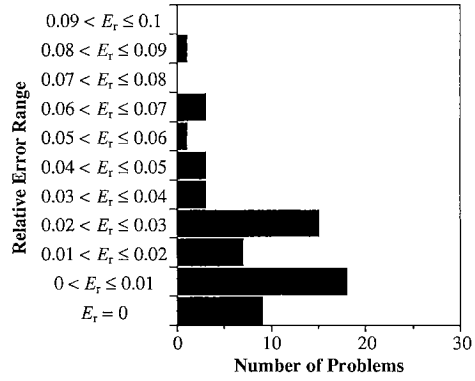


Fig. 8 Experimental result (3). Number of problems versus range of relative error for $\varepsilon=0.2$.

does not seem very efficient for these numerical values of h; the worst case time derived in Sect. 4 rapidly (proportionally to N^h) increases for the growth of N. Therefore, to declare that the algorithm is practical, we need to show that the acceleration technique employed in FastApprox effectively lessens this rapid increase of computing time. From this viewpoint, the computing time was examined in terms of N.

The measured average computing time is plotted against N in Fig. 6. The measured time increases surprisingly slowly for the increasing test set size. For example, when $\varepsilon=0.1$, the average computing time for $N=100$ is only 18 times as large as that for $N=5$. For $\varepsilon=0.2$, the time grows merely 5 times larger when N is enlarged from 5 to 100. This result suggests that the practical efficiency of the procedure FastApprox is much better than the worst case time obtained by the analysis. Thus, it is implied that unnecessary computation is successfully eliminated in FastApprox. In comparison with the exact algorithm, the time with the proposed approximate method is by far smaller particularly when $\varepsilon=0.2$. As described in Ref. (11), the examined exact algorithm is also fairly time-efficient. However, it is evident from Fig. 6 that the size of solvable problems is greatly enlarged if a relative error of 10% or 20% can be allowed.

The accuracy was examined for 60 problems with test set size 5, 6, 7, 10, 20, and 30. Figure 7 shows the number of problems versus the range of the relative error when $\varepsilon=0.1$, and Fig. 8 illustrates that for $\varepsilon=0.2$. It is plain to see that all the obtained solutions have relative errors less than the specified upper bound ε. Moreover, it is observed that the relative error is much smaller than the value of ε in most cases. For example, when $\varepsilon=0.1$, 53 of 60 problems (88%) have relative errors less than 0.01, and the cost of every solution is at most 3% larger than the optimum. Even if $\varepsilon=0.2$, the relative errors for 49 problems (81%) are in the range of $0 \leqq E_r \leqq 0.03$. This suggests that the algorithm generates almost optimal solutions for most problems if ε is set not larger than these experimental values.

6. Conclusion

This paper presents an approximate algorithm for the probabilistic decision tree design. The feature of this method is that the relative error of the obtained solution is guaranteed to be not larger than a specified value ε. Since the parameter ε can take any value such that $\varepsilon>0$, the solution can be arbitrarily close to the exact optimum. Such a guarantee of accuracy has not been obtained with any other approximate algorithms. The computing time of the method is $O(b^h N^h KL)$, i. e., a polynomial of outcome set size b, test set size N, state space size K, and action space size L. Here, the parameter h is a function of ε and the weight parameter ρ.

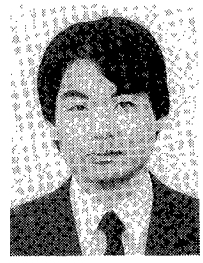
From the worst case computing time derived by the analysis, the algorithm looks quick only for ε and ρ yielding a small value of h. However, if the loss function is Hamming metric, the time efficiency can be greatly improved. This improvement is achieved by utilizing the information theoretical lower bound on the tree cost. This bound is the one derived in Ref. (11) and plays an important role also in the development of the efficient exact algorithm. The computing time using this bound was evaluated through a computational experiment. The result shows that the algorithm is much faster for most problems than the analysis suggests. This characteristic implies that employing the bound successfully eliminates unnecessary computation. From the comparison with the computing time of the exact algorithm in Ref. (11), it was clarified that much larger problems can be solved by allowing a relative error of at most 10% or 20%. Moreover, the experimental result indicates that the relative error of a solution obtained with the proposed method is much smaller than the upper bound ε for most problems.

While the worst case computing time of the proposed method is polynomial in the problem input

length, it is not polynomial in the magnitude of ρ or $1/\varepsilon$. Therefore, it is natural to ask if there is an ε-approximate algorithm whose computing time is polynomial also in ρ and $1/\varepsilon$. The development of such an algorithm is left for further study.

References

(1) Brulē J. D., Johnson R. A. and Kletsky E. J.: "Diagnosis of equipment failures", IRE Trans. Reliability and Quality Control, **RQC-9**, pp. 23-34 (April 1960).
(2) Varshney P. K., Hartmann C. R. P. and De Faria J. J. M.: "Application of information theory to sequential fault diagnosis", IEEE Trans. Comput., **C-31**, 2, pp. 164-170 (Feb. 1982).
(3) Reinwald L. T. and Soland R. M.: "Conversion of limited-entry decision tables to optimal computer programs I: Minimum average processing time", J. of the Assoc. Comp. Mach., **13**, pp. 339-358 (July 1966).
(4) Meisel W. S.: "Design of decision structures for medical diagnosis", in Proc. 1975 Int. Conf. Cybernetics and Society, New York (1975).
(5) Peters R. J.: "Zero order and nonzero order decision rules in medical diagnosis", IBM J. Res. and Develop., **21**, pp. 449-460 (Sept. 1977).
(6) Wang Q. R. and Suen C. Y.: "Analysis and design of a decision tree based on entropy reduction and its application to large character set recognition", IEEE Trans. Pattern Anal. Machine Intell., **PAMI-6**, 4, pp. 406-417 (July 1984).
(7) Casey R. G. and Nagy G.: "Decision tree design using a probabilistic model", IEEE Trans. Inf. Theory, **IT-30**, 1, pp. 93-99 (Jan. 1984).
(8) Fujimura S. and Hanaizumi H.: "Hierarchical algorithms for the classification of remotely sensed multispectral images", IEICE Trans., **E74**, 2, pp. 295-301 (Feb. 1991).
(9) Swain P. H. and Hauska H.: "The decision tree classifier: design and potential", IEEE Trans. Geosci. Electron., **GE-15**, 3, pp. 142-147 (July 1977).
(10) Quinlan J. R.: "Learning efficient classification procedures and their applications to chessend games", in Machine Learning: An Artificial Intelligence Approach, Michaelski R. S., Carbonell J. G. and Mitchell T. M., Eds. Palo Alto, CA: Tioga, 1983, pp. 463-482.
(11) Ohta S. and Kanaya F.: "Optimal decision tree design based on information theoretical cost bound", in Proc. ISITA'90, Honolulu, Hawaii (Nov. 1990).
(12) Chou P. A., Lookabaugh T. and Gray R. M.: "Optimal pruning with applications to tree-structured source coding and modeling", IEEE Trans. Inf. Theory, **IT-35**, 2, pp. 299-315 (March 1989).
(13) Goodman R. M. and Smyth P.: "Decision tree design from a communication theory standpoint", IEEE Trans. Inf. Theory, **IT-34**, 5, pp. 979-994 (Sept. 1988).
(14) Khuri S., Hartmann C. R. P. and Varshney P. K.: "Application of information theory to the construction of efficient decision diagrams", preprint.
(15) Khuri S., Hartmann C. R. P. and Varshney P. K.: "Applying information theory to the construction of decision diagrams", in Proc. Int. Symp. on Inf. Theory (June 1988).
(16) Garey M. R. and Graham R. L.: "Performance bounds on the splitting algorithm for binary testing", Acta Informatica, **3**, pp. 347-355 (1974).
(17) Sahni S. and Gonzalez T.: "P-complete approximation problems", J. Ass. Comput. Mach., **23**, 3, pp. 555-565 (July 1976).
(18) Sahni S. K.: "Algorithms for scheduling independent tasks", J. Ass. Comput. Mach., **23**, 1, pp. 116-127 (Jan. 1976).
(19) Lawler E. L. and Wood D. E.: "Branch-and-bound methods: A survey", Oper. Res., **14**, pp. 699-719 (July 1966).
(20) Ohta S. and Kanaya F.: "Optimal decision tree design based on information theoretical cost bound", IEICE Trans., **E74**, 9, pp. 2523-2530 (Sept. 1991).
(21) Ohta S. and Kanaya F.: "Optimization of decision trees for storage space and expected loss", to appear.
(22) Knuth D. E.: "The art of computer programming vol. 3: Sorting and searching, Addison-Wesley (1973).



Satoru Ohta received the B. E. and M. E. degree in electronics engineering from the Tokyo Institute of Technology, Tokyo, Japan, in 1981 and 1983, respectively. Since 1983 he has been with NTT, where he worked on research and development of cross-connect systems, transmission network synthesis, broadband ISDN and sequential decision processes. He is currently involved in research of the object-oriented approach for transmission systems management at NTT Transmission Systems Laboratories. He is a member of the IEEE. He received the Excellent Paper Award in 1991 from IEICE.