Relating decision tree complexity with AND and OR decision tree complexity

Anil Ada, Yeongwoo Hwang, Kumail Jaffer, Sidhanth Mohanty, Ryan O'Donnell, Yu Zhao

December 13, 2017

1 Introduction

In the decision tree model, there is a total function $f : \{0,1\}^n \to \{0,1\}$, and an unknown string x and the goal is compute $f(x)$. Access to x is via an oracle where one asks queries of the form "what is in the ith bit of x?". The goal is to compute $f(x)$ with as few oracle queries as possible. We define $\mathsf{DT}(f)$ as the minimum number of oracle queries to an input x needed to determine $f(x)$. The reason we call this model a 'decision tree' is that any algorithm can be represented as a binary tree – the root vertex can be thought of as the first query, and the left branch can be thought of as the remaining algorithm given the answer to the first query was 1 and the right branch can be thought as the algorithm given the answer to the first query was 0. The leaves of a decision tree is either a 0 or a 1, indicating the output of the query algorithm. A single lone vertex denotes an algorithm that always outputs one value. And $\mathsf{DT}(f)$ can also be thought of as the minimum depth decision tree that computes f correctly.

The AND and OR decision tree models are variants of the decision tree model where queries are of the form "what is the AND of all the bits in S" where S is a nonempty subset of indices, and the oracle returns $\wedge_{i\in S} x_i$ (the OR model is defined similarly).

A theorem from [BI87] gives the following relation between the CNF and DNF complexity of a total function f to $\mathsf{DT}(f)$:

$$\mathsf{DT}(f) \le \mathrm{CNF}(f)\mathrm{DNF}(f)$$

where $\mathrm{CNF}(f)$ and $\mathrm{DNF}(f)$ are the number of clauses in the longest CNF/DNF.

A natural result that one could hope for in the world of decision trees is the statement

$$\mathsf{DT}(f) \le \mathsf{DT}_{\mathsf{AND}}(f) \cdot \mathsf{DT}_{\mathsf{OR}}(f)$$

Note that this conjecture would be implied by the stronger statement

$$\mathsf{DT}(f) \le \max\{\mathsf{DT}_{\mathsf{AND}}(f), \mathsf{DT}_{\mathsf{OR}}(f)\}$$

However this is false, although ruling it out is nontrivial. In this writeup, we exhibit a function f such that both $\mathsf{DT}_{\mathsf{AND}}(f)$ and $\mathsf{DT}_{\mathsf{OR}}(f)$ are $o(\mathsf{DT}(f))$ using the method of cheatsheets.

We emphasize that f is total.

Our construction is inspired and uses many ideas in a construction of [ABDK16] in showing other separations in query complexity. This method, based on cheat sheets and pointer functions has been useful in showing other separations in query complexity and communication complexity in recent past, for example [ABBD$^+$16, ABB$^+$17].

2 The Construction

2.1 A Protofunction

We build up ingredients one by one for ease of exposition. Consider the following two functions AllZerosCol and AllOnesCol from $\{0,1\}^{m\times m}$ to $\{0,1\}$, which are 1 if the matrix has an all zeros column (or an all ones column respectively) and 0 otherwise. We note that the AND decision tree can solve AllOnesCol in m queries and the OR decision tree can solve AllZerosCol in m queries (simply query each column) whereas the regular decision tree needs m^2 queries.

Lemma 1. $DT(\text{AllOnesCol}) = m^2$

Proof sketch. Answer queries adversarially in the following way: if an index (i,j) is not the last index to be queried in it's column answer 1. If it is the last index to be queried but not the last bit to be queried in the matrix, answer 0. Even after m^2-1 queries are made, the adversary can still contrive the input for AllOnesCol to evaluate to 1 or 0. □

A similar argument shows $\mathsf{DT}(\text{AllZerosCol})$ is also m^2.

2.2 Cheatsheets and Gadgets

As the next ingredient, we introduce the notion of a cheat sheet. We showed that it takes m^2 queries for a regular decision tree to solve AllZerosCol. For a given instance of AllZerosCol called x, a cheatsheet for the x specifies the value of AllZerosCol(x) and a certificate to verify this value: given by a pointer to a column of all 0's (specified by an index i) or m pointers of the form (i,j) that point to a 1 in each column.

We call a cheatsheet valid for an instance x of AllZerosCol if

1. It contains the correct answer for AllZerosCol.
2. If the answer is 1, the pointer to the column of all zeros in x indeed has all zeros.
3. If the answer is 0, the m pointers to each column indeed point to 1s.

Given a cheatsheet for an instance x for AllZerosCol, a standard decision tree can solve the problem in m queries if the cheatsheet is valid, or detect that the cheatsheet is invalid.

A cheatsheet for AllOnesCol is defined analogously.

As a remark of foresight, we note that the AND decision tree can solve AllZerosCol in m queries using a valid cheatsheet (same thing for OR and AllOnesCol).

A gadget comprises of two $m\times m$ matrices x_1 and x_2, and we call a gadget valid if

$$\text{AllZerosCol}(x_1) = \text{AllOnesCol}(x_2)$$

For a valid gadget $T=(x_1,x_2)$, define $g(T)$ as AllZerosCol(x_1) (equal to AllOnesCol(x_2)).

2.3 The Function

The input to our function is $100\log m$ gadgets along with an array of length m^{100}, with each entry of the array being $200\log m$ cheatsheets, the $(2i-1)$th and $2i$th cheatsheet meant for the ith gadget (these could be potentially invalid cheatsheets).

Our function is 1 if the following conditions on the input are satisfied.

1. All $100\log n$ gadgets $T_1, T_2, \ldots, T_{100\log m}$ are valid.

2. Let x be the binary integer denoted by $g(T_1)g(T_2)\dots g(T_{100\log m})$. The condition we want satisfied is that the $(2i-1)$th and $2i$th cheatsheet in the xth entry in the array given to our function as input are valid cheatsheets for gadget T_i.

We call this function SEPARATOR.

3 Separations

3.1 AND and OR decision tree protocols of complexity $\widetilde{O}(m)$

We give a protocol for the OR decision tree. A protocol for the AND decision tree can be obtained analogously.

Lemma 2.

$$DT_{OR}(\textsc{Separator}) = \widetilde{O}(m)$$

Proof. The OR decision tree can solve ALLZEROSCOL on the first matrix in each gadget of the input, a procedure that will take $100m\log m$ queries. The solutions to ALLZEROSCOL on each gadget can be interpreted as the binary representation of an integer x. The algorithm then queries all the $200\log m$ cheatsheets stored in the xth entry of the array. Each cheatsheet takes at most $\widetilde{O}(m)$ bits to represent because it contains one bit for the answer and at most m pointers. The OR decision tree verifies the validity of each cheatsheet, and if there is even a single invalid cheatsheet, it returns 0. If each cheatsheet is valid, it can also solve ALLONESCOL on the second matrix in each gadget, and if even one gadget is not valid, it returns 0. If every gadget is valid, and every cheatsheet is valid too, the algorithm returns 1. Verifying each cheatsheet and solving ALLONESCOL if the cheatsheets are valid also takes at most $200m\log m$ queries. Overall, we use only $\widetilde{O}(m)$ queries. □

Lemma 3.

$$DT_{AND}(\textsc{Separator}) = \widetilde{O}(m)$$

Proof. Analogous to lemma 2. □

3.2 An $\Omega(m^2)$ lower bound for decision trees

Lemma 4.

$$DT(\textsc{Separator}) = \Omega(m^2)$$

Proof. Answer all queries made to the gadgets adversarially as per the proof of lemma 1 and while the number of queries made to cheatsheets is less than m^2, answer any query on the array of cheatsheets with a 0. If an algorithm responds with 1 in strictly less than m^2 queries, then if the unrevealed part of the array of cheatsheets only has 0's, the cheatsheets are not consistent with the gadgets and the correct answer is 0, which means the algorithm's answer was incorrect. On the other hand, say an algorithm responds with 0 by making strictly less than m^2 queries. Then pick an index i such that no queries have been made to the ith entry in the array of cheatsheets and consider the following possibility for unqueried bits. The unrevealed bits of the gadgets are contrived so that the integer obtained from the $200\log m$ gadgets is equal to i. This is possible from the proof of lemma 1 since the algorithm made strictly less than m^2 queries and hence didn't fully query any of the gadget matrices. The ith entry of the cheatsheet array is set to a valid cheatsheet for the corresponding gadgets. The algorithm would output 0 on this input when the correct output is 1.

□

4 $\mathsf{DT}_{\mathsf{AND}}(f) + 2^{\mathsf{DT}_{\mathsf{OR}}(f)} \geq \mathsf{DT}(f)$

4.1 Algorithm from OR and AND decision trees

Here we prove a weaker version of the result we were aiming for.

For some function f, suppose we have an AND decision tree T_{AND} that incurs cost c_1 and an OR decision tree T_{OR} that incurs cost c_2, then consider the following algorithm in the standard query model.

1. Let S_{AND} be the current vertex, initialized to the root of T_{AND}, and let S_{OR} be a tree initialized to T_{OR}.
2. If S_{AND} or S_{OR} is a lone vertex, then return the value at that vertex.
3. Pick a variable x from the root of S_{AND} that also occurs at some vertex in S_{OR} and query x.
4. If x is a 0, then update S_{AND} to the right branch of S_{AND}. Otherwise, delete all occurrences of x in S_{AND}, and for every vertex v in S_{OR} where x occurs, replace the subtree rooted at v with the subtree in the left branch of v.
5. If there is no variable at the root of S_{AND} that also occurs somewhere in S_{OR}, then update S_{AND} to the subtree in it's left branch. Go back to step 2.

This algorithm simultaneously simulates the AND and OR decision tree and picks whichever tree reaches the answer first and thus assuming correctness of the AND and OR decision tree in computing f, we know our algorithm computes f correctly.

For each query, either the height of S_{AND} decreases by 1, or the number of vertices in S_{OR} decreases by 1. c_2 is the depth of T_{OR} and hence there are at most 2^{c_2} vertices in the tree. So after at most $c_1 + 2^{c_2}$ queries, one of the decision trees reaches a leaf. This establishes that $\mathsf{DT}(f) \leq 2^{\mathsf{DT}_{\mathsf{OR}}}(f) + \mathsf{DT}_{\mathsf{AND}}(f)$. Note that the same argument can be used to show $\mathsf{DT}(f) \leq 2^{\mathsf{DT}_{\mathsf{AND}}(f)} + \mathsf{DT}_{\mathsf{OR}}(f)$.

4.2 A Quick Application

As a quick application, we show a lower bound of $\Omega(n)$ for the AND decision tree to compute the OR of the n bits of x. It is easy to see a lower bound of $\Omega(n)$ on the standard decision tree complexity since an adversary can answer 0's to any query algorithm until the very last query. And there is a 1 query protocol for computing the OR of x in the OR decision tree model, which means

$$n \leq \mathsf{DT}(\mathrm{OR}) \leq 2^{\mathsf{DT}_{\mathsf{OR}}(\mathrm{OR})} + \mathsf{DT}_{\mathsf{AND}}(\mathrm{OR}) \leq 2 + \mathsf{DT}_{\mathsf{AND}}(\mathrm{OR})$$

This immediately gives a lower bound of $n-2$ queries on $\mathsf{DT}_{\mathsf{AND}}(\mathrm{OR})$.

5 Main result to aim for

Is the conjecture that $\mathsf{DT}(f) \leq \mathsf{DT}_{\mathsf{AND}}(f)\mathsf{DT}_{\mathsf{OR}}(f)$ true or false? If it is true, then one can imagine that this could be used as a tool to prove lower bounds for some problems in the AND and OR decision tree models, in a style akin to section 4.2.

References

[ABB$^+$17] Andris Ambainis, Kaspars Balodis, Aleksandrs Belovs, Troy Lee, Miklos Santha, and Juris Smotrovs. Separations in query complexity based on pointer functions. Journal of the ACM (JACM), 64(5):32, 2017.

[ABBD$^+$16] Anurag Anshu, Aleksandrs Belovs, Shalev Ben-David, Mika Göös, Rahul Jain, Robin Kothari, Troy Lee, and Miklos Santha. Separations in communication complexity using cheat sheets and information complexity. In Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on, pages 555–564. IEEE, 2016.

[ABDK16] Scott Aaronson, Shalev Ben-David, and Robin Kothari. Separations in query complexity using cheat sheets. In Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, pages 863–876. ACM, 2016.

[BI87] Manuel Blum and Russell Impagliazzo. Generic oracles and oracle classes. In Foundations of Computer Science, 1987., 28th Annual Symposium on, pages 118–126. IEEE, 1987.