Abstract

We give a $\ln(n) + 1$-approximation for the decision tree (DT) problem. We also show that DT does not have a PTAS unless P=NP. An instance of DT is a set of m binary tests $T = (T_1, \ldots, T_m)$ and a set of n items $X = (X_1, \ldots, X_n)$. The goal is to output a binary tree where each internal node is a test, each leaf is an item and the average number of tests used to uniquely identify each item (or equivalently, the total external path length) is minimized. In addition, we show DT does not have a PTAS unless P=NP. DT has a rich history in computer science with applications ranging from medical diagnosis to experiment design. Our work, while providing the first non-trivial upper and lower bounds on approximating DT, also demonstrates that DT and a subtly different problem which also bears the name decision tree have fundamentally different approximation complexity. In addition, we show a connection between ConDT and a third type of decision tree problem called MinDT, which allows us to show that no $2^{\log^{\delta}(n)}$-approximation exists for MinDT, for $\delta < 1$, unless NP is quasi-polynomial.

1 Introduction

We consider the problem of approximating optimal binary decision trees. Garey and Johnson [5] define the decision tree (DT) problem as follows: given a set of m binary tests $T = (T_1, \ldots, T_m)$ and a set of n items $X = (X_1, \ldots, X_n)$, output a binary tree where each leaf is labeled with one item from X and each internal node is labeled with one test from T. If an item passes a test, it follows the right branch, if it fails a test it follows the left branch. A path from the root to a leaf uniquely identifies the item labeled by that leaf. The goal is to find a tree which minimizes the average number of tests used to identify the set of items (or equivalently, the total external path length of the tree). Given enough tests to distinguish each object in DT, an optimal solution always has n leaves. We note (without proof) that deciding whether there are a sufficient number of tests is easily computable in polynomial time. An equivalent formulation of the problem views the items as m-bit binary strings, where bit i is 1 if the item passes test T_i and 0 otherwise. We use instances of this type when discussing DT throughout this paper and denote them using the set of items X. Decision trees have broad application [8, 9]. In fact, Hyafil and Rivest proved DT NP-Complete precisely because "of the large amount of effort that [had] been put into finding efficient algorithms for constructing optimal binary decision trees" [7].

In this paper, we give a $(ln(n) + 1)$-approximation for the decision tree problem. We also show that DT does not afford a PTAS unless P=NP. To the best of our knowledge, these are the first non-trivial upper and lower bounds on the approximation ratio for DT. The upper bound also serves a second, more fundamental purpose. It disambiguates the approximation complexity of DT from a similar problem which also bears the name *decision tree*. We call this problem ConDT, for consistent decision tree, and define it here: The input is a set of n binary strings, each of length m, called examples (many papers take m to be the number of examples and take n to be the number of bits). The output is a binary tree where each interior node tests some bit i of the examples, and maps the example to its left child if i is a 0 and its right child if i is a 1. Each leaf is labeled either TRUE or FALSE. A consistent decision tree maps each positive example to a leaf labeled TRUE and each negative example to a leaf labeled FALSE. The size of a tree is the number of leaves. ConDT seeks the minimum size tree which is consistent with the examples.

ConDT is hard to approximate. Alekhnovich et. al. [1] show it is not possible to approximate size s decision trees by size s^k decision trees for any constant $k \geq 0$ unless NP is contained in DTIME[$2^{m^{\epsilon}}$] for some $\epsilon < 1$. This improves a result from Hancock et. al. [6] which shows that no $2^{\log^{\delta} s}$-approximation exists for size s decision trees for any $\delta < 1$ unless NP is quasi-polynomial. These results hold for $s = n^{1/k}$ for some constant k.

It is clear the small differences in problem description between DT and ConDT lead to large differences in complexity: ConDT has no $c\ln(n)$-approximation for any constant c (modulo unlikely complexity results) whereas our results yield such an approximation for DT for any $c > 1$. The differences are subtle enough that [2] and its online incarnation [3], define the decision tree problem according to the DT input and output, the ConDT minimum tree size measure and cite the Hancock et. al. paper as a negative result. Hence, we consider the separation of DT and ConDT in terms of approximation complexity, one of the primary contributions of our work.

We are not the first, however, to note that DT and ConDT are different problems. Twenty years ago, Moret aimed to unify disparate decision tree definitions with a common framework. [8]. In this framework, one may view DT and ConDT as unique instances of a very general decision tree problem where each item is tagged with k possible labels. With DT there are always $k = n$ labels, but only one item per label. With ConDT, there are only two labels, but multiple items carry the same label. But which differences in problem description lead to the differences in approximation complexity? The input to DT and ConDT are identical except that the strings in ConDT are augmented with TRUE / FALSE labels. The output is different (ConDT trees don't necessarily have n leaves) and so is the

measure (DT uses total external path length and ConDT uses tree size). The difference in measure, though, does not distinguish the approximation complexity of the problems: The lower bounds on learning ConDT trees also hold for total external path length and tree size is not an insightful measure for DT since all feasible solutions have n leaves. Hence, the difference is due entirely to the type of tree.

In the following section we describe and analyze our approximation algorithm for DT. We also consider the problem with weights associated with the tests and show that the $(ln(n)+1)$-approximation remains intact. In Section 3, we show that there exists a $\delta > 0$ such that DT does not have a $(1+\delta)$-factor approximation unless P=NP. Finally, we conclude with an improved lower bound for the MinDT problem which we describe in Section 4. These results match those given by [6] for ConDt.

2 Approximating DT

Given a set of binary m-bit strings S, choosing some bit i always partitions the items into two sets S^0 and S^1 where S^0 contains those items with bit $i = 0$ and S^1 contains those items with $i = 1$. A greedy strategy for splitting a set S chooses the bit i which minimizes the difference between the size of S^0 and S^1. In other words, it chooses the bit which most evenly partitions the set. Using this strategy, consider the following greedy algorithm for constructing decision trees of the DT type given a set of n items X:

GREEDY-DT(X)

```
1  if X = ∅
2     then return NIL
3     else Let i be the bit which most evenly partitions X into X^0 and X^1
4          Let T be a tree node with left child left[T] and right child right[T]
5          left[T] ← GREEDY-DT(X^0)
6          right[T] ← GREEDY-DT(X^1)
7          return T
```

This algorithm is not optimal, but it does approximate the optimal solution within a factor of $\ln(n)+1$. We prove this result below.

Theorem 1. *If X is an instance of DT with n items and optimal cost $\mathcal{C}^*$ then* GREEDY-DT(X) *yields a tree with cost at most $(ln(n)+1)\mathcal{C}^*$*

Proof. We begin with some notation. Let $\mathcal{T}$ be the tree constructed by GREEDY-DT on X. Let $\mathcal{C}$ be the cost of the greedy tree. For each *unordered* pair of items $\{x_i, x_j\}$ let S_{ij} be the node of $\mathcal{T}$ which splits x_i from x_j. For convenience we can also think of S_{ij} as a set of items where x is in S_{ij} if and only if x is a leaf in the subtree of S_{ij}. Let $S^0_{ij} \subset S_{ij}$ be the set of items following the 0-branch of S_{ij} and let $S^1_{ij} \subset S_{ij}$ be the set of items following the 1-branch of S_{ij}. Without loss of generality, assume $|S^0_{ij}| \geq |S^1_{ij}|$. The number of sets to which an item belongs equals the length of its path from the root, so the cost of $\mathcal{T}$ may be expressed as the sum of the sizes of each S_{ij}.

$$\mathcal{C} = \sum_{S_{ij} \in \mathcal{T}} |S_{ij}|$$

Our analysis uses an accounting scheme to spread the total cost of the greedy tree among all unordered pairs of the items. Observe that for any pair of items $\{x_i, x_j\}$ in any feasible tree for X, there exists a single interior node S_{ij} such that x_i follows one branch and x_j follows the other branch. We say S_{ij} *splits* x_i and x_j. Since each set S_{ij} contributes its size to the total cost of the tree, we spread its size uniformly among the $|S^0_{ij}||S^1_{ij}|$ pairs of items split at S_{ij}. Let c_{ij} be the pair cost assigned to each unordered pair of items $\{x_i, x_j\}$ where

$$c_{ij} = \frac{1}{|S^1_{ij}|} + \frac{1}{|S^0_{ij}|}.$$

We can now talk about the cost of a tree node S_{ij}, by the costs associated with the pairs of items split at S_{ij}. Summing the costs of these pairs is, by definition, exactly the size of S_{ij}:

$$\sum_{x_i \in S^0_{ij}} \sum_{x_j \in S^1_{ij}} c_{ij} = |S^0_{ij}||S^1_{ij}|\Big(\frac{1}{|S^0_{ij}|} + \frac{1}{|S^1_{ij}|}\Big) = |S_{ij}|$$

Because two items are split from each other exactly once, $\mathcal{C}$ is exactly the sum of the all pair costs:

$$\mathcal{C} = \sum_{i,j;i<j} c_{ij}$$

Now consider the optimal tree $\mathcal{T}^*$ for X. As before, if Z_{ij} is an interior node of $\mathcal{T}^*$ then Z_{ij} is also the set of items that are leaves of Z_{ij}. Similarly, we let Z_{ij}^0 be the set of items associated with the 0-subtree of Z_{ij} and Z_{ij}^1 be the set of items associated with the 1-subtree of Z_{ij}. The cost of the optimal tree, $\mathcal{C}^*$, is

$$\mathcal{C}^* = \sum_{Z_{ij} \in \mathcal{T}^*} |Z_{ij}| \leq \sum_{i,j;i<j} c_{ij} = \sum_{Z_{ij} \in \mathcal{T}^*} \sum_{x_i \in Z_{ij}^0} \sum_{x_j \in Z_{ij}^1} c_{ij} \tag{1}$$

The last term of Equation 1 rearranges the greedy pair costs according to the structure of the optimal tree. So if Z_{ij} is a node in the optimal tree, then it defines $|Z_{ij}^0||Z_{ij}^1|$ pairs of items. Our goal is to show that the sum of the c_{ij} associated with each pair (but which are defined with respect to the greedy tree) total at most a factor of $H(|Z_{ij}|)$ more than $|Z_{ij}|$ where $H(d) = \sum_{i=1}^{d} 1/i$ is the d^{th} harmonic number. This is made precise in the following lemma:

Lemma 1. *For each Z_{ij} in the optimal tree:*

$$\sum_{x_i \in Z_{ij}^0} \sum_{x_j \in Z_{ij}^1} c_{ij} \leq |Z_{ij}| H(|Z_{ij}|)$$

where each c_{ij} is defined with respect to the greedy tree $\mathcal{T}$.

Proof. Consider any node Z_{ij} in the optimal tree. For any unordered pair of items $\{x_i, x_j\}$ split at Z_{ij}, imagine using the bit associated with the split at Z_{ij} on the set S_{ij}. Call the resulting two sets $S_{ij}^{Z^0}$ and $S_{ij}^{Z^1}$ respectively. Since the greedy split at S_{ij} minimizes c_{ij}, we know

$$c_{ij} = \frac{1}{|S_{ij}^0|} + \frac{1}{|S_{ij}^1|} \leq \frac{1}{|S_{ij}^{Z^0}|} + \frac{1}{|S_{ij}^{Z^1}|} \leq \frac{1}{|(S_{ij} \cap Z_{ij}^0)|} + \frac{1}{|(S_{ij} \cap Z_{ij}^1)|}.$$

Hence

$$\sum_{x_i \in Z_{ij}^0} \sum_{x_j \in Z_{ij}^1} c_{ij} \leq \sum_{x_i \in Z_{ij}^0} \sum_{x_j \in Z_{ij}^1} \frac{1}{|(S_{ij} \cap Z_{ij}^0|} + \frac{1}{|(S_{ij} \cap Z_{ij}^1|} \tag{2}$$

One interpretation of the sum in (2) views each item x_i in Z_{ij}^0 as contributing

$$\sum_{x_j \in Z_{ij}^1} \frac{1}{|(S_{ij} \cap Z_{ij}^1|}$$

to the sum and each node x_j in $Z_{T'}^1$ as contributing

$$\sum_{x_i \in Z_{ij}^0} \frac{1}{|(S_{ij} \cap Z_{ij}^0|}$$

to the sum. For clarity, we can view Z_{ij} as a complete bipartite graph where Z_{ij}^0 is one set of nodes and Z_{ij}^1 is the other. Letting $b_{ij} = 1/(|(S_{ij} \cap Z_{ij}^1|)$ and $b_{ji} = 1/(|S_{ij} \cap Z_{ij}^0|)$ we can think of every edge (x_i, x_j) as having two costs: one associated with x_i (b_{ij}) and the other associated with x_j (b_{ji}). Thus, the total cost of Z_{ij} is at most the sum of all the b_{ij} and b_{ji} costs. We can bound the total cost by first bounding all the costs associated with a particular node. In particular, we claim:

Claim 1. *For any $x_i \in Z_{ij}^0$ we have*

$$\sum_{x_j \in Z_{ij}^1} b_{ij} = \sum_{x_j \in Z_{ij}^1} \frac{1}{|S_{ij} \cap Z_{ij}^1|} \leq H(|Z_{ij}^1|)$$

Proof. If Z_{ij}^1 has m items then let $v = (v_1, \ldots, v_m)$ be an ordering of Z_{ij}^1 in reverse order from when the items are separated from x_i in the greedy tree (with ties broken arbitrarily). This means item v_1 is the last item separated from x_i, v_m is the first item separated from x_i, and in general v_t is the t^{th} item separated from x_i. When v_m is separated from x_i there must be at least $|Z_{ij}^1|$ items in S_{im} — by our ordering the remaining items in Z_{ij}^1 must still be present — so $Z_{ij}^1 \subset S_{im}$. Hence b_{im}, the cost assigned to x_i on the edge (x_i, v_m), is at most $1/|(Z_{ij}^1)|$ and in general the cost assigned to the pair b_{it} is at most $1/t$. This yields :

$$\sum_{x_j \in Z_{ij}^1} b_{ij} \leq H(|Z_{ij}^1|)$$

which proves the claim. □

We can use the same argument to prove the analogous claim for all the items in Z_{ij}^1. With these inequalities in hand we have

$$\begin{aligned}
\sum_{x_i \in Z_{ij}^0} \sum_{x_j \in Z_{ij}^1} \frac{1}{|(S_{ij} \cap Z_{ij}^0|} + \frac{1}{|(S_{ij} \cap Z_{T'}^1|} &\leq |Z_{ij}^0| H(|Z_{ij}^1|) + |Z_{ij}^1| H(|Z_{ij}^0|) \\
&< |Z_{ij}^0| H(|Z_{ij}|) + |Z_{ij}^1| H(|Z_{ij}|) \\
&= |Z_{ij}| H(|Z_{ij}|) \qquad (\text{since } |Z_{ij}^0| + |Z_{ij}^1| = |Z_{ij}|)
\end{aligned}$$

□

Substituting this result into the initial inequality completes the proof of the theorem.

$$\begin{aligned}
\sum_{Z_{ij} \in \mathcal{T}^*} \sum_{x_i \in Z_{ij}^0} \sum_{x_j \in Z_{ij}^1} c_{ij} &\leq \sum_{Z_{ij} \in \mathcal{T}^*} |Z_{ij}| H |Z_{ij}| \\
&\leq \sum_{Z_{ij} \in \mathcal{T}^*} |Z_{ij}| H(n) \\
&= H(n) \mathcal{C}^* \\
&\leq (\ln(n) + 1) \mathcal{C}^*
\end{aligned}$$

□

3 Inapproximability of DT

In this section we show that there exists a universal constant $\delta > 0$ such that DT has no $(1 + \delta)$ factor approximation unless P=NP. This immediately implies that if DT has a PTAS, then P=NP. We provide a gap-preserving reduction to DT from MAX-3SAT5 which Feige defines in [4]:

Input: A set of n variables $X = \{x_1, \ldots, x_n\}$ and m clauses $C = \{C_1, \ldots, C_m\}$ where each clause has exactly three literals (a literal is a variable or its negation), no variable appears more than once in a clause, and each variable appears in exactly 5 clauses. Note that $m = \frac{5n}{3}$

Output: The maximum number of clauses which can be satisfied simultaneously by some variable assignment

Feige shows that for some $\epsilon > 0$ it is NP-hard to distinguish between those 3SAT5 formulas which are satisfiable and those which have at most $(1 - \epsilon)|C|$ clauses satisfied simultaneously. Hence, we will restrict our attention to just those instances which are either satisfiable or which, for any assignment, have at least $\epsilon|C|$ clauses that are not satisfied.

The idea is to reduce 3SAT5 to a covering problem which we then reduce to a decision tree. The form of the covering problem is important: in decision trees, cost is a function of depth so to control the cost, we require some control over the depth of the leaves. We show how to reduce instances of 3SAT5 to instances of set cover where

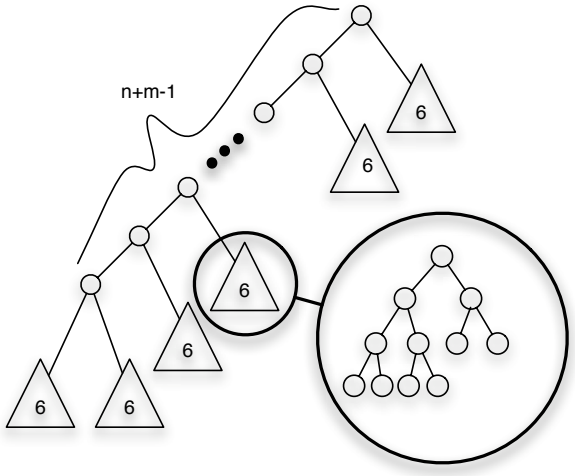

Figure 1: The topology of the optimal tree. Here the $n + m - 1$ bits along the left branch correspond to the sets in the set cover.

every set has size 6 and where every satisfiable instance of 3SAT5 has an exact cover and every unsatisfiable instance requires some constant factor number of sets more for a cover. This means we can show that a satisfiable instance of 3SAT5 maps to a decision tree with cost at most $\mathcal{C}$ but that for some universal constant $\delta > 0$ all solution trees for an unsatisfiable instance have cost at least $\mathcal{C}(1 + \delta)$. It follows that no polynomial time $(1 + \delta)$ factor approximation exists for DT unless P=NP because if an approximation did exist we could distinguish satisfiable instances of 3SAT5 from unsatisfiable instances in polynomial time.

The reduction from 3SAT5 to the bounded-size set cover problem uses a simple variation on the reduction used by Sieling [10] to show that MinDT (a problem we describe in Section 4) has no PTAS. After reviewing this reduction, we transform the resulting set cover problem into a decision tree problem.

3.1 Reduction from 3SAT5 to Set Cover

Given a 3SAT5 formula ϕ with n variables and m clauses, we define the following polynomial time reduction $f(\phi)$ to a set cover instance (U, Z) where U is a set of items and Z is a collection of subsets of U. First we define the set of items. For each variable x_i, create 11 items: y_i, a_{ij} $(1 \leq j \leq 5)$ and b_{ij} $(1 \leq j \leq 5)$. For each clause C_j create three items c_{j_1}, c_{j_2}, and c_{j_3}. In total, there are $11n + 3m = 16n$ items, so $|U| = 16n$.

Now we define the sets. For each variable x_i, create two sets:

$$S_i^a = \{y_i\} \cup \{a_{ij} \,|\, (1 \leq j \leq 5)\}$$
$$S_i^b = \{y_i\} \cup \{b_{ij} \,|\, (1 \leq j \leq 5)\}$$

For each clause C_j, create seven sets where each set corresponds to a (local) variable assignment that satisfies the clause. Construct each set in the following way: Let x_{u_1}, x_{u_2}, and x_{u_3} be the variables appearing in clause j. For each local satisfying assignment $z(x)$ (there are seven), create the set

$$S_j^z = \{c_{j_1}, c_{j_2}, c_{j_3}\} \cup \{d_1, d_2, d_3\} \quad \text{where } d_k = \begin{cases} b_{u_k j} & \text{if } z(x_{u_k}) = 1 \\ a_{u_k j} & \text{if } z_(x_{u_k}) = 0 \end{cases}$$

For example if $C_j = (x_1 \vee \bar{x_2} \vee x_3)$ then there are eight possible local assignments to the variables. The assignment $x_1 = 1, x_2 = 0, x_3 = 0$ satisfies the clause so the set

$$S_j^{100} = \{c_{j_1}, c_{j_2}, c_{j_3}\} \cup \{b_{1j}, a_{2j}, a_{3j}\}$$

is included. However, the assignment $x_1 = 0$, $x_2 = 1$, and $x_3 = 0$ fails to satisfy the clause, so the set

$$S_j^{010} = \{c_{j_1}, c_{j_2}, c_{j_3}\} \cup \{a_{1j}, b_{2j}, a_{3j}\}$$

is not included. There are seven sets per clause, so in total there $7m$ sets in Z. Note that every set has exactly 6 items. Though our reduction is not identical to the one given by Sieling, the following theorem still holds:

Theorem 2 ([10])**.** *If ϕ is a satisfiable 3SAT5 formula then there is a solution to $f(\phi)$ of size $n+m$. If, for any assignment to ϕ, at most $(1-\epsilon)m$ clauses may be satisfied simultaneously, then a solution to $f(\phi)$ has size at least $n+m+\frac{\epsilon n}{3}$.*

The idea is that if z is a satisfying assignment to ϕ and $z(x_i)=1$, then we can cover the a_{ij} with the S_i^a sets and the b_{ij} with the $S_j^:$ sets that corresponds to the assignment z. Likewise, if $z(x_i)=0$ then we can cover the b_{ij} with the S_i^b sets and the a_{ij} with the appropriate $S_j^:$ sets. If no satisfying assignment exists for ϕ then the desired $S_j^:$ sets for ϵm clauses don't exist and at best, we can cover 5 of the remaining items at a time using the S_i^a and S_i^b sets. These additional sets create the $\frac{\epsilon n}{3}$ gap. For more details, we refer the reader to [10]. Note that when ϕ is satisfiable the set cover is an *exact* cover meaning no two sets share the same item.

We now define a gap-preserving reduction g from instances of set cover (U,Z) of the form given by f to instances X of DT. Let $|U|=n'$ and $|Z|=m'$ The reduction is a natural one. For each item u in U create a binary string of length $4m'$. The first m' bits correspond to the m' sets of Z so we call them *Z-bits*. A string for item u has Z-bit i set to 1 if and only if u is in set Z_i. In other words, the first m' bits denote set membership in Z. The final $3m'$ bits allow us to disambiguate the items from one another—recall that each set has six items, so if we use a set bit at node T in the decision tree, then at most 6 items may follow the 1-branch of T. We want to control the shape of the subtree formed by these items. That is, we want them to form as near a complete tree as possible. Hence, each set Z_i in Z adds 3 more bits per string. These bits are set to 0 for all items not appearing in Z_i. For those items which do appear in Z_i, we assign the bits so that a near-complete tree of height 3 is always possible (e.g., half the items will have the first bit set to 1, the other half will have them set to 0, etc. See Figure 1 for details). Together, these n' strings comprise the DT instance X. Note that the number of items does not change and that the reduction is polynomial in n'.

Claim 2. *if ϕ is a satisfiable 3SAT5 formula then $g(f(\phi))$ has a tree with cost at most $\frac{64n^2}{3}+\frac{152n}{3}-6$.*

Proof. All we need to do is construct the cascading tree pictured in Figure 1. The bits along the left side correspond to the optimal set cover, except for the final few bits (which we can think of as being the final set of 6 items not yet covered). Selecting the $n+m-1=n+(5/3)n-1$ sets yields a cost of

$$\mathcal{C}=(-6)+\sum_{i=1}^{n+5/3n}6i+16=\frac{64n^2}{3}+\frac{152n}{3}-6.$$

□

In fact, we can also show that $\mathcal{C}$ is the minimum cost. To see this, imagine a tree $\mathcal{T}$ corresponding to some sub-optimal cover of the items. This tree has depth at least $n+m+3$. Since every bit partitions off at most 6 items, we know $\mathcal{T}$ is also a cascading-style tree, but that some of its subtrees have fewer than 6 items. It's easy to show that shifting a leaf lower in the tree to a leaf higher in the tree decreases the total cost. This is because the complete binary tree minimizes total external path length. Hence it is advantageous to maximize the number of subtrees of size 6 that occur higher in the tree. Since the tree corresponding to the optimal set cover is composed of subtrees with size 6, it has minimum cost. What's left to show is that g preserves the gap in cost when the set cover requires at least $n+m+\frac{\epsilon n}{3}$ sets.

Claim 3. *if ϕ is a 3SAT5 formula such that, for any assignment to ϕ, at most $(1-\epsilon)|C|$ clauses are simultaneously satisfied, then any solution to $g(f(\phi))$ has cost at least $\mathcal{C}+\frac{n^2\epsilon^2}{18}+\frac{n\epsilon}{6}-1$.*

Proof. From Theorem 2 we know that at least $\frac{n\epsilon}{3}$ additional sets are required to cover all the items. Given that the tree must have depth at least $n+m+n\epsilon/3-1$ for large enough n and that any bit can partition off at most 6 items, what is the minimum cost tree which adheres to these constraints? In the best case, each of the additional $\frac{n\epsilon}{3}$ internal nodes partitions off only a single element with the remaining $n'-\frac{n\epsilon}{3}+1$ items partitioned off by the first $n+m$ interior nodes. In the best case, these will be subtrees of size 6 where possible. This is illustrated in Figure 2. Another way to think about this tree is starting with the optimal tree from Claim 2 and repeatedly creating a new leaf at a lower level by deleting a leaf at a higher level. This is tantamount to shifting a leaf. This process is a convenient way to analyze

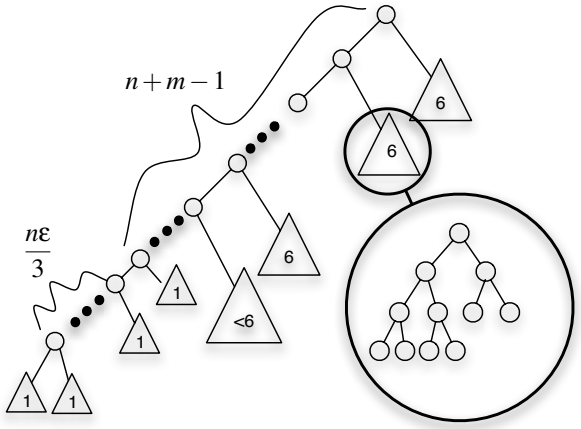

Figure 2: This tree represents the *best case* when $n + m + \frac{n\epsilon}{3}$ sets are required to cover the items. To minimize the cost in this scenario, the extra bits cover a single item and the top of the tree is filled with subtrees of size 6.

the cost since the first leaf shifted adds at least 1 to the cost, the second leaf shifted adds at least 2 to the cost, and so forth. Hence, the cost of the optimal tree for the gap case is at least

$$\mathcal{C} + 1 + 2 + \ldots + \frac{n\epsilon}{3} - 1 = \mathcal{C} + \frac{n^2\epsilon^2}{18} + \frac{n\epsilon}{6} - 1.$$

□

Combining Claims 2 and 3 yields the desired result:

Theorem 3. *There exists a $\delta > 0$ such that DT cannot be approximated in polynomial time within a factor of $(1 + \delta)$ unless $P = NP$.*

Proof. Suppose that for all $\delta > 0$, DT could be approximated in polynomial time within a factor of $(1 + \delta)$. Let $0 < \epsilon < 1$ be given so that the ϵ-gap in satisfiable 3SAT5 formulas and unsatisfiable 3SAT5 formulas exists. Choose $\delta < \frac{3\epsilon^2}{1152}$. Now any satisfiable instance has cost at most $(1 + \delta)\mathcal{C} < \mathcal{C} + \frac{n^2\epsilon^2}{18} + \frac{n\epsilon}{6} - 1$ for $n > 42/\epsilon^2$. By Claim 3, this gives us a polynomial time procedure for distinguishing satisfiable 3SAT5 formulas from unsatisfiable formulas — a contradiction unless $P = NP$. □

4 MinDT

A problem similar to ConDT is finding small, equivalent decision trees. This problem, called MinDT, takes as input a decision tree T over n variables of the ConDT type (i.e., a function from $\{0,1\}^n \rightarrow \{0,1\}$) and seeks the smallest decision tree T' (smallest, again, meaning number of leaves) which is functionally equivalent to T. Here, functionally equivalent means that T and T' compute the same function on all binary strings of length n. Zantema and Bodlaender [11] first defined this problem and showed it to be NP-Hard. Sieling [10] showed that MinDT has no constant factor approximation unless P=NP. This negative result follows from a self-improving property of decision trees which is consistent with the squaring results from [6]. We review the self-improving property here, show how to generalize it, and then apply techniques from [6] to show that no $2^{\log^{\delta} s}$ approximation exists (where s is the size of the optimal tree and $\delta < 1$) unless NP $\subseteq$ DTIME[$2^{\text{polylog}\, n}$].

Lemma 2 ([10])**.** *Let f and g be boolean functions over n and m disjoint binary variables respectively. If MIN($f \oplus g$) is the size of the minimum decision tree computing $f \oplus g$ then MIN($f \oplus g$) = MIN(f) · MIN(g). Similarly, if T is a tree of size $|T|$ which computes $f \oplus g$ then T_f, a tree which computes f and T_g, a tree which computes g can be constructed in polynomial time such that $|T_f| \cdot |T_g| \leq |T|$.*

The proof of Lemma 2 divides up T into f-regions and g-regions based on the largest contiguous parts of the tree which examine bits exclusively from f or g. The idea is that regions near the leaves may be switched with their parents to form larger contiguous regions until T_f and T_g are easily identifiable. The generalization of this result requires a few more details:

Lemma 3. *Let F be a set of r binary functions such that $f_i : \{0,1\}^{n_i} \to \{0,1\}$. Let $g = \oplus_{i=1}^{r} f_i$. If MIN(g) is the size of the minimum decision tree computing g then MIN(g) = $\prod_{i=1}^{r}$ MIN(f_i). Similarly, if T is a tree of size $|T|$ which computes g then in polynomial time, we can construct a tree T_i for each f_i such that $\prod_{i=1}^{r} |T_i| \leq |T|$.*

Proof. The proof is similar to the generalization of the squaring technique used in [6]. Call a tree *reduced* when no path contains the same variable twice. In this proof we'll assume all trees are reduced since one can reduce a tree very quickly. It is easy to show that MIN(g) $\leq \prod_{i=1}^{r}$ MIN(f_i) — just build the tree using concatenated copies of the functions in F. This yields a tree for g with size exactly $\prod_{i=1}^{r}$ MIN(n_i). The other direction is more complicated. Let x_i denote a variable from function f_i. A tree is in standard form if, on any path from the root to a leaf, variable x_i is followed only by variables $x_{i'}$ where $i' \geq i$. If the root of a tree (or a subtree) is x_i then let $s(x_i) = (s_1 \ldots s_t)$ denote the roots of the subtrees of x_i which are the first nodes along a path from the root having variables which differ from i.

A tree rooted at x_i is called *subtree equivalent* if every tree rooted at one of $s(x_i)$ is structurally identical except for the leaf labels. Once a tree is in standard form and is subtree equivalent it is easy to recover functions computing each f_i. Hence any subtree equivalent tree computing g in standard form must have size greater than or equal to the product of the size of its composite functions. We often denote a tree by its root, so saying x_i *is in standard form* really means *the tree rooted at x_i is in standard form*. We now show how to take any tree T computing g and turn it into a tree which is subtree equivalent and in standard form. The proof is by induction on the height of subtrees of T which are in standard from and subtree equivalent. It is trivially true that each leaf is subtree equivalent and in standard form. Now consider an internal node x_i with left child $x_{i'}$ and right child $x_{i''}$ both in standard form and subtree equivalent. Without loss of generality, let $i' \leq i''$. We must consider two cases: If $i \leq i'$ then x_i is in standard form. Furthermore, we make the following claim:

Claim 4. *Any two trees from $s(x_i)$ are functionally equivalent up to polarity*

Proof. Suppose two of the $s(x_i)$ trees compute different functions. Then there are two strings w_1 and w_2 that differ only in the f_i variables (specifically those at x_i and beyond) and have different outputs on $g \oplus f_i$ (g with f_i removed). This is a contradiction since $g \oplus f_i$ ignores the f_i variables so $g \oplus f_i(w_1) = g \oplus f_i(w_2)$. □

Now we can take the smallest of the $s(x_i)$ and replicate it across the remaining $s(x_i)$, negating the output on the leaves when appropriate. This tree rooted at x_i is now subtree equivalent and the induction holds. On the other hand, if $i > i'$ then x_i is not in standard form. Using the same type of argument from Claim 4 we can show that $x_{i'}$ and $x_{i''}$ are functionally equivalent up to differences in f_i. If f_i is not constant with respect to x_i then find the tree among $x_{i'}$ and $x_{i''}$ with the smallest total number of nodes leading up to its f_i regions. Without loss of generality, say this is $x_{i'}$. Insert x_i above each f_i subtree in $x_{i'}$ and make f_i the left child of x_i. Now, choose the root of any f_i region from $x_{i''}$ (they're all the same except for the labels) and make it (and its subtree) the right child of each x_i, again, negating the labels when appropriate. It's clear that for $x_i = 0$, the tree rooted at $x_{i'}$ computes the same value as the old tree. To see why it's true for $x_i = 1$, recall that $x_{i'}$ and $x_{i''}$ are functionally equivalent up to f_i so every f_j region ($j \neq i$) must independently be able to compute its proper label, hence all the f_j are sufficient discriminators. By construction $x_{i'}$ is still in standard form and subtree equivalent. We delete $x_{i''}$, promote $x_{i'}$ to root, and by virtue of it having the smaller subtree leading up to each f_i, we know it is smaller than the original x_i tree. □

With Lemma 3 in hand, it's possible to prove the following theorem:

Theorem 4. *For any $\delta < 1$, there is no $2^{\log^{\delta} s}$ approximation for MinDT where s is the size of the optimal tree, unless NP is quasi-polynomial.*

The proof is essentially identical to one given for the analogous ConDT result. We give it here for completeness.

Proof. Suppose such an approximation did exist. Take the given tree T of size n and raise it to the power d where $d = \log^{r}(n)$ where $r = 2\delta/(1-\delta)$. This takes time $n^d = n^{\log^{r}(n)} = 2^{\text{polylog}(n)}$. The smallest solution for T^d has size s^d so our approximation gives us a solution of size at most $s^d 2^{\log^{\delta}(s^d)}$ from which we can find a tree for T of size:

$$s2^{d^{\delta-1}\log^{\delta}(s)} = s2^{\log^{\frac{2\delta(\delta-1)}{1-\delta}}(n)\log^{\delta}(s)} = s2^{\log^{-2\delta}(n)\log^{\delta}(s)} = s2^{\log^{-2\delta}(n)\log^{\delta}(s)}$$

which is $O(s)$. This contradicts the no constant factor approximation result by Sieling. □

Lemma 3 is tantamount to repeatedly squaring the problem and improving the approximation. Here we show that no $\log(n)$ approximation exists for MintDT unless NP is $O(n^{\log\log(n)})$. By increasing the number of iterations we can also prove Theorem 4.

Theorem 5. *If an instance of MinDT with smallest solution size s has a $\log(s)$ approximation then NP is $n^{O(\log\log(n))}$.*

Proof. Suppose a $\log(s)$ approximation exists. Then after the kth iteration of squaring, we have an $(\log(s))^{1/2^k}$ approximation. So after $k = \log(\log(\log(s)))$ iterations we arrive at a constant factor approximation. Since each iteration effectively doubles the problem size, the total time for k iterations is

$$n^{2^k} = n^{2^{\log\log\log(s)}} = n^{\log(\log(s))} = n^{O(\log\log(s))}$$

The theorem follows since $s \leq n$. □

References

[1] Misha Alekhnovich, Mark Braverman, Vitaly Feldman, Adam R. Klivans, and Toniann Pitassi. Learnability and automatizability. In *Proceedings of the 45th Annual Symposium on Foundations of Computer Science*, pages –. IEEE Computer Society Press, Los Alamitos, CA, 2004.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer-Verlag, 1st edition, 1999.

[3] Pierluigi Crescenzi and Viggo Kann. A Compendium of NP Optimization Problems. http://www.nada.kth.se/ viggo/problemlist/.

[4] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[5] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, New York, 1979.

[6] Thomas Hancock, Tao Jiang, Ming Li, and John Tromp. Lower bounds on learning decision lists and trees. *Information and Computation*, 126(2):114–122, 1996.

[7] L. Hyafil and R. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[8] Bernard M. E. Moret. Decision trees and diagrams. *ACM Comput. Surv.*, 14(4):593–623, 1982.

[9] Kolluru Venkata Sreerama Murthy. *On growing better decision trees from data*. PhD thesis, The Johns Hopkins University, 1996.

[10] Detlef Sieling. Minimization of decision trees is hard to approximate. In *IEEE Conference on Computational Complexity*, pages 82–92. IEEE Computer Society, 2003.

[11] H. Zantema and H. L. Bodlaender. Finding small equivalent decision trees is hard. *International Journal on Foundations of Computer Science*, 11(2):343–354, 2000.