ELSEVIER

Theoretical Computer Science 270 (2002) 609–623

Theoretical Computer Science

www.elsevier.com/locate/tcs

Decision tree approximations of Boolean functions

Dinesh Mehta[a,1], Vijay Raghavan[b,*,2]

[a] *Department of Mathematical and Computer Sciences, Colorado School of Mines, Golden, CO 80401-1887, USA*

[b] *Department of Computer Science, Box 1670-B, Vanderbilt University, Nashville, TN 37235, USA*

Received January 2000; revised October 2000; accepted November 2000
Communicated by O. Watanabe

Abstract

Decision trees are popular representations of Boolean functions. We show that, given an alternative representation of a Boolean function f, say as a read-once branching program, one can find a decision tree T which approximates f to any desired amount of accuracy. Moreover, the size of the decision tree is at most that of the smallest decision tree which can represent f and this construction can be obtained in quasi-polynomial time. We also extend this result to the case where one has access only to a source of random evaluations of the Boolean function f instead of a complete representation. In this case, we show that a similar approximation can be obtained with any specified amount of confidence (as opposed to the absolute certainty of the former case.) This latter result implies proper PAC-learnability of decision trees under the uniform distribution without using membership queries.

MSC: 05C60; 20B25; 68Q15; 68Q25

Keywords: Decision trees; Representations of Boolean functions; Learning theory; Algorithms

1. Introduction

Decision trees are popular representations of Boolean functions. They form the basic inference engine in well-known machine learning programs such C4.5 [11, 12]. Boolean decision trees have also been used in the problem of performing reliable computations in the presence of faulty components [10] and in medical diagnosis. The popularity of

* Corresponding author. Tel.: +1-615-322-3067/2796; fax: +1-615-343-5459.
E-mail addresses: dmehta@mines.edu (D. Mehta), raghavan@vuse.vanderbilt.edu (V. Raghavan).

[1] Supported by NSF grant CCR-9988338.
[2] Supported by NSF grant CCR-9820840.

PII: S0304-3975(01)00011-1

Table 1
The complexity of operations in different representation schemes

—	DNF formulas	Read-once branching programs	Decision trees
Universality	Yes	Yes	Yes
AND of 2 representations	Polynomial time[a]	Superpolynomial time[b]	Polynomial time[a]
OR of 2 representations	Polynomial time[a]	Superpolynomial time[b]	Polynomial time[a]
Complement of a representation	Exponential time[a]	Polynomial time[a]	Polynomial time[a]
Deciding satisfiability	Polynomial time[a]	Polynomial time[a]	Polynomial time[a]
Deciding unsatisfiability	NP complete[c]	Polynomial time[a]	Polynomial time[a]
Deciding monotonicity	co-NP complete[c]	Open	Polynomial time[d]
Deciding equivalence	co-NP complete[c]	co-RP[e]	Polynomial time[f]
Deciding symmetry	co-NP complete[c]	Polynomial time[d]	Polynomial time[d]
Deciding relevance of variables	co-NP complete[c]	Open	Polynomial time[f]
Counting number of satisfying assignments	#P-complete[g]	Polynomial time[d]	Polynomial time[d]
Making representation irredundant	NP hard[c]	co-RP[e]	Polynomial time[f]
Making representation minimum	NP hard[c]	Open	NP-hard[h]
Truth-table minimization	NP hard[i]	Open	Polynomial time[j]

[a] Straightforward from the definition of the representation scheme.
[b] It can be shown that the AND and OR of two read-once branching programs B1 and B2 cannot in general be expressed as a read-once branching program in time polynomial in B1 and B2 unless P = NP.
[c] Easy reduction from CNF-SATISFIABILITY.
[d] Proved in this paper.
[e] Is a result (or follows from one) in [1].
[f] It is a folk theorem that decision trees are testable for equivalence in polynomial time; the other results follow from this.
[g] Proved in [13].
[h] Proved in [14].
[i] This is a well-known result of Masek cited in Garey and Johnson's book [7].
[j] Proved in [8].

decision trees for representing Boolean functions may be attributed to the following reasons:

– *Universality*: Decision trees can represent all Boolean functions.
– *Amenability to manipulation*: Many useful operations on Boolean functions can be performed efficiently in time polynomial in the size of the decision tree representation. In contrast, most such operations are intractable under other popular representations. Table 1 gives a comparison of decision trees with DNF formulas and read-once branching programs.

The advantages of a decision tree representation motivate the following problem: *Given an arbitrary representation of a Boolean function f, find an equivalent representation of f as a decision tree of as small a size as can be.*

It is immediately evident that this problem is bound to be hard as stated. Polynomial time solvability of this problem would imply that satisfiability of CNF formulas can be decided in polynomial time which is impossible unless P = NP. We therefore consider a slightly different problem. Let us say that g is an ε-*approximation* of f if the fraction of assignments on which g and f differ in evaluation is at most ε.

Given an arbitrary representation of a Boolean function f, find an ε-approximation of f as a decision tree of as small a size as can be.

In order not to fall into the same trap as before, we are now interested in solving this problem efficiently but realistically: that is, we may use time polynomial in the following parameters:

1. The size of the given representation of f.
2. The size of the smallest decision tree representation of f for a given ε.
3. The inverse of the desired error tolerance, i.e., $1/\varepsilon$.

Such approximations would be useful in all applications, where a small amount of error can be tolerated in return for the gains that would accrue from having a decision tree representation. Indeed this is the case for most applications in machine learning and data mining. For example, one could post-process the hypothesis output of a learning program and convert it into a decision tree while ensuring that not much error has been introduced by choosing a suitably small ε. Note here that one may use knowledge of special properties of the representation scheme of the hypothesis in constructing the decision tree approximation. Further note that one may even construct a decision tree approximation for a decision tree hypothesis! This would be useful in conjunction with programs like C4.5 which output decision trees but do not make special efforts to ensure that the output tree is provably the smallest it can be for a desired error tolerance. At the expense of sacrificing a little more error, one could achieve the desired minimization in such cases.

We first show that in the case of some well-known representation schemes, small ε-approximating decision trees can be obtained in *quasi*-polynomial time. (More precisely, a polynomial factor of the size $|f|$ of the input function f is multiplied by a factor which involves an exponent logarithmic in $1/\varepsilon$, where ε is the desired error tolerance, and the size m of the smallest decision tree which can represent f.) These schemes are:

1. Decision trees,
2. Ordered binary decision diagrams,
3. Read-once branching programs,
4. $O(\log n)$-height branching programs,
5. Sat-j DNF formulas, for constant j,
6. μ-Boolean formulas.

The third item above is a generalization of the first two, so the result for the first two follows from the third. Our quasi-polynomial time algorithm actually holds with more generality than for just these classes. Roughly speaking, all representation schemes for which the number of satisfying assignments of the input function under "small" projections can be computed efficiently – a property we call *sat-countable* in this paper – would come under the technique employed here. Indeed, we present the algorithm in this more general way and then argue that the required properties hold for all the above schemes. It is worth emphasizing here that although the time taken by our algorithm is quasi-polynomial, the size of the decision tree approximation is *not*: in fact, the output decision tree has the smallest size that *any* decision tree of its height and level

of approximation can have. In this sense, it is optimal and certainly has size no larger than that of the smallest decision tree which can represent the Boolean function being approximated.

We also consider the situation where only *some* evaluations of a Boolean function f are available. Given a sample S of such evaluations, we show that the previous algorithm can be modified slightly to give a quasi-polynomial time algorithm which produces a small ε-approximating decision tree over the sample S. That is, the decision tree may disagree with f in evaluating at most $\varepsilon \cdot |S|$ assignments out of S, for any given ε.

We argue that this latter result implies proper quasi-polynomial time PAC-learnability of decision trees under the uniform distribution. Informally, the learning result may be interpreted as follows. Compared to the absolute certainty of the ε-approximation in the first result, the learning result says that if we are given access only to a source of random evaluations of f (instead of a complete representation of f) then the output of our algorithm will be an ε-approximating decision tree with as much confidence as desired, but not absolute certainty. This may be the only way to obtain decision tree approximations for representation schemes like DNF formulas for which counting the number of satisfying assignments is #P-complete [7].

A novel feature of the learning algorithm is that it is *not* an Occam algorithm [2] unlike the ones known in learning theory. This is because our algorithm may actually make a few errors even on the training sample used. Consequently, the analysis of the sample complexity is a generalization of the ones normally used, and may be of some independent interest. This work may be compared with the work of Domingo, Tsukiji, and Watanabe which considers Occam algorithms with some errors [5].

The learning result can be compared with similar ones in learning theory. Bshouty's monotone-theory-based algorithm [3] can be deployed to learn decision trees under any arbitrary but fixed distribution in polynomial time but has the following drawbacks in comparison with our algorithm: the algorithm uses membership queries and outputs not a decision tree but a depth-3 formula. Similarly, Bshouty and Mansour's algorithm [4] does not output a decision tree. Ehrenfeucht and Haussler [6] show that decision trees of rank r are learnable in time $n^{O(r)}$ under any distribution. The rank of a decision tree T is the height of the largest complete binary tree that can be embedded in T. Since a decision tree of m nodes has rank at most $\log m$, at first glance, this result would seem to be an improvement over the learning result of this paper since one could learn m node decision trees in quasi-polynomial time under any distribution! The difference is this: in learning m node decision trees over n variables our algorithm would always produce a decision tree of size no larger than m using a sample of size at most polynomial in m and the inverse of the error and confidence parameters. In contrast, the algorithm of Ehrenfeucht and Haussler may output a tree of size $n^{O(\log m)}$ using a sample of size quasi-polynomial in n, m and polynomial in the inverse of the error and confidence parameters.

The rest of the paper is organized as follows. Section 2 contains definitions and lemmas used in the remaining sections. Section 3 has our algorithm for finding an

ε-approximating decision tree given a sat-countable representation. Section 4 contains the results on ε-approximating decision trees given only a source of random evaluations of a Boolean function. We conclude with some open problems in Section 5.

2. Preliminaries

Let f be a Boolean function over a set $V=\{v_1, v_2, \ldots, v_n\}$ of n variables. A (*total*) *assignment* is obtained by setting each of the n variables to either 0 or 1; such an assignment may be represented by an n-bit vector in $\{0,1\}^n$ in the natural way. A *satisfying* assignment α for f is one for which $f(\alpha)=1$. The number of satisfying assignments for f is denoted by $\sharp f$.

A *partial assignment* is obtained when only a subset of variables in V is assigned values. A partial assignment may be represented by a vector of length n each of whose elements is either $0, 1$, or *. A vector element is * if the corresponding variable was not assigned a value. Thus, the total number of partial assignments is 3^n and the number of partial assignments with k variables assigned values is $\binom{n}{k}2^k$. The *size* of a partial vector α, denoted $|\alpha|$, is the number of elements in α assigned 0 or 1. The *empty* partial vector, denoted λ, is the one in which all variables are assigned *.

The projection of f under a partial assignment α, denoted f_α, is the function obtained by "hardwiring" the values of the variables included in α. More precisely, given a total assignment β and a partial assignment α, let β_α denote the total assignment obtained by setting each variable whose value is not * in α to the value in α and each variable whose value is * in α to the value in β. Then, f_α is defined by $f_\alpha(\beta)=f(\beta_\alpha)$.

We are interested only in *projection-closed* representation classes of Boolean functions, i.e., ones for which given a representation for a Boolean function f and any partial vector α, the Boolean function f_α can also be represented in the class and, moreover, such a representation can be computed in polynomial time. We say that a projection-closed representation class is (*polynomial-time*) *sat-countable* if given a representation for f, the value of $\sharp f$ can be computed in time polynomial in the size of the representation and n, the total number of variables. If d is a representation of the function f, we use $|d|$ to denote the size of d. Where the context assures that there is no ambiguity, we treat a representation as synonymous with the Boolean function being represented.

The *error* $\mathrm{err}(f, f')$ of f with respect to another Boolean function f' defined over the same set of n variables is the total number of assignments α such that $f(\alpha)\neq f'(\alpha)$; moreover f is an ε-approximation of f' if

$$\frac{\mathrm{err}(f, f')}{2^n} \leqslant \varepsilon.$$

We consider the following projection-closed representation classes of Boolean functions in this paper.

1. *Decision trees*: A decision tree T is a binary tree where the leaves are labeled either 0 or 1, and each internal node is labeled with a variable. Given an assignment

$\alpha \in \{0,1\}^n$, $T(\alpha)$ is evaluated by starting at the root and iteratively applying the following rule, until a leaf is reached: let the variable at the current node be x_i; if the value of α at position i is 1 then branch right; otherwise branch left. If the leaf reached is labeled 0 (resp. 1) then $T(\alpha)=0$ (resp. 1). The *size* of a decision tree is its number of nodes.

2. *Branching programs* (*BPs*): A branching program is a directed acyclic graph with a unique node of in-degree 0 (called the *root*, and two nodes of out-degree 0 (called *leaves*)), one labeled 0 and the other labeled 1; each non-leaf node of the graph contains a variable, and has outdegree exactly two.
   If every variable appears at most once on any root–leaf path, then the branching program is called *read-once* (ROBP). Note that a decision tree can effectively be considered to be an ROBP. Assignments are evaluated following the same rule as for decision trees. The *height* of a BP is the length of the longest path from the root to a leaf node.
   An *ordered binary decision diagram* (OBDD) is an ROBP with the additional property that variables appear in the same order on any path from root to leaf.
3. *SAT-j DNF formulas*: DNF formulas in which every assignment satisfies at most j terms of the formula. (In our usage, j is a fixed constant.)
4. *μ-formulas*: Boolean formulas in which every variable occurs at most once.

Proposition 1. *Decision trees, OBDDs, ROBPs, BPs, SAT j-DNF formulas, and μ-formulas are projection-closed.*

Proof. For any BP, the projection under a partial vector α can be computed as follows: redirect incoming edges for each vertex labeled by a variable that is assigned a value in α to the left (resp. right) child of the vertex if that variable is assigned the value 0 (resp. 1) in α. Recursively delete vertices with no incoming edges. By using depth-first search, these steps can be achieved in linear time. Note that if the BP is a decision tree, OBDD, ROBP, or a h-height BP then the projection also belongs to the same class.

For SAT j-DNF and μ-formulas, the projection can be obtained by substituting the values for each assigned variable in α. A 0 in a DNF term will result in the deletion of that term, whereas a 1 results in the deletion of that variable from the term. In a μ-formula, appropriate Boolean algebra rules are applied to eliminate the 1's and 0's so obtained. This is accomplished in linear time in both cases. □

Proposition 2. *ROBPs and* O(log n)*-height BPs are sat-countable.*

Proof. The number of satisfying assignments of an ROBP f is computed as follows.

Traverse the nodes of f in reverse topological order. Let $f(x)$ denote the sub-ROBP rooted at a node x consisting of all vertices that can be reached from x and the edges joining them. When a node x is visited the fraction ϕ_x, $0 \leqslant \phi_x \leqslant 1$, of assignments of $f(x)$ that are satisfying assignments is computed as follows. If x is a

leaf then ϕ_x is the same (0 or 1) as the value of the leaf node; otherwise x is an internal node and ϕ_x is $(\phi_y + \phi_z)/2$, where y and z are the left and right children of x.

A simple inductive argument shows that on completion ϕ_r, where r is the root of the ROBP, is the fraction of satisfying assignments of f. Consequently, $\sharp f = \phi_r \cdot 2^n$, where n is the number of variables in f.

Next, let B be a $O(\log n)$ height BP representing a Boolean function f. First, construct a decision tree equivalent to f by "spreading out" B by creating a separate copy of a node whenever needed rather than sharing subfunctions as in a branching program. Such a decision tree may not immediately satisfy the "read-once" property, but it is easily converted into one by eliminating subtrees under duplicated variables along a path. The total number of nodes in this resultant decision tree is at most $2^{O(\log n)} = n^{O(1)}$. Finally, compute the number of satisfying assignments for the decision tree as described above for a ROBP. □

The next two propositions are not used in the paper; they are proved here simply in order to complete Table 1.

Proposition 3. *Decision trees can be tested for monotonicity in polynomial time.*

Proof. Let T be a given decision tree over n variables. It is convenient to extend the partial order $\geqslant$ defined over the Boolean lattice to the set of partial vectors by: $\alpha \geqslant \beta$ if for all i, $\beta_i = 1$ implies that $\alpha_i \neq 0$. For any partial vector β, we will say that $T(\beta) = 0$ (resp. 1) if for every total vector $\alpha \geqslant \beta$, $T(\alpha) = 0$ (resp. 1).

Each leaf node x in T determines a partial vector $p(x)$ based on the assignment to variables on the path from the root of T to the leaf node. Let us say that x is a *counterexample* to the monotonicity of T if there is a partial vector $\alpha \geqslant p(x)$ such that $T(\alpha) = 0$ and x has a value of 1. The essential observation is that T is monotone if and only if no leaf of T is a counterexample to its monotonicity.

It is easy to test for monotonicity of T using the above observation: for each leaf node x assigned the value 1, let $p'(x)$ be the partial vector obtained by setting to 1 only the variables in $p(x)$ assigned a 1 and leaving the remaining variables as *. If under the projection $p'(x)T$ is not identically 1, then x is a counterexample to monotonicity as demonstrated by any path to 0 in the projection. □

A Boolean function $f(v_1, v_2, \ldots, v_n)$ is *symmetric* if $f(v_1, v_2, \ldots, v_n) = f(v'_1, v'_2, \ldots, v'_n)$ for every permutation $(v'_1, v'_2, \ldots, v'_n)$ of $(v_1, v_2, \ldots, v_n)$.

Proposition 4. *ROBPs can be tested for symmetry in polynomial time.*

Proof. This proof is inspired by the central idea in [1]. Let f be any Boolean function over the set of variables $V = \{v_1, v_2, \ldots, v_n\}$ and let f^+ denote the set of assignments which satisfy f.

We first generalize f to be a *real*-valued function by treating V to be a set of real variables; more precisely redefine f by

$$f(v_1, v_2, \ldots, v_n) = \sum_{\alpha \in f^+} \left(\prod_{i:\alpha_i=1} v_i \cdot \prod_{i:\alpha_i=0} (1 - v_i) \right).$$

When the variables in V assume the values 0 and 1, the value of the redefinition coincides with the value of the Boolean function; so we do have a true generalization. As shown in [1], given a ROBP representation of the Boolean function f, the value of the real function on any real vector over V can be computed in linear time by visiting the ROBP in topological order.

Next, let $g(x) = f(x, x, \ldots, x)$ and let R_k, $0 \leqslant k \leqslant n$ be the set of assignments in f^+ with precisely k ones. Then,

$$\begin{aligned} g(x) &= f(x, x, \ldots, x) \\ &= \sum_{\alpha \in f^+} \left(\prod_{i:\alpha_i=1} x \cdot \prod_{i:\alpha_i=0} (1 - x) \right) \\ &= \sum_{k=0}^{n} \sum_{\alpha \in R_k} x^k (1 - x)^{n-k} \\ &= \sum_{k=0}^{n} |R_k| x^k (1 - x)^{n-k}. \end{aligned}$$

Now computing the values of $g(0), g(1), \ldots, g(n)$ as mentioned above by using the ROBP representation of f and treating $|R_k|$ as variables leads to the system of linear equations:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 1 \\ (1-2)^n & 2(1-2)^{n-1} & 2^2(1-2)^{n-2} & \cdots & 2^n \\ (1-3)^n & 3(1-3)^{n-1} & 3^2(1-3)^{n-2} & \cdots & 3^n \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ (1-n)^n & n(1-n)^{n-1} & n^2(1-n)^{n-2} & \cdots & n^n \end{pmatrix} \begin{pmatrix} |R_0| \\ |R_1| \\ |R_2| \\ \vdots \\ |R_n| \end{pmatrix} = \begin{pmatrix} g(0) \\ g(1) \\ g(2) \\ \vdots \\ g(n) \end{pmatrix}.$$

It is easily shown that the rank of the coefficient matrix is $n+1$; therefore the system admits a unique solution for $\langle |R_0|, |R_1|, \ldots, |R_n| \rangle$. Finally, observe that the Boolean function f is symmetric if and only if $|R_k|$ is either 0 or $\binom{n}{k}$ for all values of k, $0 \leqslant k \leqslant n$. □

From the above proof, it follows that we can decide symmetry for OBDDs and decision trees also in polynomial time.

Proposition 5. *SAT j-DNF formulas are sat-countable.*

Proof. Let us say that two terms t and t' are *conflicting* if t contains a literal l and t' contains a literal $\bar{l}$. The *consensus* of two non-conflicting terms t and t', denoted

tt' is the term obtained from the union of all the literals in t and t'; if t and t' are conflicting, then their consensus is 0.

The definition of a SAT-j DNF formula f implies that in every set $\{t_1, t_2, \ldots, t_{j+1}\}$ of $j+1$ terms of the formula, there must be at least two conflicting terms. Therefore, using the principle of inclusion and exclusion,

$$\sharp f = \sum_{t \in f} \sharp t - \sum_{t,t' \in f} \sharp(tt') + \sum_{t,t',t'' \in f} \sharp(tt't'') - \cdots .$$

Here, for any term t of k literals $\sharp t$ is simply 2^{n-k}. From the comment above, this sum needs to consider at most the consensus of j terms of f. For constant j, the total time $O(|f|^j)$ for the computation is a polynomial. □

Proposition 6. *μ-formulas are sat-countable.*

Proof. Let f be a μ-formula over a set of n variables. If f is the constant 1, then $\sharp f = 2^n$ and if f is the constant 0, then $\sharp f = 0$; if f is a term containing a single literal, then $\sharp f = 2^{n-1}$. Otherwise f can be written either as $f_1 f_2$ or $f_1 + f_2$, where f_1 and f_2 are μ-formulas over disjoint sets of n_1 and n_2 variables respectively. Then, it is easy to argue that $\sharp f = \sharp f_1 \cdot \sharp f_2$ if $f = f_1 f_2$, and that $\sharp f = 2^n - (2^{n_1} - \sharp f_1)(2^{n_2} - \sharp f_2)$ if $f = f_1 + f_2$. Recursive application of these rules ensures that $\sharp f$ can be computed in $O(|f| + n)$ time. □

3. Finding a decision tree approximation

The main result of this section is an algorithm for constructing a decision tree ε-approximation of any Boolean function f represented in a projection-closed sat-countable class. The heart of our algorithm is a procedure *FIND* which is a generalization of the dynamic programming method used in [8] for truth-table minimization of decision trees.

FIND works as follows. Given f, a Boolean function over n variables, a height parameter h and a size parameter m, it builds precisely one tree from the set $T_{\alpha,k}$, for each partial vector α of size at most h and for each k, $0 \leqslant k \leqslant m$. (Here, $T_{\alpha,k}$ is the set of all decision tree representations of the function f_α of size at most k and height at most $h - |\alpha|$ that have minimum error with respect to f_α and among all such trees, are of minimum size.) The desired approximation will therefore be the tree constructed for the set $T_{\lambda,w}$, where $w = \min\{m, 2^h - 1\}$.

The algorithm employs a two-dimensional array $P[\alpha, k]$ to hold a tree in $T_{\alpha,k}$. A tree in the P array will be represented by a triple of the form (*root*, *left subtree*, *right subtree*), unless it consists of a single-leaf node, in which case it will be represented by the leaf's value. For a partial vector α, the notation $\alpha.v \leftarrow 1$ ($\alpha.v \leftarrow 0$, respectively) denotes the partial vector obtained by extending α by setting the variable v to 1 (0, respectively).

```
FIND(boolFunctionRep f, int m, int h)
01. foreach α such that |α| ⩽ h do
02.   if ♯(f_α) > 2^{n−|α|−1} then P[α, 0] ← 1;
03.   else P[α, 0] ← 0;
04.
05. for i = h − 1 to 0 do
06.   foreach α such that |α| = i do
07.     foreach k = 1 to min{2^{h−|α|} − 1, m} do
08.       P[α, k] = P[α, k − 1];
09.       foreach variable v not used in α and each k′, k″ such that k′ + k″ + 1 = k do
10.           errV ← err((v, P[α.v ← 0, k′], P[α.v ← 1, k″]), f_α);
11.           if err(P[α, k], f_α) > errV then
12.               P[α, k] ← (v, P[α.v ← 0, k′], P[α.v ← 1, k″]);
13.           else if err(P[α, k], f_α) = errV then
14.               if (|P[α.v ← 0, k′]| + |P[α.v ← 1, k″]| + 1) < |P[α, k]| then
15.                 P[α, k] ← (v, P[α.v ← 0, k′], P[α.v ← 1, k″]);
16. output P[λ, m].
end
```

Fig. 1. Algorithm *FIND*.

Lemma 1. *Algorithm FIND is correct, i.e., given a sat-countable representation of a Boolean function f, a height parameter h, and a size parameter m, FIND outputs a decision tree T' of height at most h and size at most m such that among all such decision trees, $\mathrm{err}(T', f)$ is minimum; if there is more than one decision tree with the same minimum error, then $|T'|$ is of minimum size among these trees.*

Proof. We show by induction on $l_\alpha = h - |\alpha|$ and k that $P[\alpha, k]$ is a tree in $T_{\alpha,k}$, for all $0 \leqslant |\alpha| \leqslant h$ and $0 \leqslant k \leqslant \min\{2^{h-|\alpha|} - 1, m\}$. For $k = 0$ and any α, the tree must be a leaf with value 0 or 1, depending on which value yields the minimum error relative to f_α. Lines 2 and 3 of Algorithm *FIND* examine the hypercube corresponding to f_α and determine whether the majority of assignments are 0 or 1. This is also true for any α such that $l_\alpha = 0$ since $l_\alpha = 0 \Rightarrow |\alpha| = h \Rightarrow k = 0$.

Assume that $P[\beta, k']$ has been correctly computed for all β such that $l_\beta < l_\alpha$ and all k' in $[0, \min\{2^{h-|\beta|} - 1, m\}]$. Also assume that all $P[\alpha, k']$ have been correctly computed for all k' in $[0, k-1]$. We show that *FIND* causes a tree in $T_{\alpha,k}$ to be placed in $P[\alpha, k]$. If the size of the trees in $T_{\alpha,k}$ is less than k, then, from the induction hypothesis, $P[\alpha, k]$ is initialized to a tree in $T_{\alpha,k}$ in line 8. Lines 9–15 cannot then modify $P[\alpha, k]$ and the algorithm is correct. Therefore, let the size of the trees in $T_{\alpha,k}$ be exactly k. Let *Opt* be any tree in $T_{\alpha,k}$ and let v be its root. Now v must be a variable that is not assigned a value in α. Let the sizes of *Opt*'s left and right subtrees be k_0 and k_1, respectively. Observe that k_0 and k_1 are one of the (k', k'') pairs examined in line 9.

From the induction hypothesis, $\mathrm{err}(P[\alpha.v \leftarrow 0, k_0], f_{\alpha.v\leftarrow 0}) \leqslant \mathrm{err}$(Left subtree of *Opt*, $f_{\alpha.v\leftarrow 0}$) and $\mathrm{err}(P[\alpha.v \leftarrow 1, k_1], f_{\alpha.v\leftarrow 1}) \leqslant \mathrm{err}$ (Right subtree of *Opt*, $f_{\alpha.v\leftarrow 1}$). Since the error of a tree is the sum of the errors of its two subtrees, the algorithm finds a tree for $P[\alpha, k]$ which has error at most that of *Opt* and size at most that of *Opt*. The lemma follows. □

Lemma 2. *Let $p(|f|, n)$ denote the time complexity for computing the number of satisfying assignments of an arbitrary projection of a given sat-countable function f. The time complexity of FIND is $\mathrm{O}(n^{\mathrm{O}(h)}(m^2 + p(|f|, n)))$.*

Proof. Since f is a sat-countable representation, the time required by line 2 is $\mathrm{O}(p(|f|, n))$. The number of partial vectors examined in line 1 is $\sum_{i=0}^{h} \binom{n}{i} 2^i = \mathrm{O}(n^{\mathrm{O}(h)})$. Thus, lines 1–4 take $\mathrm{O}(p(|f|, n)n^{\mathrm{O}(h)})$ time. Lines 5 and 6 cause the same $\mathrm{O}(n^{\mathrm{O}(h)})$ partial vectors to be examined. The variable k (line 7) takes on at most m values, there are at most n possibilities for v and m possible combinations of k' and k'' in line 9.

The complexity of lines 10–15 is dominated by O(1) error computations between a decision tree T in P and the sat-countable function f_α. Each such error computation can be implemented as follows. For any leaf node x in T, let β be the partial vector corresponding to the evaluation path in T leading up to x. The contribution to the total error of the partial vector β is then either $\sharp((f_\alpha)_\beta)$ if the leaf x has value 0 and $2^{n-|\alpha|-|\beta|} - \sharp((f_\alpha)_\beta)$ if it has value 1. The total error $\mathrm{err}(T, f_\alpha)$ is obtained by summing the errors computed in this fashion over each leaf of T. The complexity of this computation is bounded by $\mathrm{O}(P(|f|, n)m)$ and that of lines 5–15 and hence Algorithm *FIND* is bounded by $\mathrm{O}(n^{\mathrm{O}(h)}m^3 p(|f|, n))$. As is common in dynamic programming algorithms, memorizing helps to reduce the overall complexity a little. Observe that the complexity of error computation can be reduced by maintaining a second two-dimensional array E each of whose elements contains the error of the corresponding element in array P. First $E[\alpha, 0]$ can be computed in $\mathrm{O}(p(|f|, n))$ time in lines 2 and 3. Then the remaining $E[\alpha, k]$'s are computed every time $P[\alpha, k]$ is updated in O(1) time by simply summing the error of the left and right subtrees of $P[\alpha, k]$. With this time-saving modification, the time complexity becomes $\mathrm{O}((p(|f|, n) + m^2)n^{\mathrm{O}(h)})$. □

Lemma 3. *Let T be an m-node decision tree. Then there exists a decision tree T^* of height at most $h = \log((m+1)/4\varepsilon)$ and at most m nodes such that T^* is an ε-approximation of T.*

Proof. Restrict T to height h by converting any node x at level h to either 0 or 1 depending on whether there are more 0s or 1s, respectively, in the hypercube defined by the path leading to x. Call this tree T^*. Clearly, T^* has no more than m nodes and the error of T^* is confined to the hypercubes of the converted nodes x at level h in the original tree. Since there are at most $\lceil m/2 \rceil$ such nodes and the error of each node is at most 2^{n-h-1}, it follows that T^* is a $\lceil m/2 \rceil \cdot 2^{n-h-1}/2^n \leqslant (m+1)/4.2^h$-approximation of T. Substituting $h = \log((m+1)/4\varepsilon)$ now yields the desired result. □

Theorem 1. *Given a sat-countable Boolean function representation f whose smallest* decision tree *representation has at most m nodes and any error parameter ε, we can find a decision tree T' of at most m nodes which ε-approximates f in time polynomial in $|f|$ and $n^{\log m/\varepsilon}$.*

Proof. Given f, we use the standard doubling trick to determine in $O(\log m^*)$ iterations of the algorithm the least value m^* such that $FIND(f, m^*, \log((m^*+1)/4\varepsilon))$ returns a decision tree which ε-approximates f. By Lemma 3, m^* is at most m, the size of the smallest decision tree which can represent f. The correctness and time complexity then follow from Lemmas 1 and 2, respectively. □

4. Learning decision trees under the uniform distribution

We show that the algorithm of the previous section can be extended to learn decision trees under the uniform distribution. As we remarked in the introduction, this means that, given access to a uniformly distributed sample of evaluations of a Boolean function f an error parameter ε and a confidence parameter δ, our algorithm will output a decision tree T of at most m nodes, where m is the least number of nodes needed to represent f as a decision tree and such that T ε-approximates f with confidence at least $1-\delta$. The algorithm takes time polynomial in $n^{\log(m/\varepsilon)}$ and $\log(1/\delta)$, i.e., it is a quasi-polynomial time algorithm. However, the sample-complexity of the algorithm is only a modest polynomial in the parameters $m, \log n, \log(1/\delta)$ and $\log(1/\varepsilon)$.

We use the following additional terminology to prove the results of this section. Let $\mathcal{T}_{m,h,n}$ denote the class of decision trees over n variables that have height at most h and size at most m. For any decision tree T, let $T^*(h)$ be the tree of height h obtained from T by converting all non-leaf nodes of depth h in T to leaf nodes with classification 0 or 1, depending on whether the majority of the assignments in the corresponding hypercube of f are classified as 0 or 1, respectively.

Recall that for any two Boolean functions, f_1, f_2 over n variables, $\mathrm{err}(f_1, f_2)$ denotes the number of assignments α for which $f_1(\alpha) \neq f_2(\alpha)$; by extension, if S is a sample of classified examples of the form $\langle \alpha, b \rangle$ where α is an assignment and $b \in \{0,1\}$, then $\mathrm{err}(S, f) = \mathrm{err}(f, S)$ is the number of examples in S of the form $\langle \alpha, b \rangle$ where $f(\alpha) \neq b$.

We need the following well-known inequalities.

Proposition 7 (Chernoff bounds). *Let $X_1, X_2, \ldots, X_r$ denote the outcomes of r identical, independent Bernoulli trials with* $\mathrm{Prob}[X_i = 1] = p$, *for all i, $1 \leqslant i \leqslant r$.*

Let $R = \sum_{i=1}^{r} X_i$. Then $E[R] = pr$ and for $0 \leqslant \gamma \leqslant 1$,

- $\mathrm{Prob}[R \geqslant (p+\gamma)r] \leqslant e^{-2r\gamma^2}$, *and*
- $\mathrm{Prob}[R \leqslant (p-\gamma)r] \leqslant e^{-2r\gamma^2}$.

Lemma 4. *Given a sample S of classified examples of a Boolean function of the form $\langle \alpha, b \rangle$ where α is an assignment and $b \in \{0,1\}$, a height parameter h, and a*

size parameter m, a decision tree D of height at most h and size at most m can be computed such that among all such decision trees, $\mathrm{err}(D,S)$ is minimum, and among all such minimum error trees, D has minimum size. The computation requires $O(n^{O(h)}(m^2+|S|))$.

Proof. Let S_α denote the assignments in S that extend the partial assignment α. For a given α, S_α can be computed in $O(|S|n)$ time. Modify the condition of Line 2 of Algorithm *FIND* so that number of assignments of S_α whose values are 1 and 0 are compared. The modified Line 2 takes $O(|S|n)$ time. All references to f_α (lines 10, 11, and 13) are replaced by S_α. Error computations can be carried out as described in the proof of Lemma 2. Each error computation takes $O(|S|m)$ time. Since the rest of the algorithm is unchanged, the complexity is obtained by replacing $p(|f|,n)$ by $|S|$. Note that this is also true of the modified algorithm proposed in the proof of Lemma 2. Correctness follows from Lemma 1. □

Theorem 2. *Given*

- *a uniformly distributed sample S of size*

$$r=\frac{8}{\varepsilon^2}\left(m\ln(4n)+\ln\left(\frac{4}{\delta}\right)\right)$$

of examples of an m-node decision tree T over n variables,
- *an error parameter ε, $0<\varepsilon<1$, and*
- *a confidence parameter δ, $0<\delta<1$,*

we can find a decision tree D in $\mathcal{T}_{m,h,n}$ with $h=\log((m+1)/2\varepsilon)$ in time $O(rm^2n^{O(h)})$ such that with confidence at least $1-\delta$, the error of D in approximating T is at most ε, i.e.,

$$\mathrm{Prob}[\mathrm{err}(D,T)<\varepsilon]\geqslant 1-\delta.$$

Proof. We execute algorithm *FIND* modified to deal with a sample S as described in Lemma 4 with the parameters m and h as above. Let $\varepsilon'=(m+1)2^{-h}/4=\varepsilon/2$.

Call a decision tree T' in $\mathcal{T}_{m,h,n}$ *bad* if $\mathrm{err}(T',T)\geqslant\varepsilon$. For any *fixed* bad decision tree T',

$$\begin{aligned}
&\mathrm{Prob}[\textit{FIND} \text{ outputs } T']\\
&\leqslant \mathrm{Prob}[T'\in\mathcal{T}_{m,h,n} \text{ and has least error over sample } S]\\
&\leqslant \mathrm{Prob}\left[\mathrm{err}(S,T')\leqslant\mathrm{err}(S,T^*(h))\right]\\
&\leqslant \mathrm{Prob}\left[\mathrm{err}(S,T')\leqslant\frac{\varepsilon'+\varepsilon}{2}|S| \text{ or } \mathrm{err}(S,T^*(h))\geqslant\frac{\varepsilon'+\varepsilon}{2}|S|\right]\\
&\leqslant \mathrm{Prob}\left[\mathrm{err}(S,T')\leqslant\frac{\varepsilon'+\varepsilon}{2}|S|\right]+\mathrm{Prob}\left[\mathrm{err}(S,T^*(h))\geqslant\frac{\varepsilon'+\varepsilon}{2}|S|\right]\\
&\leqslant e^{-2r[(\varepsilon-\varepsilon')/2]^2}+e^{-2r[(\varepsilon-\varepsilon')/2]^2}=2e^{-(r\varepsilon^2)/8}.
\end{aligned}$$

Here, the last inequality follows from Chernoff bounds applied to the number of errors in S of the trees T' and $T^*(h)$.

Now the probability p that *FIND* outputs *any* bad tree T' in $\mathscr{T}_{m,h,n}$ is certainly at most $|\mathscr{T}_{m,h,n}|\cdot 2e^{-r\varepsilon^2/8}$. The number of binary trees on at most m nodes is at most $2\cdot 4^m$ and so the number of decision trees of at most m nodes is at most $2\cdot(4n)^m$, which also is an upper bound on $\mathscr{T}_{m,h,n}$. Consequently, for our choice of r in the proposition (and after a little bit of arithmetic), the probability p turns out to be at most δ. □

5. Conclusions

Given a sat-countable representation of a Boolean function or a uniformly distributed sample of evaluations of a Boolean function, this paper presents a quasi-polynomial algorithm for computing a decision tree of smallest size that approximates this function. Is it possible to achieve this in polynomial time? Failing this, is it possible to obtain a decision tree whose size is within a polynomial factor of the smallest approximating decision tree in polynomial time?

Finding a decision tree of smallest size equivalent to a given one is NP-hard [14]. This opens the question of whether at least a polynomial approximation of the smallest equivalent decision tree is possible in polynomial time. The ideas in this paper do not seem enough to answer this question, but there is some hope that combining these ideas with the results of Ehrenfeucht and Haussler [6] will work. As a matter of fact, their results can already be used to give a *quasi*-polynomial approximation to the smallest decision tree equivalent to *any* projection-closed representation which allows testing for tautology and satisfiability in polynomial time. This is done in the following way.

We consider the sample S in the Ehrenfeucht and Haussler algorithm to be all 2^n assignments. However, we avoid using time polynomial in the sample size, by noting that the operations on the sample in the algorithm consist only of:

1. checking if all assignments in S evaluate to either 0 or 1, and
2. computing a new sample S' obtained by projecting given variable to 0 or 1.

Doing these operations in time polynomial in the given representation converts their algorithm into one whose complexity has an added factor of the form $O(n^{O(r)})$, where r is the smallest rank of any equivalent decision tree; since r cannot exceed $O(\log m)$, where m is the size of the smallest equivalent decision tree, we get the desired quasi-polynomial approximation.

Finally, can the ideas of this paper be combined with those of Ehrenfeucht and Haussler to properly learn decision trees under arbitrary distributions with or without membership queries?

Acknowledgements

A preliminary version of this paper appeared in COLT 2000. We thank an anonymous COLT referee who suggested the sharper bound on $\mathscr{T}_{m,h,n}$ which led to an improvement in the sample complexity in Theorem 2.

References

[1] M. Blum, A. Chandra, M. Wegman, Equivalence of free Boolean graphs can be decided probabilistically in polynomial time, Inform. Process. Lett. 10 (1980) 80–82.
[2] A. Blumer, A. Ehrenfeucht, D. Haussler, M.K. Warmuth, Occam's razor, Inform. Process. Lett. 24 (1987) 377–380.
[3] N.H. Bshouty, Exact learning Boolean functions via the monotone theory, Inform. and Comput. 123 (1) (1995) 146–153.
[4] N.H. Bshouty, Y. Mansour, Simple learning algorithms for decision trees and multivariate polynomials, Proc. Foundations of Computer Science (FOCS), 1995, pp. 304–311.
[5] C. Domingo, T. Tsukiji, O. Watanabe, Partial Occam's razor and its applications, Inform. Process. Lett. 64 (4) (1997) 179–185.
[6] A. Ehrenfeucht, D. Haussler, Learning decision trees from random examples, Inform. and Comput. 82 (3) (1989) 231–246.
[7] M.R. Garey, D.S. Johnson, Computers and Intractability, a Guide to the Theory of NP-completeness, W.H. Freeman and Co., New York, 1979.
[8] D. Guijarro, V. Lavin, V. Raghavan, Exact learning when irrelevant variables abound, Inform. Process. Lett. 70 (1999) 233–239.
[9] L. Hyafil, R. Rivest, Constructing optimal binary decision trees is NP-complete, Inform. Process. Lett. 5 (1976) 15–17.
[10] C. Kenyon, V. King, On Boolean decision trees with faulty nodes, Random Struct. Algorithms 5 (3) (1994) 453–464.
[11] J.R. Quinlan, Induction of decision trees, Mach. Learning 1 (1) (1986) 81–106.
[12] J.R. Quinlan, Learning decision tree classifiers, Comput. Surveys 28 (1) (1996) 71–72.
[13] J. Simon, On some central problems in computational complexity, Ph.D. Thesis, Cornell University, Ithaca, New York, 1975.
[14] H. Zantema, H. Bodlaender, Finding small equivalent decision trees is hard, Tech. Report UU-CS-1999-31, Department of Computer Science, Utrecht University, Utrecht, The Netherlands, 1999.