# Lecture 7. Decision Tree Complexity and Fourier analysis

## Decision Tree Complexity

Recall that in the first lecture, we introduced decision tree as a model to compute a function. Basically, we make a sequence of queries of the form and finally output the answer . The (deterministic) decision tree complexity, or query complexity, of a function *f*, is the minimum number of queries a decision tree needs to make in order to compute *f*. It’s interesting that some functions have sublinear query complexity, namely one can always get to know long before learning *x* itself. We’ve seen a couple of such examples. Let’s see one more.

**Example: Sink problem**. We are given a directed graph, and we wish to know whether the graph has a “sink” vertex, which has out-degree n-1 and in-degree 0. (Namely, everyone else points to it, but it doesn’t point to anyone.) What we can ask is for any ordered pair (i,j), whether there is an edge from i to j. What’s the minimum number of queries we need?

We can also consider randomized decision trees, which can make random choices during the computation. Two types of requirement for correctness are used:

* Monte Carlo: On any input, the algorithm finishes in at most *q* queries and is correct with probability .
* Las Vegas: On any input, the algorithm is correct with probability 1, and the expected the number of queries is at most *q*.

The randomized query complexity is thus the minimum number *q* of queries needed. Denote by and the randomized query complexity of *f* in the Monte Carlo and Las Vegas models, respectively. Similar as in the communication complexity case, for any distribution on input set , we can define the -distributional query complexity as the minimum number of queries needed for any deterministic query algorithm which has error probability, averaged over , at most . Also define as the minimum expected (over ) number of queries by any deterministic query algorithm that computes *f* correctly on all inputs. By Yao’s minmax principle, the following holds.

**Theorem 1.1**.

While some functions enjoy small decision tree complexity, many don’t. As always in complexity theories, we hope to understand what functions have small decision tree complexity.

An important class of functions are those with symmetries. For example, consider

* .
* .
* , and otherwise.

These functions are “very symmetric”. Actually they depend only on the number of 1’s in ---it’s not important which variables are 1’s.

Some other functions are “less symmetric”. For example, consider all graph properties---functions on graphs that don’t depend on the labeling of the vertices. That is, the input to the function has variables, indicating whether each pair is an edge in the graph. The function depends only on the graph structure, not the name of vertices. Formally, *f* is invariant to vertex permutations. If we view the input G as an matrix AG (which is symmetric with all diagonal entries being 0), and simultaneously perform a permutation to rows and to columns, then remains unchanged. Examples include the functions that we are familiar, such as

* Hamiltonian Cycle: Does the graph have a Hamiltonian cycle?
* k-Clique: Does the graph have a clique of size k?
* Connectivity: Is the graph connected?
* Bipartiteness: Is the graph a bipartite graph?
* Matching: Does the graph have a perfect matching?
* …

Much attention has been drawn on the effect of symmetry on the decision tree complexity. Aanderaa once conjectured that all graph properties have decision tree complexity , namely one has to ask all variables. Functions f on N variables with are called *evasive*. It was soon found by Rosenberg that this is not correct. Consider the Sink problem at the beginning of this lecture. (Well, technically speaking, Sink is a directed graph property. But there is also an analog called Scorpion, which is indeed a graph property, and the decision tree complexity is ) So Rosenberg added one more condition: *monotonicity*. This leads to the most famous conjecture in decision tree complexity:

**Aanderaa-Rosenberg Conjecture**: All monotone graph properties are evasive.

While this conjecture was not fully solved, it was known that such functions need queries. It can also be proved that if the input size *N* is a prime power [RV76]. See [LY94] for a survey of evasiveness.

However, things become much more complicated when it comes to randomized query complexity. Karp modified the conjecture to the following.

**Aanderaa-Karp-Rosenberg Conjecture**: All monotone graph properties *f* of *N* variables have .

This conjecture was still wide open despite many attacks by first-tier researchers. It’s considered to be one of the few most notorious conjectures in theoretical computer science. The best lower bound that we have is . (Notation: Tilde hides a log factor.) There are two proofs of this fact. One is by graph packing, initialized by Yao [Yao87] and improved by King [Kin88], Hajnal [Haj91], Chakrabarti-Khot [CK01]. The other approach is by Fourier analysis. Both are very beautiful, but the second one is simpler, which we’ll talk about.

First we’ll need some basic knowledge of Fourier analysis on Boolean functions.

## Fourier analysis

There are two common ways to represent Boolean values: or . To emphasize the difference, we will use *x* and *f* for and *y* and *g* for . For each , define by (or equivalently .) The following fact is easy to verify.

**Fact 2.1**. is an orthonormal basis of the space under the inner product .

By the above fact, any real function h on can be expanded in this basis. We can write it as , where the coefficients are called the *Fourier coefficients*. The transform from h to is the *Fourier transform*. A basic property is about the transform is that it keeps the -norm.

**Fact 2.2**. **(Parseval’s formula)** .

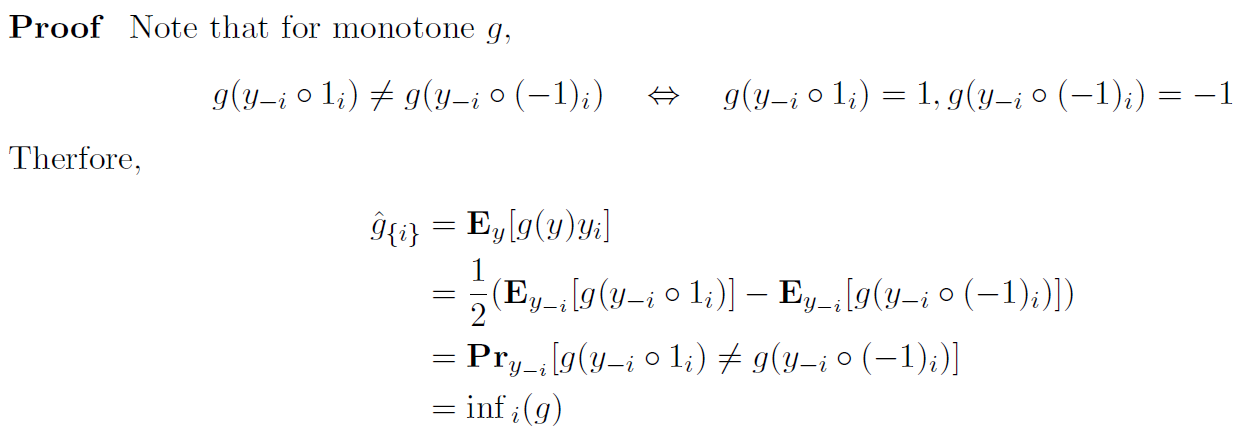
Thus for Boolean functions of range , we have . So can be viewed as a probability distribution over .

An important concept is that of the influence of variables. Suppose we pick input *x* randomly by letting eachwith probability p. Denote by this probability distribution of *x*. For a Boolean function h, define

where is the string obtained from *x* by flipping the i-th variable. Intuitively, the influence of variable *i* is the probability that it matters for the function, when other input variables are drawn randomly. When we omit it and just write .

Let be the n-bit string where all bits except the i-th one are 0. We usually denote .

**Fact 2.3**. If a Boolean function is monotone, then



We finally need a fact about symmetry of influences.

**Fact**. For graph properties, all the influences are the same.

*Proof*. For any variables *e* and *e’*, there is a permutation over vertices to map *e* to *e’*. (More precisely, and , there is a permutation over vertices to map to , and to .) Then

(*h* is invariant to )

(flipping & then acting acting & then flipping )

.

## Aanderaa-Karp-Rosenberg Conjecture

Now we can prove the theorem.

**Theorem 3.1**. for all monotone graph properties *f*.

The proof below is so short that it fits in one page. (You’d appreciate the proof more if you’ve ever ~~tried along the approaches of~~ read the original proofs by graph packing…) The lower bound is obtained by combining two inequalities:

and

#### 3.1 Eq.(1)

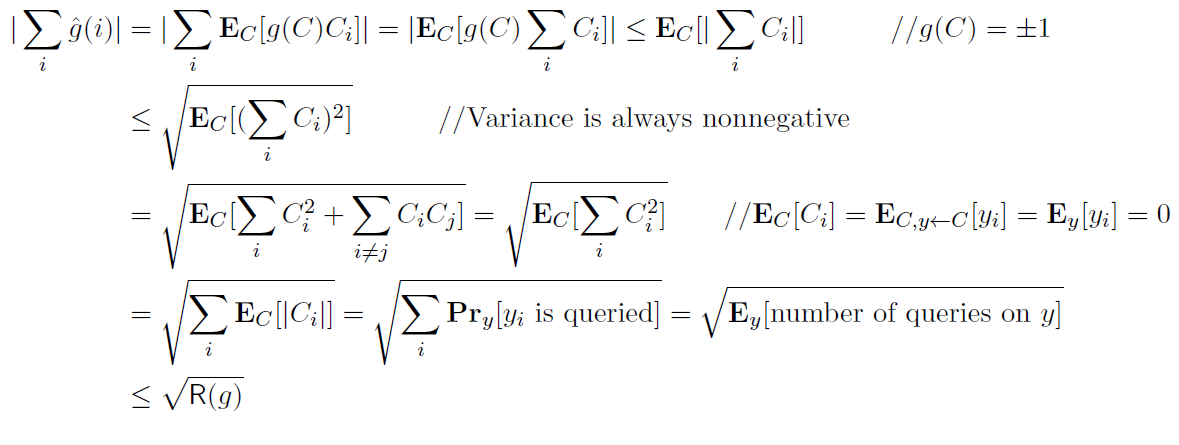
The original proof was given in [OSSS05]. Here we present a simplified proof given in [JZ11].

Take a best deterministic algorithm A with k queries and -distributional error at most , and we’ll construct another algorithm A’ with queries and -distributional error at most . Then repeat this k times and we’ll get an algorithm making no query and has error . But an algorithm without any query can do nothing but guessing the most likely ; the error is , and we then obtained the bound.

A’ is very simple. Suppose A’s first query is , then A’ is the same as A except that A’ doesn’t make this query. Instead, A’ draws as a guess of . With probability , , great: A’ guessed correctly. With probability , , bad. But how bad is it? The newly introduced error probability is at most by definition of influence! Thus we are done. (The above algorithm is randomized. An averaging argument gives a deterministic one.)

#### 3.2 Eq (2)

For simplicity, we prove the special case of ; the general case is similar. By Yao's minimax principle, it's enough to argue that for any fixed deterministic query algorithm, the expected number of queries is at least (equality because of Fact 2.). Note that any decision tree partitions into monochromatic subcubes . A key (and very simple) observation is that the following two random processes give the same distribution of : 1) Draw uniformly at random, and then let C be the cube containing *y*. 2) Draw C with probability , and pick uniformly at random. In the following, the subscripts *y* and C are from their marginal distributions. Note that  , where for any . Now



## References

[CK01] Amit Chakrabarti, Subhash Khot, Improved Lower Bounds on the Randomized Complexity of Graph Properties, In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, pages 285-296, 2001.

[Haj91] Peter Hajnal. **An lower bound on the randomized complexity of graph properties**. *Combinatorica*, 11:131–143, 1991.

[JZ11] Rahul Jain, Shengyu Zhang. **The influence lower bound via query elimination**, *Theory of Computing*, Volume 7, pp. 147-153, 2011.

[Kin88] Valerie King. **Lower bounds on the complexity of graph properties**. In *Proc. 20th Annual ACM Symposium on the Theory of Computing*, pages 468–476, 1988.

[LY94] L. Lovasz, N. Young, **Lecture notes on evasiveness of graph properties**. Technical Report, Princeton University, 1994. Available at <http://www.uni-paderborn.de/fachbereich/AG/agmadh/WWW/english/scripts.html>.

[OSSS05] Ryan O'Donnell, Michael E. Saks, Oded Schramm, Rocco A. Servedio. **Every decision tree has an influential variable**. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pp. 31–39. 2005.

[RV76] R. L. Rivest and J. Vuillemin **On recognizing graph properties from adjacency matrices**, *Theoretical Computer Science* 3(3), 371–384, 1976.

[Yao87] Andrew C. Yao. **Lower bounds to randomized algorithms for graph properties**. In *Proc. 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 393–400, 1987.