Approximating Optimal Binary Decision Trees

Micah Adler[1] and Brent Heeringa[2]

[1] Department of Computer Science, University of Massachusetts, Amherst, 140 Governors Drive, Amherst, MA 01003
micah@cs.umass.edu
[2] Department of Computer Science, Williams College, 47 Lab Campus Drive, Williamstown, MA 01267
heeringa@cs.williams.edu

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 total external path length of the tree is minimized. 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 (but which we call ConDT) have fundamentally different approximation complexity. We conclude with a stronger lower bound for a third decision tree problem called MinDT.

1 Introduction

We consider the problem of approximating optimal binary decision trees. Garey and Johnson [6] 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 an item from X and each internal node is labeled with a 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 depth of a leaf is the length of its path from the root. The total external path length of the tree is the sum of the depths of all the leaves in the tree. The goal of DT is to find a tree which minimizes the total external path length. An equivalent formulation of the problem views each item as an m-bit binary string 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. If no two strings in X are identical, every feasible solution to DT has n leaves. In this paper we always assume the input is a set of unique strings since finding duplicate strings is easily computable in polynomial time. Decision trees have many natural applications (see [9, 11] and references therein) including medical diagnosis (tests are symptoms) and experiment design (tests are experiments which determine some property). In fact, Hyafil and Rivest proved that DT was NP-Complete precisely because "of the large amount of effort that [had] been put into finding efficient algorithms for constructing optimal binary decision trees" [8].

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 noticeable absence of work studying the approximability of DT seems surprising given the age and wide applicability of the problem. However, a close examination of the literature offers some explanation: the name decision tree also refers to a similar but subtly different problem which we call ConDT (for consistent decision tree) that is extremely hard to approximate. The input to ConDT is a set of n positive / negative labeled binary strings, each of length m, called examples[3]. The output is a binary tree where each internal 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

[3] Many papers take m to be the number of examples and take n to be the number of bits.

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.

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^ϵ}] for some $\epsilon < 1$. This improves a result from Hancock et. al. [7] 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 $s = \Omega(n)$.

Our results demonstrate that DT and ConDT – although closely related – are quite different in terms of approximability: ConDT has no $c \ln n$-approximation for any constant c (unless P = NP) whereas our results yield such an approximation for DT for $c > 1$. Also, we show that the lower bounds on learning decision trees of the ConDT type hold when minimizing total external path length instead of minimum size. Note that tree size is not an insightful measure for DT since all feasible solutions have n leaves. Thus, it is the difference in input and output, and not the difference in measure, that accounts for the difference in approximation complexity.

Not surprisingly, the difference in approximation complexity between DT and ConDT combined with the ambiguity of the name decision tree has caused confusion in the literature. For example, [2] and its online incarnation [3], define the decision tree problem according to the DT input and output but cite the negative results for ConDT in Hancock et. al [7]. Therefore, we consider the separation of DT and ConDT in terms of approximation complexity one contribution of our work.

Moret [9] views DT and ConDT as unique instances of a 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. It appears then that labeling restrictions play a crucial role in the complexity of approximating decision trees.

DT shares some similarities with set cover. Since each pair of items is separated exactly once in any valid decision tree, one can view a path from the root to a leaf as a kind of covering of the items. In this case, each leaf defines a set cover problem where it must cover the remaining $n - 1$ items using an appropriate set of bits or tests. In fact, our analysis is inspired by this observation. However, in the decision tree problem, the n set cover problems defined by the leaves are not independent. For example, the bit at the root of an optimal decision tree appears in each of the n set cover solutions, but it is easy to construct instances of DT for which the optimal (independent) solutions to the n set cover instances have no common bits. More specifically, one can construct instances of DT where the n independent set cover problems have solutions of size 1, yielding a decision tree with cost $\Theta(n^2)$ but where the optimal decision tree has cost $O(n \log(n))$. Hence, the interplay between the individual set cover problems appears to make the DT problem fundamentally different from set cover. Conversely, set cover instances naturally map to decision tree instances, however, the difference in cost between the two problems means that the optimal set cover is not necessarily the optimal decision tree.

The min-sum set cover (MSSC) problem is also similar to DT. The input to MSSC is the same as set cover (i.e., a universe of items X and a collection C of subsets of X), but the output is a linear ordering of the sets from 1 to $|C|$. If $f(x)$ gives the index of the first set in the ordering that covers x then the cost of the ordering is $\sum_{x \in X} f(x)$. This is similar, but not identical to the cost of the corresponding DT problem because the covered items must still be separated from one another, thus adding additional cost. Greedily selecting the set which covers the most remaining uncovered items yields a 4-approximation to MSSC [5, 10]. This approximation is tight unless **P=NP**. As with set cover, we can think of DT as n instances of MSSC, but again, these instances are not independent so the problems inherent in viewing DT as n set cover problems remain when considering DT as n instances of MSSC.

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**. We also show that the lower bounds on learning ConDTs hold for total external path length. Finally, we conclude with a discussion of some open problems including the gap between the upper and lower bounds on the approximation ratio.

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
```

Fig. 1. A greedy algorithm for constructing decision trees.

A straightforward implementation on this algorithm runs in time $O(mn^2)$. While the algorithm does not always give an optimal solution, it does approximate it within a factor of $\ln n + 1$.

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 with cost $\mathcal{C}$. An unordered pair of items $\{x, y\}$ (hereafter just *pair of items*) is *separated* at an internal node S if x follows one branch and y follows the other. Note that each pair of items is separated exactly once in any valid decision tree. Conversely, each internal node S defines a set $\rho(S)$ of pairs of items separated at S. That is

$$\rho(S) = \{\{x, y\} \mid \{x, y\} \text{ is separated at } S\}$$

For convenience we also use S to denote the set of items in the subtree rooted at S. Let S^+ and S^- be the two children of S such that $|S^+| \geq |S^-|$. Note that $|S| = |S^+| + |S^-|$. 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:

$$\mathcal{C} = \sum_{S \in \mathcal{T}} |S|$$

Our analysis uses an accounting scheme to spread the total cost of the greedy tree among all unordered pairs of items. Since each set S contributes its size to the total cost of the tree, we spread its size uniformly among the $|S^+||S^-|$ pairs of items separated at S. Let c_{xy} be the pair cost assigned to each pair of items $\{x, y\}$ where

$$c_{xy} = \frac{1}{|S^+_{xy}|} + \frac{1}{|S^-_{xy}|}.$$

and S_{xy} separates x from y. We can now talk about the cost of a tree node S by the costs associated with the pairs of items separated at S. Summing the costs of these pairs is, by definition, exactly the size of S:

$$\sum_{\{x,y\}\in\rho(S)} c_{xy} = |S^+||S^-|\Big(\frac{1}{|S^+|} + \frac{1}{|S^-|}\Big) = |S|$$

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

$$\mathcal{C} = \sum_{\{x,y\}} c_{xy}$$

Now consider the optimal tree $\mathcal{T}^*$ for X. If Z is an internal node of $\mathcal{T}^*$ then we also use Z to denote the set of items that are leaves of the subtree rooted at Z. Following our notational conventions, we let Z^+ and Z^- be the children of Z such that $|Z^+| \geq |Z^-|$ and $|Z| = |Z^+| + |Z^-|$. The cost of the optimal tree, $\mathcal{C}^*$, is

$$\mathcal{C}^* = \sum_{Z\in\mathcal{T}^*} |Z| \tag{1}$$

Since, every feasible tree separates each pair of items exactly once, we can rearrange the greedy pair costs according to the structure of the optimal tree:

$$\mathcal{C} = \sum_{Z\in\mathcal{T}^*} \sum_{\{x,y\}\in\rho(Z)} c_{xy} \tag{2}$$

If Z is a node in the optimal tree, then it defines $|Z^+||Z^-|$ pairs of items. Our goal is to show that the sum of the c_{xy} associated with the $|Z^+||Z^-|$ pairs of items split at Z (but which are defined with respect to the greedy tree) total at most a factor of $H(|Z|)$ more than $|Z|$ 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 internal node Z in the optimal tree:*

$$\sum_{\{x,y\}\in\rho(Z)} c_{xy} \leq |Z|H(|Z|)$$

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

Proof. Consider any node Z in the optimal tree. For any unordered pair of items $\{x, y\}$ split at Z, imagine using the bit associated with the split at Z on the set S_{xy} separating x from y in the greedy tree. Call the resulting two sets $S_{xy}^{Z^+}$ and $S_{xy}^{Z^-}$ respectively. Since the greedy split at S_{xy} minimizes c_{xy}, we know

$$c_{xy} = \frac{1}{|S_{xy}^+|} + \frac{1}{|S_{xy}^-|} \leq \frac{1}{|S_{xy}^{Z^+}|} + \frac{1}{|S_{xy}^{Z^-}|} \leq \frac{1}{|S_{xy} \cap Z^+|} + \frac{1}{|S_{xy} \cap Z^-|}.$$

Hence

$$\sum_{\{x,y\}\in\rho(Z)} c_{xy} \leq \sum_{\{x,y\}\in\rho(Z)} \frac{1}{|S_{xy} \cap Z^+|} + \frac{1}{|S_{xy} \cap Z^-|} \tag{3}$$

One interpretation of the sum in (3) views each item x in Z^+ as contributing

$$\sum_{y\in Z^-} \frac{1}{|S_{xy} \cap Z^-|}$$

to the sum and each node y in Z^- as contributing

$$\sum_{x \in Z^+} \frac{1}{|S_{xy} \cap Z^+|}$$

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

Claim. For any $x \in Z^+$ we have

$$\sum_{y \in Z^-} b_{xy} = \sum_{y \in Z^-} \frac{1}{|S_{xy} \cap Z^-|} \leq H(|Z^-|)$$

Proof. If Z^- has m items then let $(y_1, \ldots, y_m)$ be an ordering of Z^- in reverse order from when the items are split from x in the greedy tree (with ties broken arbitrarily). This means item y_1 is the last item split from x, y_m is the first item split from x, and in general y_{m-t+1} is the t^{th} item split from x. When y_m is split from x there must be at least $|Z^-|$ items in S_{xy_m} — by our ordering the remaining items in Z^- must still be present — so $Z^- \subseteq S_{xy_m}$. Hence b_{xy_m}, the cost assigned to x on the edge (x, y_m), is at most $1/|(Z^-)|$ and in general, when y_t is separated from x there are at least t items remaining from Z^-, so the cost b_{xy_t} assigned to the edge (x, y_t) is at most $1/t$. This means, for any $x \in Z^+$

$$\sum_{y \in Z^-} b_{xy} \leq H(|Z^-|)$$

which proves the claim. □

We can use the same argument to prove the analogous claim for all the items in Z^-. With these inequalities in hand we have

$$\begin{aligned}
\sum_{\{x,y\} \in \rho(Z)} \frac{1}{|S_{xy} \cap Z^+|} + \frac{1}{|S_{xy} \cap Z^-|} &\leq |Z^+|H(|Z^-|) + |Z^-|H(|Z^+|) \\
&< |Z^+|H(|Z|) + |Z^-|H(|Z|) \\
&< |Z|H(|Z|) \qquad \text{(since } |Z^+| + |Z^-| = |Z|\text{)}
\end{aligned}$$

□

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

$$\sum_{Z \in \mathcal{T}^*} \sum_{\{x,y\} \in \rho(Z)} c_{xy} \leq \sum_{Z \in \mathcal{T}^*} |Z|H(|Z|) \leq \sum_{Z \in \mathcal{T}^*} |Z|H(n) = H(n)\mathcal{C}^* \leq (\ln n + 1)\mathcal{C}^*$$

□

2.1 Tests with Weights

In many applications, different tests may have different execution costs. For example, in experiment design, a single test might be a good separator of the items, but it may also be expensive. Running multiple, inexpensive tests may serve the same overall purpose, but at less cost. To model scenarios like these we associate a weight $w(k)$ with each bit k and without confusion take $w(S)$ to be the weight of the bit used at node S. We call this problem DT with weighted tests. In the original problem formulation, we can think of each test as having unit weight, so the cost of identifying an item is just the length of the path from the root to the item. When the tests have non-uniform weights, the cost of identifying an item is the sum of the weights of the tests along that path. We call this the path cost. The cost of the tree is the sum of the path costs of each item. When all the tests have equal weight, we choose the bit which most evenly splits the set of items into two groups. In other words, we minimize the pair cost c_{xy}. With equal weights, the cost of an internal node is just its size $|S|$. With unequal weights, the cost of an internal node is the weighted size $w(S)|S|$, so assuming S separates x from y the pair cost becomes

$$c_{xy} = \frac{w(S)}{|S^+|} + \frac{w(S)}{|S^-|} \tag{4}$$

and our new greedy algorithm recursively selects the bit which minimizes this quantity. This procedure yields a result equivalent to Theorem 1 for DT with weighted tests. A straightforward implementation on this algorithm still runs in time $O(mn^2)$.

Theorem 2. *The greedy algorithm which recursively selects the bit that minimizes Equation 4 yields a* $(\ln n + 1)$*-approximation to DT with weighted tests.*

Proof. Following the structure of the proof for Theorem 1 leads to the desired result. The key observation is that choosing the bit that minimizes Equation 4 yields the inequality

$$c_{xy} \leq w(Z)\Big(\frac{1}{|S_{xy} \cap Z^+|} + \frac{1}{|S_{xy} \cap Z^-|}\Big). \tag{5}$$

Since the weight term $w(Z)$ may be factored out of the summation

$$w(Z) \sum_{\{x,y\} \in \rho(Z)} \frac{1}{|S_{xy} \cap Z^+|} + \frac{1}{|S_{xy} \cap Z^-|}$$

we can apply the previous claim and the theorem follows:

$$\sum_{Z \in \mathcal{T}^*} \sum_{\{x,y\} \in \rho(Z)} c_{xy} \leq \sum_{Z \in \mathcal{T}^*} w(Z)|Z|H(n) \leq (\ln n + 1)\mathcal{C}^*$$

Here $C^* = \sum_{Z \in \mathcal{T}^*} w(Z)|Z|$ is the cost of the optimal tree. □

Another natural extension to DT considers the problem with weighted items. Here, one weights each path length by the weight of the item which defines the path. Unfortunately, our analysis does not hold for this case. We discuss this further in Section 5.

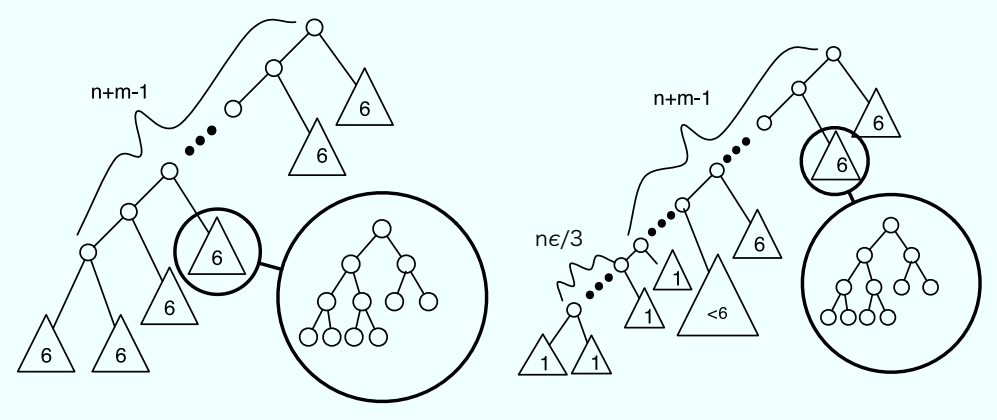

Fig. 2. (left) The topology of the optimal tree. Here the $n + m - 1$ bits along the left branch correspond to the sets in the set cover. (right) This tree represents the *best case* when $n+m+\frac{n\epsilon}{3}$ sets are required to cover the items. To minimize the cost in this scenario, the extra bits cover a single item and the top of the tree is filled with subtrees of size 6.

3 Approximating DT is Hard

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 give 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: with total external path length, cost is a function of leaf 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 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.

The reduction from 3SAT5 to the bounded-size set cover problem uses a variation of the reduction used by Sieling [12] to show that MinDT (a problem quite similar, but distinct from ConDT) has no PTAS.

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. Let $Y(i)$ be the indices of the 5 clauses in which x_i appears. First we define the set of items. For each variable x_i,

create 11 items: y_i, a_{ij} (for j in $Y(i)$), and b_{ij} (again, for j in $Y(i)$). 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:

$$S_i^a = \{y_i\} \cup \{a_{ij} \mid j \in Y(i)\} \qquad \text{and} \qquad S_i^b = \{y_i\} \cup \{b_{ij} \mid j \in Y(i)\}$$

Call these sets the *variable* sets since they are constructed from the variables of the formula. For each clause C_j, create seven sets called the *clause sets*. 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 exactly 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. Since there are seven locally satisfying assignments per clause, there are a total of $7m$ sets in Z. Note that every set has exactly 6 items. The key to the reduction is that the a_{ij} correspond to a positive assignment of x_i and the b_{ij} correspond to a negative assignment to x_i. The clause sets include the items which are opposite of the assignment, so when a formula is satisfiable, it is always possible to cover it with $n + m$ sets. However, if no satisfying assignment exists for an assignment then more sets are required. Though our reduction is not identical to the one given by Sieling, the following theorem still holds:

Theorem 3 ([12]). *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 $n + m + \frac{\epsilon n}{3}$.*

Proof. It's clear that $n + m$ sets are necessary to cover U, but when ϕ is satisfiable, $n + m$ sets are also sufficient. If z is a satisfying assignment to ϕ then for each variable x_i, choose S_i^a if $z(x_i) = 1$ or choose S_i^b if $z(x_i) = 0$. For each clause j, choose S_j^z where S_j^z is the set for clause j corresponding to assignment z. It's clear that all the y_i and c_j items are covered. Every a_{ij} item is also covered: If $z(x_i) = 1$ then S_i^a covers it. If $z(x_i) = 0$ then S_j^z covers it. The b_{ij} cases are symmetric, so $n + m$ sets are sufficient. If ϕ is not a satisfying assignment then more sets are required. To see this, let Y be a solution to the the set cover problem. Let Y' be a subset of Y such that for all i, Y' contains the first occurrence of a set covering y_i and, for all j, the first occurrence of the set covering c_j. The variable sets of Y' define an assignment z to the variables. At least ϵm clauses aren't satisfied by this assignment. Let C_j be such a clause, say (x_1, x_2, x_3). Since it isn't satisfied, $z(x_i) = 0$ for $i \in [1, 3]$. So all the b_i for $i \in [1, 3]$ are covered by the variables sets, but the clause set j cannot cover the remaining a_i because, by construction, it doesn't exist. To cover the remaining items requires at least one more variable set. This has the potential of covering at most 5 other uncovered items. So if ϕ is unsatisfiable we require at least $\frac{\epsilon m}{5} = \frac{\epsilon n}{3}$ more sets. For more details, we refer the reader to [12]. 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 $|Z| = m'$. 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 the m' sets 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 Z-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, and so on. See Figure 2 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' and m'.

Claim. 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 on the left in Figure 2. The bits along the left side correspond to the optimal set cover, except for the final 3 bits (which are used to disambiguate the final remaining set). 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 entirely 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. 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 3 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 $n + m$ internal nodes. In the best case, these will be subtrees of size 6 where possible. This is illustrated on the right in Figure 2. Another way to think about this tree is starting with the optimal tree from Claim 3.1 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 the cost since the first leaf shifted adds at least 1 to the total cost, the second leaf shifted adds at least 2 to the total 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 3.1 and 3.1 yields the desired result:

Theorem 4. *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 $\delta < \frac{\epsilon^2}{912}$. 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 $n > \frac{6}{\epsilon}$. By Claim 3.1, this gives us a polynomial time procedure for distinguishing satisfiable 3SAT5 formulas from unsatisfiable formulas — a contradiction unless **P=NP**. □

3.2 Hardness of Approximation for ConDT under Total External Path Length

Alekhnovich et. al. [1] showed 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^ϵ}] for some $\epsilon < 1$. Decision tree here refers to trees of the ConDT type and the measure here is tree size. In this section we show that these hardness results also hold when for ConDT under minimum total external path length. Our theorem relies on the observation that if I is an instance of ConDT with minimum total external path length s then I has minimum tree size at least $\Omega(\sqrt{s})$. If it didn't, a tree of smaller size would have smaller total external path length, a contradiction. The case where minimum total external path length s corresponds to minimum size $\Omega(\sqrt{s})$ is a cascading tree; that is, a tree with exactly one leaf at each depth save the deepest two.

Theorem 5. *If there exists an s^k approximation for some constant $k > 0$ to decision trees with minimum total external path length s then NP is contained in DTIME[2^{m^ϵ}] for some $\epsilon < 1$.*

Proof. Let I be an instance of ConDT with minimum total external path length $s = r^2$. It follows that I has minimum tree size at least $\Omega(r)$. Now, if an s^k approximation did exist for some k then there would exist an $\Omega(r^{2k}) = r^{k'}$ approximation for some constant k' for ConDT under minimum tree size; a contradiction. □

4 New Lower Bounds on Approximating 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, in terms of 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 [13] first defined this problem and showed it to be **NP**-hard. Sieling [12] 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 [7]. We review the self-improving property here, show how to generalize it, and then apply arguments and techniques from [7] 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 ([12]). *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 that computes f and T_g, a tree that 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 [7]. 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. 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 are 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 6. *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 $d = \log^r(n)$ and 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 MinDT unless **NP** is contained in $O(n^{\log\log(n)})$ time. By increasing the number of iterations we can also prove Theorem 6.

Theorem 7. *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 $s \leq n$.

5 Open Problems and Discussion

In this paper, we present a $(\ln n + 1)$-approximation for the decision tree problem. Besides being the first non-trivial upper bound on the approximation ratio, the result serves a second purpose; it separates the approximation complexity of DT from a similar, hard-to-approximate decision tree problem which we call ConDT. In addition we give a polynomial-time reduction from MAX-3SAT5 to DT which preserves the ϵ gap between satisfiable formulas and unsatisfiable formulas. This means DT does not have a PTAS unless **P=NP**.

The most prominent open problem is the gap between the upper and lower bounds on the approximation ratio of DT. Amplifying the gap using techniques from [7] for ConDT does not work for DT. There, one squares an instance of ConDT, applies an α-approximation, and recovers a solution to the original instance which is a $\sqrt{\alpha}$-approximation. Repeating this procedure yields the stronger lower bound. This does not work for DT because the average path length only doubles when squaring the problem, so solving the squared problem with an α-approximation and recovering a solution to the original problem simply preserves (and unfortunately does not improve) the approximation ratio. The hardness results from [1] rely on the construction of a binary function which is difficult to approximate accurately when certain instances of a hitting-set problem have large solutions. These techniques do not appear to work for DT either. Improving the upper bound also seems reasonable. For example, we have yet to find a family of DT instances where our analysis is tight for the greedy algorithm.

The second open problem involves DT with weighted items. While our analysis generalizes to tests with weights, it does not work for items with weights. This is because our method does not allow us to easily compare the weight of a node in the greedy tree with the weight of a node in the optimal tree when the weight of a node is determined not by the number of items in its subtree, but the total weight of the items in the subtree. Overcoming this obstacle presents an interesting analytical challenge.

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 621–630. 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. Uriel Feige, László Lovász, and Prasad Tetali. Approximating min-sum set cover. *Algorithmica*, 40(4):219 – 234, September 2004.
6. 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.
7. 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.
8. L. Hyafil and R. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.
9. Bernard M. E. Moret. Decision trees and diagrams. *ACM Comput. Surv.*, 14(4):593–623, 1982.
10. Kamesh Munagala, Shivnath Babu, Rajeev Motwani, and Jennifer Widom. The pipelined set cover problem. In *ICDT*, pages 83–98, 2005.
11. Kolluru Venkata Sreerama Murthy. *On growing better decision trees from data*. PhD thesis, The Johns Hopkins University, 1996.
12. Detlef Sieling. Minimization of decision trees is hard to approximate. In *18th Annual IEEE Conference on Computational Complexity*, pages 82–92. IEEE Computer Society, 2003.
13. 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.