Electronic Colloquium on Computational Complexity, Report No. 54 (2002)

Minimization of Decision Trees is Hard to Approximate

Detlef Sieling*
FB Informatik, LS 2, Univ. Dortmund,
44221 Dortmund, Fed. Rep. of Germany
sieling@ls2.cs.uni-dortmund.de

Abstract

Decision trees are representations of discrete functions with widespread applications in, e.g., complexity theory and data mining and exploration. In these areas it is important to obtain decision trees of small size. The minimization problem for decision trees is known to be NP-hard. In this paper the problem is shown to be even hard to approximate up to any constant factor.

1. Introduction

Decision trees are one of the simplest representations of discrete functions. Roughly, a decision tree queries properties of the given input in an adaptive way, i.e., the queries may depend on the results of previous queries. Eventually, the result has to be output merely depending on the results of the queries. We are interested in decision trees that may query the values of the input variables; however, more general queries like comparison of numbers also lead to useful representations. Furthermore, we focus on decision trees for Boolean functions.

More formally, a decision tree consists of internal nodes and leaves. Each node except the root has exactly one incoming edge. Each internal node is labeled by a Boolean variable and has an outgoing 0-edge and an outgoing 1-edge. Each leaf is labeled by 0 or 1. The evaluation of the represented function starts at the root. At each internal node labeled by x_i the computation proceeds at the node reached via the outgoing c-edge if x_i takes the value c. Eventually, a leaf is reached and the label of the leaf is the value computed. The path from the root to the leaf that is chosen for the input x is also called the computation path for x. In the drawings 0-edges are shown as dashed lines and 1-edges as solid lines. It is easy to verify that the example of a decision tree in Fig. 1 represents the function $f(x_1, x_2, x_3) = \bar{x}_1 x_2 \vee x_1 \bar{x}_2 x_3$.

The basic complexity measures of decision trees are their depth and their size. The depth corresponds to the computation time and the size corresponds to the size of the representation by a decision tree. In complexity theory the relation between the size and the depth of a decision

*The author was supported by DFG grant Si 577/1.

ISSN 1433-8092

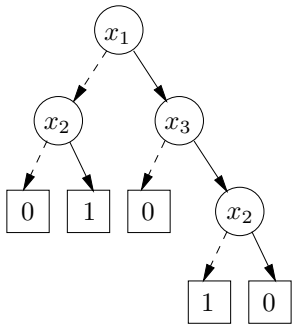

Figure 1: An example of a decision tree for $f(x_1, x_2, x_3) = \bar{x}_1 x_2 \vee x_1 \bar{x}_2 x_3$.

tree and various complexity measures for Boolean functions has been explored. E.g., sensitivity and certificate complexity are polynomially related to the depth of decision trees; for an overview see Buhrman and de Wolf [1]. The relation between the size of decision trees and the size of DNFs and CNFs was considered by Ehrenfeucht and Haussler [2] and Jukna, Razborov, Savický and Wegener [10]. Another application of decision trees (with generalized queries) is the proof of the $\Omega(n \log n)$ lower bound for comparison-based sorting of n items and for geometric problems (Knuth [11], Preparata and Shamos [13]).

Much research has been done in the automatic construction of decision trees from data. The goals are to reduce the representation size of the data as well as to deduce properties from the data, e.g., to discover whether the data can be partitioned into disjoint classes of objects or to find methods to classify new data with as few queries as possible. This has many applications, e.g., in computational learning theory (see below), in biology (classification of unknown species), machine fault location or questionnaire design. For an overview we refer to Murthy [12].

We discuss the scenario of computational learning theory. There the goal is to determine an unknown function, which is called the concept, from a set of examples, i.e., inputs together with the values the function takes for these inputs. One tries to find a function that coincides with the given examples and has, e.g., a small-size decision tree. If the decision tree is small, it is likely that the function represented by the decision tree coincides with the concept also on many inputs not given as examples. In fact, decision trees are the core of learning systems, see e.g. Quinlan [14]. The theoretically best learning algorithm for decision trees is due to Ehrenfeucht and Haussler [2]. The question for PAC-learnability of decision trees was considered by Hancock, Jiang, Li and Tromp [8]. They show the problem of finding a minimal decision tree consistent with a given set of examples to be hard to approximate, which implies the nonlearnability of decision trees in the PAC-learning model. We recall that a polynomial time approximation algorithm for some minimization problem always has to compute a legal solution; however, the size of the output may be larger than the minimum size by some constant factor, which we call the performance ratio.

It is convenient to define the size of a decision tree as the number of its leaves. The considered problem is defined as follows:

MinDT

Instance: A decision tree representing some function $f : \{0,1\}^n \to \{0,1\}$.
Problem: Compute a decision tree for f of minimum size.

For decision trees with generalized queries the corresponding minimization problem was shown to be NP-hard by Hyafil and Rivest [9]. The stronger NP-hardness result for MinDT was shown by Zantema and Bodlaender [17]. They also raise the question for an approximation algorithm for MinDT. We solve their question by proving the following result.

Theorem 1: *If there is a polynomial time approximation algorithm with a constant performance ratio for MinDT, then P = NP.*

Zantema and Bodlaender [17] prove their NP-hardness result by a reduction from the independent set problem for bounded-degree graphs. The reduction seems not to be approximation preserving. Hancock, Jiang, Li and Tromp [8] prove the hardness of approximating a minimum decision tree consistent with a set of examples by a reduction from the problem of computing a shortest monomial consistent with a set of examples combined with a self-improving property of minimization algorithms for decision trees. However, the minimization of decision trees representing a function describing a single monomial is trivial such that their reduction does not work for MinDT. On the other hand, a similar self-improving property is also helpful for us to prove a stronger nonapproximability result.

A representation of Boolean functions related to decision trees are Binary Decision Diagrams (BDDs), where the condition of fan-in 1 of each node is relaxed. This allows subgraphs of the representation to be shared. A nonapproximability result for minimizing so-called Ordered BDD (OBDDs) was shown in Sieling [15]; however, the proof of the nonapproximability result is based on the sharing of subgraphs such that it cannot be adapted to MinDT.

A dynamic programming algorithm for exact minimization of decision trees with an exponential run time was presented by Guijarro, Lavín and Raghavan [6]. Their algorithm is similar to an algorithm for minimizing OBDDs due to Friedman and Supowit [4].

Our hardness result is proved in three steps. First, we define a covering problem and provide an approximation preserving reduction from a variant of satisfiability (Section 3). This proves that polynomial time approximation schemes for the covering problem imply P $=$ NP. Afterwards, we provide an approximation preserving reduction to MinDT (Section 4). Finally, we use a self-improving property of approximation algorithms for MinDT to show that polynomial time approximation algorithms for MinDT can be converted to polynomial time approximation schemes for MinDT (Section 5).

2. Preliminaries

For the definitions of approximation algorithms and approximation schemes we follow Garey and Johnson [5]. Let Π be a minimization problem. For an instance I for Π let $OPT(I)$ denote the value of an optimal solution of I. Let A be an algorithm that for given I only computes legal solutions for I and let $A(I)$ denote the value of the solution computed by A on the instance I. The algorithm A is called approximation algorithm with the performance ratio R iff for all instances I it holds that $A(I)/OPT(I) \leq R$. A polynomial time approximation scheme for Π is an algorithm B with an extra input ε. For all instances I and all $\varepsilon > 0$ the algorithm B has to compute an output with a value $B(I)$ such that $B(I)/OPT(I) \leq 1 + \varepsilon$. The run time of B has to be polynomial in the length of I but may arbitrarily depend on ε.

Our nonapproximability result is based on a nonapproximability result for the satisfiability problem. It is convenient to consider the following special variant of satisfiability. We recall that an algorithm for a promise problem does not have to check whether the promise is fulfilled. If the promise is fulfilled, the algorithm has to work correctly, and otherwise it may behave arbitrarily.

εGap3SAT5

Instance: A set $X = \{x_1, \ldots, x_n\}$ of variables and a set $C = \{C_1, \ldots, C_m\}$ of clauses, where each clause consists of exactly 3 literals, no variable occurs more than once in each clause and each variable occurs exactly five times.

Promise: If (X, C) is not satisfiable, for each setting of the variables at least $\varepsilon|C|$ clauses are not satisfied.

Problem: Is there a satisfying assignment to the variables?

We remark that from the requirements on the instances of εGap3SAT5 the equality $m = 5n/3$ easily follows. The following hardness result is due to Feige [3].

Theorem 2: *For some $\varepsilon > 0$ the problem εGap3SAT5 is NP-hard.*

Let T be a decision tree. Then $|T|$ denotes the number leaves of T. The subtree of a node v of T is the decision tree whose root is v and which consists of all nodes reachable from v. We call T reduced if on each path from the root to a leaf each variable is queried at most once. It is well-known that decision trees can be reduced in linear time by removing multiple tests of variables, see e.g. Wegener [16].

3. A Nonapproximability Result for SetSelection

We first define a covering problem, which we call SetSelection, and prove afterwards that a polynomial time approximation scheme for SetSelection implies P = NP. This is an intermediate step in our proof of the hardness of MinDT. The restrictions on the instances are helpful to make the reductions easier.

SetSelection

Instance: A finite set P, sets $D_1, \ldots, D_r \subseteq P$ and sets $S_1, \ldots, S_l \subseteq \{D_1, \ldots, D_r\}$ with the following properties:

- **(S1)** $\forall j \in \{1, \ldots, r\} : |D_j| = 2$ and $\forall i \in \{1, \ldots, l\} : |S_i| \leq 6$.
- **(S2)** For each $i, j \in \{1, \ldots, r\}$, $i \neq j$, there are $k, k' \in \{1, \ldots, l\}$ such that $D_i \in S_k$, $D_j \notin S_k$, $D_i \notin S_{k'}$, $D_j \in S_{k'}$.
- **(S3)** For each $i \in \{1, \ldots, r\}$ there are $k, k' \in \{1, \ldots, l\}$, $k \neq k'$, such that $D_i \in S_k \cap S_{k'}$.
- **(S4)** For each $p \in P$ and each S_i it holds that p is contained in none, exactly one or in all sets in S_i.

Problem: Compute a sequence $S_{i(1)}, \ldots, S_{i(k)} \in \{S_1, \ldots, S_l\}$ such that no set occurs more than once in the sequence, $\forall j \in \{1, \ldots, r\} : D_j \in S_{i(1)} \cup \cdots \cup S_{i(k)}$ and the goal function

$$\sum_{q=1}^{k} \left| \bigcup_{D \in S_{i(q)} \setminus (S_{i(1)} \cup \cdots \cup S_{i(q-1)})} D \right|$$

is minimal.

SetSelection is a covering problem where the objects $D_1, \ldots, D_r$ have to covered by the sets $S_1, \ldots, S_l$. However, the goal function also depends on the order of the covering sets. An object D_i contributes to the goal function only for its first occurrence. "Similar" sets D_j in the same $S_{i(\cdot)}$ contribute less to the goal function since the contribution only depends on the size of the union of the $D_j \in S_{i(\cdot)}$. We mention that the task to compute a permutation of the sets $S_1, \ldots, S_l$ minimizing the goal function is equivalent to the above definition since appending the remaining sets to the covering sequence does not change the value of the goal function. In the following we use the terms D-sets and S-sets to distinguish the different types of sets in instances for SetSelection.

Theorem 3: *If there is a polynomial time approximation scheme for SetSelection, then P = NP.*

Proof: Let $\varepsilon > 0$ be chosen such that εGap3SAT5 is NP-hard. We provide a reduction from εGap3SAT5. Let (X, C) be an instance of εGap3SAT5. Let $Y(i)$ denote the set of indices of clauses containing x_i or $\bar{x}_i$. By the definition of εGap3SAT5 it holds that $|Y(i)| = 5$. We now construct an instance for SetSelection. The construction is depicted in Fig. 2. Let

$$P = \{a_i, b_i \mid 1 \leq i \leq n\} \cup \{c_j, d_j \mid 1 \leq j \leq m\}.$$

For each $i \in \{1, \ldots, n\}$ there is a D-set $D_i = \{a_i, b_i\}$ and for each $j \in Y(i)$ there are D-sets $D^j_{x_i} = \{b_i, c_j\}$ and $D^j_{\bar{x}_i} = \{a_i, c_j\}$. Furthermore, for each $j \in \{1, \ldots, m\}$ there is the D-set $D_{C_j} = \{c_j, d_j\}$. For each $i \in \{1, \ldots, n\}$ there are the S-sets

$$S_{x_i} = \{D_i\} \cup \{D^j_{x_i} | j \in Y(i)\} \quad \text{and} \quad S_{\bar{x}_i} = \{D_i\} \cup \{D^j_{\bar{x}_i} | j \in Y(i)\}.$$

Let $x_{u(j)}$, $x_{v(j)}$ and $x_{w(j)}$ be the variables that occur (negated or not) in the jth clause C_j. For each clause C_j there are exactly seven S-sets according to the following case distinction.

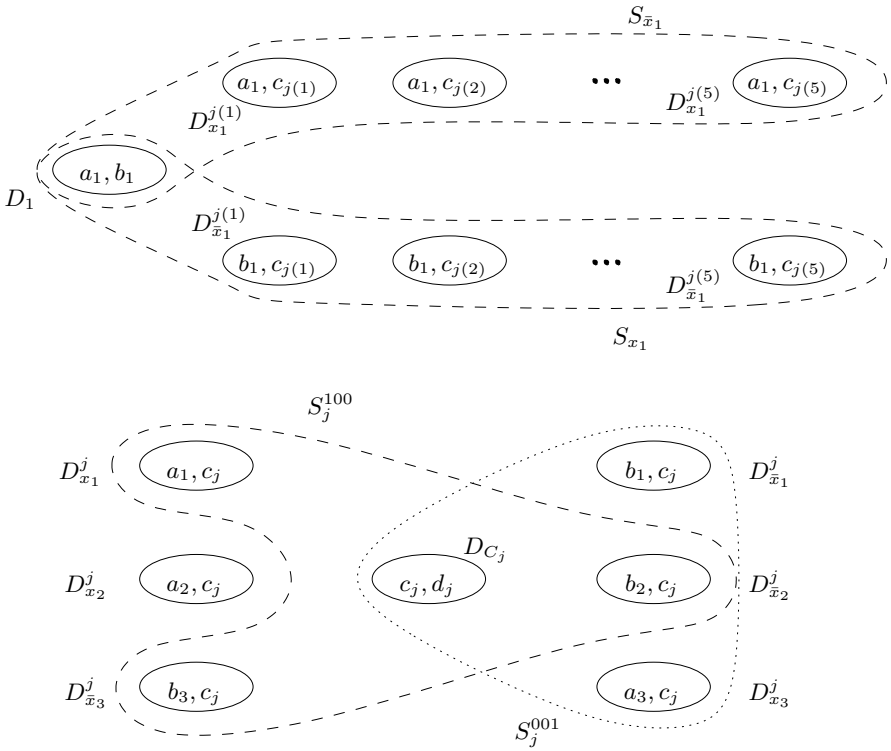

Figure 2: The D- and S-sets for the variable x_1 and the clause $C_j = x_1 \vee x_2 \vee \bar{x}_3$. We assume that x_1 occurs in the clauses $C_{j(1)}, \ldots, C_{j(5)}$.

$$
\begin{aligned}
S_j^{000} &= \{D_{C_j}, D^j_{x_{u(j)}}, D^j_{x_{v(j)}}, D^j_{x_{w(j)}}\}, && \text{if } C_j \neq x_{u(j)} \vee x_{v(j)} \vee x_{w(j)},\\
S_j^{001} &= \{D_{C_j}, D^j_{x_{u(j)}}, D^j_{x_{v(j)}}, D^j_{\bar{x}_{w(j)}}\}, && \text{if } C_j \neq x_{u(j)} \vee x_{v(j)} \vee \bar{x}_{w(j)},\\
S_j^{010} &= \{D_{C_j}, D^j_{x_{u(j)}}, D^j_{\bar{x}_{v(j)}}, D^j_{x_{w(j)}}\}, && \text{if } C_j \neq x_{u(j)} \vee \bar{x}_{v(j)} \vee x_{w(j)},\\
S_j^{011} &= \{D_{C_j}, D^j_{x_{u(j)}}, D^j_{\bar{x}_{v(j)}}, D^j_{\bar{x}_{w(j)}}\}, && \text{if } C_j \neq x_{u(j)} \vee \bar{x}_{v(j)} \vee \bar{x}_{w(j)},\\
S_j^{100} &= \{D_{C_j}, D^j_{\bar{x}_{u(j)}}, D^j_{x_{v(j)}}, D^j_{x_{w(j)}}\}, && \text{if } C_j \neq \bar{x}_{u(j)} \vee x_{v(j)} \vee x_{w(j)},\\
S_j^{101} &= \{D_{C_j}, D^j_{\bar{x}_{u(j)}}, D^j_{x_{v(j)}}, D^j_{\bar{x}_{w(j)}}\}, && \text{if } C_j \neq \bar{x}_{u(j)} \vee x_{v(j)} \vee \bar{x}_{w(j)},\\
S_j^{110} &= \{D_{C_j}, D^j_{\bar{x}_{u(j)}}, D^j_{\bar{x}_{v(j)}}, D^j_{x_{w(j)}}\}, && \text{if } C_j \neq \bar{x}_{u(j)} \vee \bar{x}_{v(j)} \vee x_{w(j)},\\
S_j^{111} &= \{D_{C_j}, D^j_{\bar{x}_{u(j)}}, D^j_{\bar{x}_{v(j)}}, D^j_{\bar{x}_{w(j)}}\}, && \text{if } C_j \neq \bar{x}_{u(j)} \vee \bar{x}_{v(j)} \vee \bar{x}_{w(j)}.
\end{aligned}
$$

This means the following: If in C_j the variables $x_{u(j)}$, $x_{v(j)}$ and $x_{w(j)}$ occur (negated or not), there is the set $S_j^{z_{u(j)}z_{v(j)}z_{w(j)}}$ iff C_j is satisfied by the assignment z. In Fig. 2 the set S_j^{100} for the clause $C_j = x_1 \vee x_2 \vee \bar{x}_3$ is indicated by a dashed line. The set S_j^{001}, which is indicated by a dotted line, is the set that does *not* exist for this clause.

It is easy to verify that the constructed sets form a valid instance of SetSelection and that this instance can be computed from (X, C) in polynomial time. Theorem 3 follows from two lemmas.

Lemma 4: *If (X, C) is satisfiable, the constructed instance of SetSelection has a solution with the value $46n/3$.*

Lemma 5: *If (X, C) is not satisfiable, each solution of the constructed instance of SetSelection has a value of at least $46n/3 + \varepsilon n/3$.*

We first show that the lemmas imply Theorem 3. Let us assume that there is a polynomial time approximation scheme A for SetSelection. Then we can construct a polynomial time algorithm for εGap3SAT5 in the following way. Let (X, C) be an instance for εGap3SAT5. We construct in polynomial time an instance I for SetSelection as described above and apply the polynomial time approximation scheme A to I for the performance ratio $1 + \delta$, where $\delta < \varepsilon/46$. If (X, C) is satisfiable, by Lemma 4 there is a solution for I with a value of at most $46n/3$ and, hence, the value of the output of A is bounded above by $(1 + \delta) \cdot 46n/3$. If (X, C) is not satisfiable, by Lemma 5 all solutions of the constructed instance of SetSelection have a value of at least $46n/3 + \varepsilon n/3$. Hence, also the output of A has at least this value, which, by the choice of δ, is larger than $(1+\delta)\cdot 46n/3$. Hence, we can distinguish satisfiable and non-satisfiable instances of εGap3SAT5 in polynomial time, which by Theorem 2 implies P = NP. Altogether, the proofs of Lemmas 4 and 5 imply Theorem 3. □

Proof of Lemma 4: Let $z_1, \ldots, z_n$ be a satisfying assignment for (X, C). We choose the following S-sets as a solution for the constructed instance of SetSelection, where the ordering of these sets is not important since the sets are disjoint. If $z_i = 1$, we choose S_{x_i}, and otherwise $S_{\bar{x}_i}$. For each of these sets the term in the sum of the goal function is 7. For each clause C_j containing the variables $x_{u(j)}$, $x_{v(j)}$ and $x_{w(j)}$ (negated or not) we choose $S_j^{z_{u(j)}z_{v(j)}z_{w(j)}}$. This set exists in the constructed instance since C_j is satisfied by the assignment z. For each of these

sets the term in the sum of the goal function is 5. Altogether the goal function takes the value $7n + 5m = 46n/3$.

It remains to show that all D-sets are covered by our selection. The sets D_i are covered by S_{x_i} or $S_{\bar{x}_i}$, resp. Similarly D_{C_j} is covered by the set $S_j^{z_{u(j)}z_{v(j)}z_{w(j)}}$. Finally, each set $D_{x_i}^k$ is covered by S_{x_i}, if $z_i = 1$, and otherwise by $S_j^{z_{u(j)}z_{v(j)}z_{w(j)}}$ where j is chosen such that the kth occurrence of x_i is in the jth clause. The analogous statement holds for $D_{\bar{x}_i}^k$. □

Proof of Lemma 5: Let a solution L for the constructed instance of SetSelection be given. First we observe that we may exchange neighbored S-sets in the solution without changing the value of the goal function if these S-sets are disjoint.

Since D_i is only contained in S_{x_i} and $S_{\bar{x}_i}$, for each i at least one of these two S-sets is contained in L. If only S_{x_i} occurs or if S_{x_i} occurs before $S_{\bar{x}_i}$, we choose $z_i = 1$, and otherwise $z_i = 0$. Similarly for each $j \in \{1, \ldots, m\}$ at least one of the sets $S_j^\bullet$ has to be contained in L in order to cover D_{C_j}. We reorder the sequence L such that for each $i \in \{1, \ldots, n\}$ either S_{x_i} or $S_{\bar{x}_i}$ is among the first n elements of the sequence.

W.l.o.g. let $z_i = 1$, i.e., S_{x_i} occurs before $S_{\bar{x}_i}$ or only S_{x_i} occurs. We successively exchange S_{x_i} with its direct predecessor in L without increasing the goal function. If S_{x_i} and its predecessor are disjoint, the goal function does not increase by the exchange as remarked above. Since $S_{\bar{x}_i}$ does not occur before S_{x_i}, these sets are not exchanged. Then the only possibility of non-disjoint sets is $S_{x_i} \cap S_j^\bullet = \{D_{x_i}^j\}$. Let T be the union of the S-sets in L before $S_j^\bullet$. Before exchanging $S_j^\bullet$ and S_{x_i} the terms in the sum of the goal function for $S_j^\bullet$ and S_{x_i} are

$$\left| \bigcup_{D \in S_j^\bullet \setminus T} D \right| \quad \text{and} \quad \left| \bigcup_{D \in S_{x_i} \setminus (T \cup S_j^\bullet)} D \right|, \text{ resp.}$$

After exchanging S_{x_i} with its predecessor $S_j^\bullet$ the terms are

$$\left| \bigcup_{D \in S_j^\bullet \setminus (T \cup S_{x_i})} D \right| \quad \text{and} \quad \left| \bigcup_{D \in S_{x_i} \setminus T} D \right|, \text{ resp.}$$

It suffices to consider the effect of $D_{x_i}^j$ to these two terms of the sum. If $D_{x_i}^j \in T$, the terms do not change. Otherwise $D_{x_i}^j = \{b_i, c_j\}$ is removed from the union for $S_j^\bullet$. Since b_i does not occur in any other D-set in $S_j^\bullet$, the term for $S_j^\bullet$ decreases by one. On the other hand, the term for S_{x_i} increases by only one (for c_j), since b_i is already contained in the union for S_{x_i}. Altogether, the value of the goal function does not increase.

Finally, we use an accounting argument to prove the lower bound on the value of the goal function. There are accounts for each variable x_i, each clause C_j and each set $D_{x_i}^\bullet$ and $D_{\bar{x}_i}^\bullet$. The contributions to the goal function are distributed to these accounts and the accounts are summed up separately.

W.l.o.g. $z_i = 1$. By the reordering of L we know that S_{x_i} covers all sets $D_{x_i}^\bullet$. Hence, S_{x_i} contributes 7 to the goal function. This contribution is distributed in the following way. Each

covered set $D^{\bullet}_{x_i}$ is charged 1 and x_i is charged 2. If L also contains $S_{\bar{x}_i}$ and q sets $D^{\bullet}_{\bar{x}_i}$ are covered by $S_{\bar{x}_i}$, the contribution of $S_{\bar{x}_i}$ is $q+1$. Then each of the covered sets is charged $1+1/q$, which is bounded below by $1+1/5$ because of $q \leq 5$.

We consider the set S^{ν}_j. If this is the first one among the sets $S^{\bullet}_j$ in L, it covers D_{C_j} and $q \leq 3$ sets $D^j_{\bullet}$ and contributes $q+2$ to the goal function. Then C_j is charged 2 and each of the covered sets $D^j_{\bullet}$ is charged 1. If S^{ν}_j is not the first set among $S^{\bullet}_j$, it covers q sets $D^j_{\bullet}$ and contributes $q+1$. One of the covered sets is charged 2 and the other ones 1.

The crucial observation is that for each clause C_j not satisfied by $(z_1, \ldots, z_n)$ there is at least one set $D^j_{\bullet}$ not covered by the first set $S^{\bullet}_j$. Hence, at least one of these sets is charged at least $1+1/5$. By the assumption that (X, C) is not satisfiable and by the promise, there are at least εm non-satisfied clauses. Hence, there are at least εm sets $D^j_{\bullet}$ that are charged at least $1+1/5$. The remaining sets $D^j_{\bullet}$ are charged at least 1. Altogether, there are $10n$ sets $D^j_{\bullet}$. Each variable and each clause are charged 2. The sum of all accounts and, therefore, the value of the goal function is bounded below by

$$\varepsilon m \cdot \left(1 + \frac{1}{5}\right) + (10n - \varepsilon m) \cdot 1 + (n+m) \cdot 2 = \frac{46n}{3} + \frac{\varepsilon n}{3}.$$

□

4. A Weak Nonapproximability Result for MinDT

In this section we provide an approximation preserving reduction from SetSelection to MinDT.

Theorem 6: *If there is a polynomial time approximation scheme for MinDT, then P = NP.*

Proof: We assume the existence of a polynomial time approximation scheme A for MinDT and construct a polynomial time approximation scheme B for SetSelection. Then from Theorem 3 the claim follows. Let $I = (P, D_1, \ldots, D_r, S_1, \ldots, S_l)$ be the instance for B and let $1+\varepsilon$ be the required performance ratio. W.l.o.g. let $\varepsilon \leq 1$.

Furthermore, w.l.o.g. we assume $r \geq 27/\varepsilon$. Otherwise r and $l \leq 2^r$ are bounded above by constants such that I can be solved in constant time using an exhaustive search. We derive two simple lower bounds on the value V_{opt} of an optimal solution for I. Since each S-set contains at most 6 D-sets, a covering consists of at least $r/6$ S-sets. Each of these sets contributes at least 2, the size of the D-sets, to the goal function; hence,

$$V_{\text{opt}} \geq r/3. \tag{1}$$

Second, if $V_{\text{opt}} < 9/\varepsilon$, Eq. (1) implies $r/3 < 9/\varepsilon$ and $r < 27/\varepsilon$ in contradiction to the above assumption. Hence,

$$V_{\text{opt}} \geq 9/\varepsilon. \tag{2}$$

From I we construct an instance T for MinDT. Let $c = l(r+1)$ and $u = 2(r+1)c$. The function f represented by T is defined as a disjunction of $r+1$ monomials over the following $|P|c + u + l$ variables:

1. For each $a \in P$ there are c variables $a^1, \ldots, a^c$, which we call a-variables.
2. There are u variables $b_1, \ldots, b_u$, which we call b-variables.
3. There are l selection variables $s_1, \ldots, s_l$.

For each set $D_i = \{a, a'\}$ there is the monomial

$$m_i = a^1 \wedge \cdots \wedge a^c \wedge a'^1 \wedge \cdots \wedge a'^c \wedge \bigwedge_{j|D_i \in S_j} s_j \wedge \bigwedge_{j|D_i \notin S_j} \bar{s}_j.$$

Furthermore, the $(r+1)$-th monomial m^* is defined by

$$m^* = \bar{s}_1 \wedge \cdots \wedge \bar{s}_l \wedge b_1 \wedge \cdots \wedge b_u.$$

We remark that each monomial contains all selection variables. Finally,

$$f = m_1 \vee \cdots \vee m_r \vee m^*.$$

In the following lemmas we collect some properties of decision trees for f.

Lemma 7: *If m is among the monomials in the definition of f and if $x \in m^{-1}(1)$, in each decision tree for f all variables contained in m are queried on the computation path for x.*

Proof: We recall that mm' is called consensus of the two monomials $x_i m$ and $\bar{x}_i m'$, if $mm' \neq 0$. The iterated consensus method computes all prime implicants of a function given as a disjunction of monomials. In particular, a disjunction of monomials consists of all prime implicants of the represented function if there is no consensus and no monomial m' can be obtained from another monomial m'' by deleting literals in m''. For more details we refer to Hachtel and Somenzi [7].

Property (S2) of the definition of SetSelection implies that for different monomials m_i and $m_{i'}$ there are two selection variables s and s' such that s occurs positively in m_i and negatively in $m_{i'}$ and the opposite holds for s'. Hence, there is no consensus of such monomials. Property (S3) implies that each monomial m_i has at least two positive literals of selection variables such that there is no consensus of m^* and any m_i either. Obviously no monomial m_i or m^* can be obtained from another of these monomials by deleting literals. As remarked above this implies that all monomials $m_1, \ldots, m_r, m^*$ are prime implicants of f and that there are no further prime implicants. Moreover, for each input $x \in f^{-1}(1)$ there is at most one prime implicant m such that $x \in m^{-1}(1)$.

Now assume contrary to the claim that for $x \in m^{-1}(1)$ there is a variable $\hat{x}$ that occurs in m but is not queried on the computation path for x. Let $\hat{m}$ be the monomial consisting of the tests performed on the computation path for x. Then $\hat{m}$ is an implicant of f, which does not contain $\hat{x}$. Let $\tilde{m}$ be a prime implicant of f that we obtain from $\hat{m}$ by deleting literals. Then m and $\tilde{m}$ are different prime implicants of f, because $\hat{x}$ is only contained in m, and $x \in m^{-1}(1) \cap \tilde{m}^{-1}(1)$ in contradiction to the last statement of the previous paragraph. □

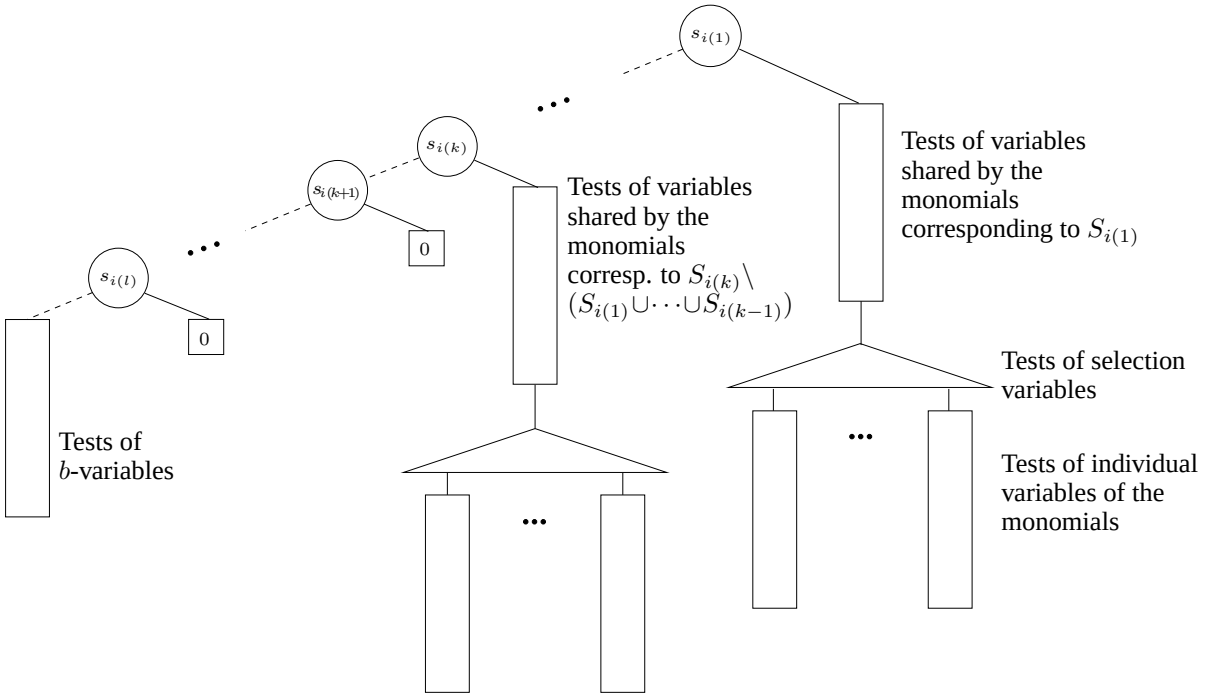

Figure 3: The decision tree constructed from a solution for SetSelection.

The next lemma shows how to obtain a decision tree for f from a solution for I and provides a relation between the size of the decision tree and the value of the solution.

Lemma 8: *Let $S_{i(1)}, \ldots, S_{i(k)}$ be a solution with the value V for I. Then a decision tree T for f with the following properties can be computed in polynomial time. T contains at most cV nodes labeled by a-variables and has size at most $cV + c + u + 1$. In particular, each such decision tree has a size of at most $2cr + c + u + 1$.*

Proof: The decision tree T is depicted in Fig. 3. At the root $s_{i(1)}$ is tested, at the 0-successor the variable $s_{i(2)}$ and so on up to $s_{i(k)}$. At the 0-successor of this node in the same way the remaining selection variables, which we call $s_{i(k+1)}, \ldots, s_{i(l)}$, are tested. In order to distinguish those tests of selection variables in the top of the decision tree from other tests of selection variables we call these nodes *special $s_{i(\cdot)}$-nodes*. At the 0-successor of the special $s_{i(l)}$-node, the b-variables are tested for 1. The rectangles in Fig. 3 denote computations of the conjunction of the corresponding variables.

The 1-successor of the special $s_{i(q)}$-node is reached for all inputs x, where $s_{i(1)}, \ldots, s_{i(q-1)}$ take the value 0 and $s_{i(q)}$ the value 1. These are in particular the inputs $x \in m^{-1}(1)$ for monomials m containing $\bar{s}_{i(1)}, \ldots, \bar{s}_{i(q-1)}$ and $s_{i(q)}$. These monomial are generated from the D-sets in $S_{i(q)} \setminus (S_{i(1)} \cup \cdots \cup S_{i(q-1)})$. For $q > k$ there are no such monomials since all D-sets are covered by $S_{i(1)}, \ldots, S_{i(k)}$. Hence, the 1-successors of the special $s_{i(k+1)}$-, $\ldots$, $s_{i(l)}$-nodes are leaves labeled by 0.

Let $q \leq k$. At the 1-successor of the special $s_{i(q)}$-node the monomials m_i, where $D_i \in S_{i(q)} \setminus (S_{i(1)} \cup \cdots \cup S_{i(q-1)})$, are evaluated. First, the a-variables contained in all considered monomials

are tested for 1. Here we exploit that the a-variables only occur positively. Afterwards, we test the selection variables. Because of property (S2) for different monomials different nodes are reached. Finally, we test the a-variables only contained in a single monomial for 1. Because of property (S4) there are no a-variables contained in more than one but not all monomials. Hence, the number of nodes labeled by a-variables equals

$$c \cdot \left| \bigcup_{D \in S_{i(q)} \setminus (S_{i(1)} \cup \cdots \cup S_{i(q-1)})} D \right|.$$

Summing up for $q = 1, \ldots, k$, we obtain the total number of nodes labeled by a-variables to be equal to cV.

The number of nodes labeled by b-variables is obviously u. For each monomial m there is exactly one computation path in the constructed tree that is chosen for all inputs $x \in m^{-1}(1)$. Hence, we may crudely estimate the number of nodes labeled by selection variables by the number of selection variables times the number of monomials. Then we obtain the upper bound $cV + u + (r+1)l = cV + c + u$ on the total number of internal nodes and the upper bound $cV + c + u + 1$ on the number of leaves.

For the last statement of the lemma we observe that the value of each solution of SetSelection is bounded above by $2r$ because each of the r D-sets may only once contribute a term of 2 to the goal function. Plugging this estimate in the bound on the number of leaves leads to the claim. □

Finally, we show how to obtain a solution for I from a decision tree for f.

Lemma 9: *If there is a decision tree T for f with at most N leaves, there is a solution for I with a value of at most $(N - u - 1)/c$, which can be computed from T in polynomial time.*

Proof: We rearrange the given decision tree T in such a way that on the path starting at the root and following the 0-edges the selection variables $s_{i(1)}, \ldots, s_{i(k)}$ are tested such that the sequence $S_{i(1)}, \ldots, S_{i(k)}$ is a solution for I. We shall prove the upper bound $(N - u - 1)/c$ on the value of this solution.

We say that T has the Property $P(q)$ if the first q nodes on the path starting at the root and following the 0-edges are labeled by selection variables $s_{i(1)}, \ldots, s_{i(q)}$. In particular, each tree has the empty property $P(0)$. We show how to construct a decision tree T_{new} for f with the property $P(q+1)$ from a decision tree T_{old} for f with the property $P(q)$ if $S_{i(1)}, \ldots, S_{i(q)}$ do not cover all D-sets. Furthermore, T_{new} is not larger than T_{old}. W.l.o.g. T_{old} is reduced.

Let v_1 be the $(q+1)$-th node on the path from the root following the 0-edges, i.e., the first node not labeled by a selection variable. We successively name nodes and subtrees of T_{old}, see also the left part of Fig. 4. Let $i = 1$. While v_i is labeled by a b-variable, w.l.o.g. b_i, give the 1-successor of v_i the name v_{i+1} and the subtree of the 0-successor of v_i the name T_i. Let v_{t+1} be the last node named in this way. Then v_{t+1} is labeled by some variable y, which is either an a-variable or a selection variable; in the former case also $t = 0$ is possible. Let T' and T'' denote the 0- and 1-successor of v_{t+1}, resp.

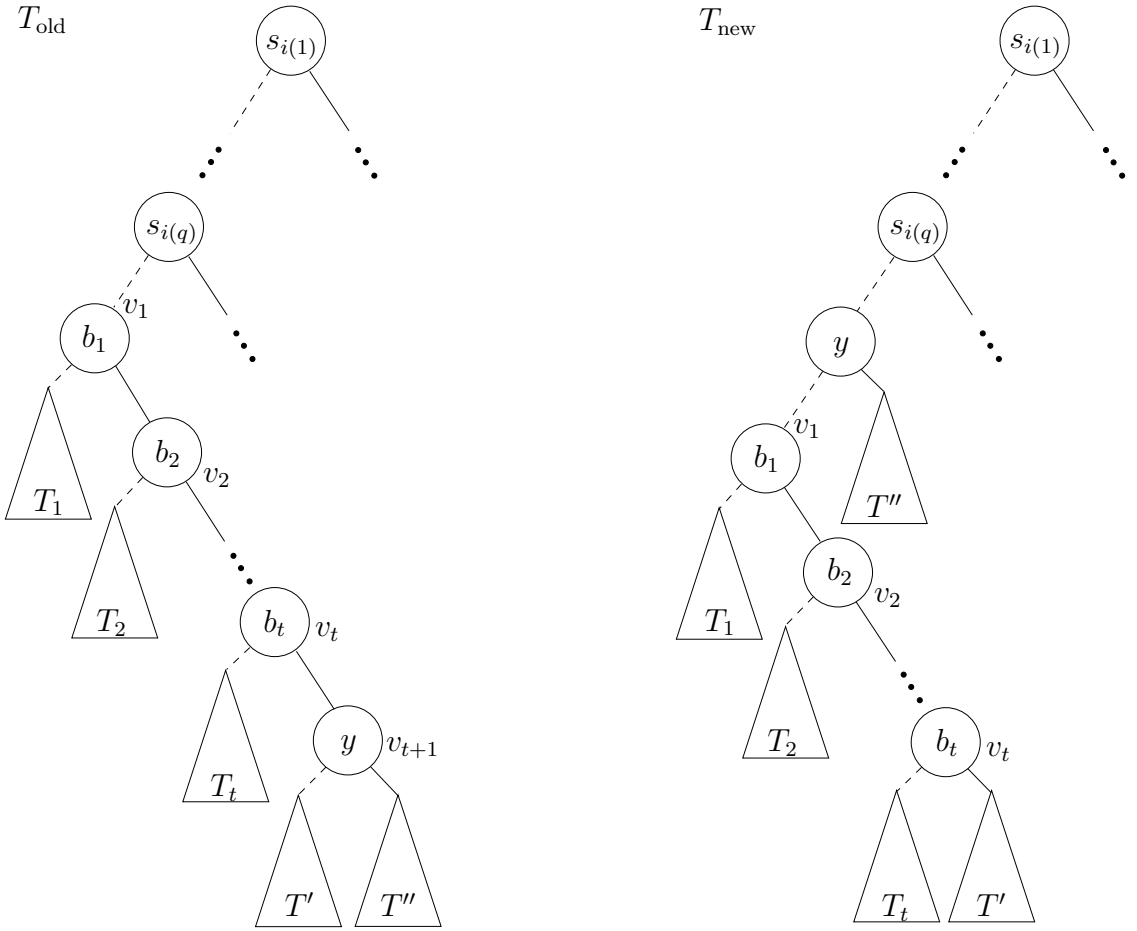

Figure 4: Construction of a decision tree T_{new} with the property $P(q+1)$ from a decision tree T_{old} with the property $P(q)$.

Case 1: The node v_{t+1} is labeled by an a-variable.
Since $S_{i(1)}, \ldots, S_{i(q)}$ are not a solution for I, there is a set $D_j \notin S_{i(1)} \cup \cdots \cup S_{i(q)}$. Since m_j does not contain any b-variable, there are inputs $x_1, \ldots, x_t \in m_j^{-1}(1)$ for which $T_1, \ldots, T_t$ are reached, resp. Lemma 7 in particular implies that $T_1, \ldots, T_t$ are not empty. Furthermore, there are inputs $x', x'' \in (m^*)^{-1}(1)$ such that T' and T'' are reached for x' and x'', resp. Lemma 7 implies that in T' and T'' as well all b-variables except $b_1, \ldots, b_t$ are queried. Hence, T_{old} has at least $2u = 2cr + u + 2c$ nodes and can thus be replaced by the tree constructed in Lemma 8 for an arbitrary solution for I, which even has the Property $P(l)$.

Case 2: The node v_{t+1} is labeled by a selection variable $s_{i(q+1)} := y$.
We replace T_{old} by the tree T_{new} shown in the right part of Fig. 4. Then T_{new} is obviously not larger than T_{old} and has the property $P(q+1)$. It remains to show that T_{new} computes the function f. We prove that for all inputs x the following statement holds.

$$\text{For } x \text{ a 1-leaf of } T_{\text{old}} \text{ is reached.} \Leftrightarrow \text{For } x \text{ a 1-leaf of } T_{\text{new}} \text{ is reached.}$$

For inputs x where at least one of the variables $s_{i(1)}, \ldots, s_{i(q)}$ takes the value 1 the claim is obvious. Hence, it suffices to consider inputs x for which in T_{old} the node v_1 is reached.

If-part: If in T_{old} the subtree T' is reached, we have $y = 0$ and $b_1 = \cdots = b_t = 1$. Hence, also in T_{new} the subtree T' is reached. Similar arguments show the same for the subtree T''.

Finally, assume that in T_{old} the subtree T_i is reached. Hence, $b_1 = \cdots = b_{i-1} = 1$ and $b_i = 0$. If $y = 0$, also in T_{new} the subtree T_i is reached. Hence, let $y = 1$. Since y is a selection variable, the considered input $x \in f^{-1}(1)$ is covered by one of the monomials $m_1, \ldots, m_r$, which do not contain any b-variable. Hence, T_{old} still computes 1 if we replace in x the values of $b_i, \ldots, b_t$ by 1. Then in T_{old} the subtree T'' is reached. Since on each computation path each variable is tested at most once, in T'' the variables $b_i, \ldots, b_t$ are not tested. Hence, also T_{new} computes the value 1 independent from the values of $b_i, \ldots, b_t$.

Only-if-part: By the same arguments as above, in T_{old} a 1-leaf is reached if in T_{new} a 1-leaf in T' or in T_i is reached. It remains the case that in T_{new} a 1-leaf in the subtree T'' is reached. Then $y = 1$. Since in T_{old} each variable is tested at most once on each computation path, the variables $b_1, \ldots, b_t$ are not tested in T''. Let x' be the input obtained from x by replacing $b_1, \ldots, b_t$ by 1. Then for x' the same computation path as for x is chosen, i.e., T'' computes a 1 also for x'. By the definition of f the subfunction $f_{|y=1}$ does not essentially depend on $b_1, \ldots, b_t$, which implies $f(x) = f(x')$. Finally, T_{old} computes for x' the same value as T_{new}. Hence, $f(x) = 1$ and T_{old} computes a 1 for x.

Eventually, we reduce T_{new}. By iteration of the whole procedure we can construct a decision tree T for f with the property $P(k)$ such that $S_{i(1)}, \ldots, S_{i(k)}$ cover all D-sets. From this solution for I we compute a lower bound on the number of nodes in T that are labeled by a-variables.

Consider the subtree at the 1-successor of the special $s_{i(q)}$-node of T. This subtree is reached for all inputs $x \in m_j^{-1}(1)$ if m_j contains $\bar{s}_{i(1)}, \ldots, \bar{s}_{i(q-1)}$ and $s_{i(q)}$. By Lemma 7 the number of a-variables in these monomials is a lower bound on the number of nodes labeled by a-variables

in the considered subtree. Hence, there are at least

$$c\left|\bigcup_{D\in S_{i(q)}\setminus(S_{i(1)}\cup\cdots\cup S_{i(q-1)})} D\right|$$

nodes labeled by a-variables in this subtree. Summing up over the different subtrees we obtain a lower bound on the number of nodes labeled by a-variables. Furthermore, there are at least u nodes labeled by b-variables. On the other hand, the sum of these lower bounds is bounded above by the number of internal nodes of the given decision tree, i.e., $N-1$. Hence,

$$u + c\sum_{q=1}^{k}\left|\bigcup_{D\in S_{i(q)}\setminus(S_{i(1)}\cup\cdots\cup S_{i(q-1)})} D\right| \leq N-1,$$

which implies that the constructed solution for I has a value of at most $(N-u-1)/c$. □

We complete the proof of Theorem 6 by providing the polynomial time approximation scheme B. We have already shown how to obtain a decision tree T from I. We apply the polynomial time approximation scheme A to T for the performance ratio $1+\varepsilon/36$. Let T^* denote the result. By the choice of the performance ratio, $|T^*| \leq (1+\varepsilon/36)|T_{\min}|$, where $T_{\min}$ is a minimal decision tree for f.

From an optimal solution with the value V_{opt} for I we may construct a decision tree of size at most $cV_{\text{opt}}+c+u+1$ by Lemma 8. Hence, $|T_{\min}| \leq cV_{\text{opt}}+c+u+1$ and

$$|T^*| \leq \left(1+\frac{\varepsilon}{36}\right)(cV_{\text{opt}}+c+u+1).$$

By Lemma 9 we can construct from T^* a solution for I with a value of at most

$$\begin{aligned} V^* &\leq \frac{|T^*|-u-1}{c} \leq \frac{(1+\varepsilon/36)(cV_{\text{opt}}+c+u+1)-u-1}{c} \\ &= V_{\text{opt}} + 1 + \frac{u}{c} + \frac{1}{c} + \underbrace{\frac{\varepsilon V_{\text{opt}}}{36}}_{\leq \varepsilon V_{\text{opt}}/3} + \underbrace{\frac{\varepsilon}{36}}_{\leq 1} + \underbrace{\frac{u\varepsilon}{36c}}_{\leq \varepsilon V_{\text{opt}}/3} + \underbrace{\frac{\varepsilon}{36c}}_{\leq 1} - \frac{u}{c} - \frac{1}{c} \\ &\leq V_{\text{opt}} + \frac{2\varepsilon}{3}V_{\text{opt}} + 3 \leq V_{\text{opt}}(1+\varepsilon). \end{aligned}$$

For the last inequality we exploit $V_{\text{opt}}\varepsilon/3 \geq 3$ which follows from Eq. (2). We discuss the other inequalities: $\varepsilon V_{\text{opt}}/36 \leq \varepsilon V_{\text{opt}}/3$ is obvious. From $\varepsilon \leq 1$ the inequalities $\varepsilon/36 \leq 1$ and $\varepsilon/(36c) \leq 1$ easily follow. Finally, $u\varepsilon/(36c) \leq \varepsilon V_{\text{opt}}/3$ follows from the definition of u, from $r \geq 1$ and from Eq. (1). Altogether, we obtain a polynomial time approximation scheme for SetSelection. Together with Theorem 3 the claim of Theorem 6 follows. □

5. The Self-Improving Property of Approximation Algorithms for MinDT

We prove the following result.

Theorem 10: *If there is a polynomial time approximation algorithm with a constant performance ratio for MinDT, there is also a polynomial time approximation scheme for MinDT.*

It is easy to see that Theorems 6 and 10 imply Theorem 1.

Proof of Theorem 10: The proof of Theorem 10 is based on the following lemma on the construction of minimal decision trees for parities of functions. Let $\mathrm{DT}(f)$ denote the size of a minimal decision tree for f.

Lemma 11: *Let $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ be Boolean functions that are defined on disjoint sets of variables. Then $\mathrm{DT}(f \oplus g) = \mathrm{DT}(f) \cdot \mathrm{DT}(g)$. Furthermore, from a decision tree T for $f \oplus g$ decision trees T_f and T_g can be constructed in polynomial time such that $|T_f||T_g| \leq |T|$.*

Proof of Lemma 11: The proof of $\mathrm{DT}(f \oplus g) \leq \mathrm{DT}(f) \cdot \mathrm{DT}(g)$ is simple. Start with minimal decision trees T_f and T_g for f and g. First observe that we obtain a decision tree for $\bar{g}$ by replacing the labels c of the leaves of T_g by $\bar{c}$. Now replace in T_f each leaf labeled by c by a decision tree for $c \oplus g$. Let T^* denote the resulting tree. Obviously it computes $f \oplus g$ and $|T^*| = |T_f||T_g|$.

For the proof of $\mathrm{DT}(f \oplus g) \geq \mathrm{DT}(f) \cdot \mathrm{DT}(g)$ we first note that the claim is obvious if f or g is a constant function. Hence, let f and g be nonconstant. We start with a decision tree T for $f \oplus g$ and modify this tree without increasing the size or changing the represented function. Eventually, we obtain a decision tree T^* consisting of a decision tree for f where the leaves are replaced by copies of a decision tree for g, or the similar decision tree, where the roles of f and g are exchanged. It is easy to obtain from T^* decision trees T_f and T_g for f and g such that $|T_f||T_g| \leq |T^*| \leq |T|$. If T is minimal, the claim follows. Moreover, we obtain a polynomial algorithm for constructing T_f and T_g.

W.l.o.g. T is reduced. Let f be defined over the set $X = \{x_1, \ldots, x_n\}$ of variables and g over $Y = \{y_1, \ldots, y_m\}$. We partition the set of internal nodes of T into *regions*, where two internal nodes v and v' are contained in the same x-region, if both v and v' are labeled by an x-variable and all nodes on the unique path between v and v' are labeled by x-variables, where we neglect the direction of the edges. Similarly we define y-regions. A region has the rank 0 iff the outgoing edges only lead to leaves. A region has the rank r iff the maximum of the ranks of the regions reached via the outgoing edges is $r - 1$.

Let R_1 be a region of rank 1. Since f and g are nonconstant, such a region exists in T. If the root of R_1 is also the root of T, then T is already the desired decision tree T^* and the claim follows. Otherwise, the root of R_1 has a predecessor, which is contained in some region R_2. W.l.o.g. let R_2 be an x-region and R_1 be a y-region. Then the nodes reached by the edges leaving R_1 are contained in x-regions or are leaves. We call these x-regions or leaves $S_1, \ldots, S_k$. The situation is also depicted in Fig. 5. By the definition the rank of the x-regions among $S_1, \ldots, S_k$ is 0, i.e.,

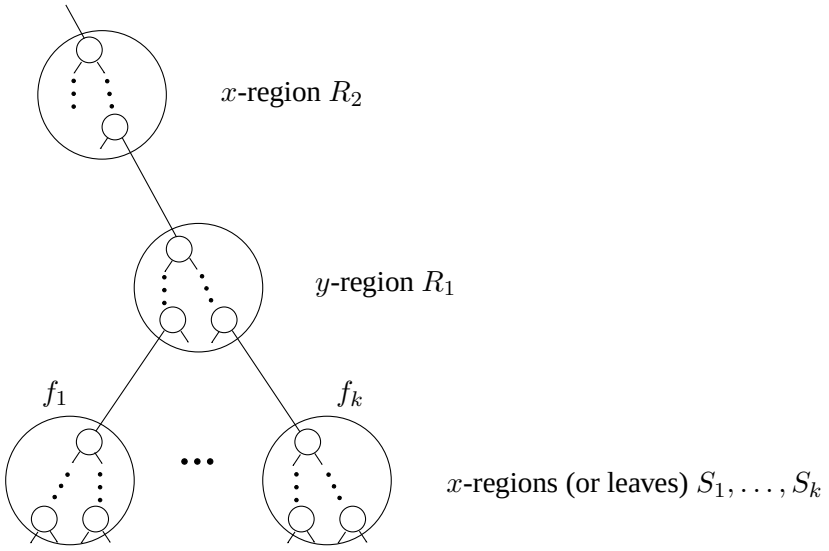

Figure 5: The arrangement of the regions.

the outgoing edges only lead to leaves. Let $f_1, \ldots, f_k$ be the functions computed at the roots of $S_1, \ldots, S_k$. We claim that for all $i \in \{2, \ldots, k\} : (f_1 = f_i \vee f_1 = \bar{f_i})$, i.e., these functions coincide up to polarity.

Consider the evaluation of f_1. This function only depends on some set X' of x-variables that are tested in S_1. Hence, on the computation path leading to S_1 some set of y-variables has to be tested in such a way that by the corresponding setting of these variables the function g becomes a constant. Let $X'' = X \setminus X'$. From the x-variables only those in X'' are tested in T before reaching S_1 since T is reduced. Then for the evaluation of all functions $f_1, \ldots, f_k$ the variables in X'' get the same values. On the other hand, the y-variables tested before reaching $S_1, \ldots, S_k$ have different values such that also g may take different values. Hence, $f_1, \ldots, f_k$ coincide merely up to polarity. In particular, they are not constant, i.e., there is no subtree among $S_1, \ldots, S_k$ that merely consists of a leaf.

Assume w.l.o.g. that S_1 is not larger than $S_2, \ldots, S_k$ and that S_1 has l leaves. We "exchange" R_1 and S_1. Let $q_1, \ldots, q_l$ denote the labels of the leaves of S_1. Let $p_i := f_1 \oplus f_i$, i.e., 0, if these functions are equal, and otherwise 1. In a new copy of S_1 we replace the leaves by disjoint copies of R_1, where the ith leaf in the jth copy gets the label $p_i \oplus q_j$. It is easy to see that the resulting decision tree has kl leaves and computes the same function as the subtree consisting of R_1 and $S_1, \ldots, S_k$. Hence, we may replace the subtree of R_1 by this new decision tree without increasing the size.

After the exchange the number of regions has become smaller since R_2 and the copy of S_1 are merged into one x-region. Hence, we may iterate this procedure until there is no longer a region of rank 2. Then we obtain the desired decision tree T^*. □

We continue the proof of Theorem 10. Let a polynomial time approximation algorithm A for MinDT with the performance ratio c be given. We show how to construct a polynomial time approximation algorithm B with the performance ratio $\sqrt{c}$. Hence, for each $\varepsilon > 0$ a constant

number of iterations of this procedure suffice to obtain a polynomial time approximation algorithm with a performance ratio of at most $1+\varepsilon$. Hence, we get a polynomial time approximation scheme for MinDT.

Let T be the input for B and let f denote the function represented by T. Let g be a copy of f on a disjoint set of variables and let T' be a decision tree for $f \oplus g$, which can easily be constructed from T. We apply A to T' and obtain a decision tree T''. By the second statement of Lemma 11 from T'' a decision tree for f with size at most $\sqrt{|T''|}$ can be constructed in polynomial time, which is the output.

By Lemma 11 and the bound c on the performance ratio of A we have $OPT(f)^2 = OPT(f \oplus g) \geq A(f \oplus g)/c = |T''|/c$. This implies $\sqrt{|T''|} \leq \sqrt{c} \cdot OPT(f)$. Hence, B is an approximation algorithm with the performance ratio $\sqrt{c}$. □

References

[1] Buhrman, H. and de Wolf, R. (1999). Complexity measures and decision tree complexity: a survey. To appear in *Theoretical Computer Science*.

[2] Ehrenfeucht, A. and Haussler, D. (1989). Learning decision trees from random examples. *Information and Computation* 82, 231–246.

[3] Feige, U. (1998). A threshold of $\ln n$ for approximating set cover. *Journal of the Association for Computing Machinery* 45, 634–652.

[4] Friedman, S.J. and Supowit, K.J. (1990). Finding the optimal variable ordering for binary decision diagrams. *IEEE Transactions on Computers* 39, 710–713.

[5] Garey, M.R. and Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman.

[6] Guijarro, D., Lavín, V. and Raghavan, V. (1999). Exact learning when irrelevant variables are abound. *Information Processing Letters* 70, 233–239.

[7] Hachtel, G.D. and Somenzi, F. (1996). *Logic Synthesis and Verification Algorithms*. Kluwer.

[8] Hancock, T., Jiang, T., Li, M. and Tromp, J. (1996). Lower bounds on learning decision lists and trees. *Information and Computation* 126, 114–122.

[9] Hyafil, L. and Rivest, R.L. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters* 5, 15–17.

[10] Jukna, S., Razborov, A., Savický, P. and Wegener, I. (1999). On P versus NP ∩ co-NP for decision trees and read-once branching programs. *Computational Complexity* 8, 357–370.

[11] Knuth, D.E. (1973). *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison Wesley.

[12] Murthy, S.K. (1998). Automatic construction of decision trees from data: a multi-disciplinary survey. *Data Mining and Knowledge Discovery* 2, 345–389.

[13] Preparata, F.P. and Shamos, M.I. (1985). *Computational Geometry.* Springer.

[14] Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning* 1, 81–106.

[15] Sieling, D. (2002). The nonapproximability of OBDD minimization. *Information and Computation* 172, 103–138.

[16] Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams, Theory and Applications.* Society for Industrial and Applied Mathematics.

[17] Zantema, H. and Bodlaender, H.L. (2000). Finding small equivalent decision trees is hard. *International Journal of Foundations of Computer Science* 11, 343–354.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'