CSCI 5170: Computational Complexity **Lecture 4 and 5**
The Chinese University of Hong Kong, Fall 2019

A sublinear-time algorithm is an algorithm that produces an answer before looking at its whole input. One example is polling: To find out who will win an election it is usually not necessary to ask every single voter. A representative sample typically tells who the potential winner will be.

Polling, like many other sublinear-time algorithms, is a randomized procedure. Randomness can make a difference in the power of such algorithms. Quantum computation is a more general type of computation that yields further improvements in some significant cases.

# 1 Randomness and queries

The *(deterministic) query complexity* $D(f)$ of a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ is the smallest possible depth of a decision tree computing $f$. It is the smallest number of bits that must be probed to determine its value. For example, the AND function on $n$ inputs has query complexity $n$, and so does the PARITY function. Many examples you can think of have query complexity $n$. Are there any exceptions?

One silly kind of exception is a function whose value only depends on some strict subset of the $n$ bits. Such functions are called *juntas*. A more interesting example is the addressing function: This is a function $\mathit{Addr}\colon [n] \times \{0,1\}^n \to \{0,1\}$ that takes as inputs an index $i$ and a string $x = x_1 \cdots x_n$ and returns its $i$-th bit $x_i$. This function takes an $n + \log n$-bit long input and its query complexity is $\log n + 1$.

In Homework 1 you argued that the recursive majority $RMAJ_d$ function on $n = 3^d$ inputs has query complexity $n$. In this case choosing the queries in a randomized order can provide considerable savings. Recall that this function has the recursive structure $RMAJ_d(x, y, z) = MAJ(RMAJ_{d-1}(x), RMAJ_{d-1}(y), RMAJ_{d-1}(z))$ (with base case $RMAJ_0(x) = x$). To evaluate $RMAJ_d(x, y, z)$, first recursively evaluate two of the three functions $RMAJ_{d-1}(x)$, $RMAJ_{d-1}(y)$, $RMAJ_{d-1}(z)$ *chosen at random.* If their evaluations match, output the common value. If they don't evaluate the third one and output the majority of the three values.

While this procedure can make as many as $n$ queries in case it made poor random choices, this is unlikely to happen. At least two out of the three of the functions $RMAJ_{d-1}(x)$, $RMAJ_{d-1}(y)$, $RMAJ_{d-1}(z)$ must have matching values. The probability that these two are chosen to be evaluated first is (at least) 1/3. By the total expectation theorem, the *expected* number of queries $Q_d$ made by this algorithm satisfies the recurrence

$$Q_d \leq \tfrac{1}{3}(2Q_{d-1}) + \tfrac{2}{3}(3Q_{d-1}) = \tfrac{8}{3}Q_{d-1}$$

with base case $Q_0 = 1$, which solves to $Q_d \leq (8/3)^d \leq n^{0.893}$. Thus the value of recursive majority on $n$ input bits can be determined after querying $n^{0.893}$ bits of the input on average. This holds true for every possible input in $\{0,1\}^n$; the averaging occurs over the random choices made by the algorithm.

To summarize, we have an example of a function on $n$ bits which requires deterministic query complexity $n$, but admits algorithms of expected randomized query complexity $n^{0.893}$. How large can the gap between these two quantities be?

## 2 Randomized decision trees

When we described the algorithm for recursive majority it was natural to have the algorithm toss a 3-sided die every time it has to choose the evaluation order of its subtrees. To analyze the power of randomness it is more useful to think of the algorithm as tossing all these dies at the beginning and then running a deterministic procedure that depends on their outcomes. Every randomized algorithm can be implemented in this way without affecting its query complexity.

From this perspective, a *randomized decision tree* is a probability distribution over deterministic decision trees. For a given input $x$, the query complexity of a randomized decision tree on input $x$ is then a random variable: Some decision trees might get lucky and output an answer after querying few bits of $x$, while others may need to look at most or all of them.

There are two reasonable definitions of correctness for a randomized algorithm. A Las Vegas algorithm is one that is always correct (with probability 1). The recursive majority algorithm is of the Las Vegas type. A Monte Carlo algorithm is one that is correct most of the time. Polling algorithms are examples of the Monte Carlo type: Assuming that there is some gap between the popularity of the candidates, polling sufficiently many people can predict the outcome of an election to within any given sampling error, but never with 100% confidence.

**Definition 1.** A randomized decision tree $T$ computes $f$ with error $\varepsilon$ if $\Pr[T(x) \neq f(x)] \leq \varepsilon$ for every input $x$. The average randomized decision tree complexity of $T$ is the largest expected number of queries that $T$ makes over all inputs. The randomized decision tree complexity $R_\varepsilon(f)$ of a function $f$ is the smallest possible average complexity among all randomized decision trees that compute $f$ with error $\varepsilon$.

Thus $R_0(f)$ is the measure of the best Las Vegas algorithm for $f$. As for Monte Carlo algortihms, if we don't care about constant factors in query complexity we can fix $\varepsilon$ to a specific constant like 1/3 and moreover assume that the decision trees have bounded depth not only on average but with probability 1.

**Claim 2.** $R_\delta(f) = O((1/\varepsilon^2)\log(1/\delta)R_{1/2-\varepsilon}(f))$.

*Proof.* Repeat the $(1/2-\varepsilon)$-error algorithm $O(1/\varepsilon^2 \log(1/\delta))$ times and output the majority of the answers. By the Chernoff bound for any given input the probability the majority value is incorrect is at most $\delta$. □

**Claim 3.** *There is a randomized decision tree of that computes $f$ with error $\varepsilon + \delta$ and whose query complexity is at most $(1/\delta)R_\varepsilon(f)$ with probability 1.*

*Proof.* Clip the decision tree paths of length more than $(1/\delta)R_\varepsilon(f)$ and output an arbitrary answer. By Markov's inequality the probability that for any fixed input, the path is $1/\delta$ times longer than its expected length is at most $\delta$. So the clipping incurs at most $\delta$ additional error probability. □

These two claims justify the adoption of $R_{1/3}(f)$ as a measure of Monte Carlo decision tree complexity: Up to constant factor, this quantity is preserved by small changes in the model.

## 3 Degree and sensitivity

We already saw several ways to lower bound decision tree depth. Let's see a couple more.

In Lecture 2 we talked about representing Boolean functions as polynomials (over the group $\mathbb{Z}_2$) with $+$ and $\times$ are the XOR and AND operations, respectively. Such functions can also be represented as polynomials over the real numbers. It is more convenient to represent input bits by the values 1 and $-1$ instead of 0 and 1. For the output bits we'll stick to a 0/1 representation (though it doesn't matter much). The PARITY function is then represented by the polynomial

$$PARITY(x_1, \dots, x_n) = \frac{1 - x_1 \cdots x_n}{2}$$

while AND is represented by

$$AND(x_1, \dots, x_n) = \frac{1 - x_1}{2} \cdots \frac{1 - x_n}{2}.$$

Every function has a unique representation as a degree-$n$ multilinear polynomnial. We have already seen it; it is the Fourier expansion. When input bits are represented by the values $-1$ and 1, the Fourier characters $\chi_a(x)$ take the form $\prod_{i \in S} x_i$, where $S$ is the set of 1-bits of $a$. So every function from $\{-1, 1\}^n$ to the real numbers can be uniquely written as

$$f(x) = \sum_{S \subseteq [n]} \hat{f}_S \cdot \prod_{i \in S} x_i$$

where the Fourier coefficients $\hat{f}_S$ are given by $\mathrm{E}\, f(x) \prod_{i \in S} x_i$.

The *degree* of $f$ is the degree of its representing real-valued multilinear polynomial, or alternatively the size of the largest set that appears in its Fourier expansion.

The deterministic query complexity of a function must be at least as large as its degree:

**Claim 4.** *$D(f) \geq \deg(f)$.*

*Proof.* A decision tree can be written as a sum of functions $f_p$ of at most $D(f)$ variables each, where each function $f_p$ is an indicator for a particular path $p$ leading to a 1-leaf being taken. Since each $f_p$ has degree at most $D(f)$ so does $f$. □

Another more local complexity measure of a function is its *sensitivity*. Say bit $i$ flips $f$ at $x$ if $f(x) \neq f(x^i)$, where $x^i$ is the string obtained from $x$ by changing its $i$-th bit. The *sensitivity* $\mathrm{sens}(f)$ is the maximum number of bits that flip $f$ taken among all its inputs $x$. For example, $AND$ has sensitivity $n$ because all bits flip it at $x = 0^n$ ($1^n$ in the $-1/1$ representation).

**Claim 5.** *$D(f) \geq \mathrm{sens}(f)$.*

*Proof.* For every input $x$, the only bits that may flip $f$ at $x$ are those queried by the decision tree, and there are at most $D(f)$ of them. □

There is an analogue of Claim 4 for randomized decision trees. We say $f$ has *$\varepsilon$-approximate degree* at most $d$ if there is a real-valued polynomial $p$ of degree at most $d$ such that $|f(x) - p(x)| \leq \varepsilon$.

**Claim 6.** *If $f$ has a randomized decision tree of depth $d$ and error $\varepsilon$ then it has $\varepsilon$-approximate degree at most $d$. In particular, $R_{1/3}(f) = \Omega(\widetilde{\deg}_{1/3}(f))$.*

*Proof.* By assumption there is a probability distribution over deterministic decision trees $T$ of depth $d$ such that $\Pr[f(x) \neq T(x)] \leq \varepsilon$. Since $f$ and $T$ take 0/1 values, this means $|\mathrm{E}[T(x)] - f(x)| \leq \varepsilon$. By Claim 4 each $T$ has degree at most $d$, so the polynomial $p(x) = \mathrm{E}[T(x)]$ has degree at most $d$ and approximates $f$ with error at most $\varepsilon$. □

By Claims 2, 3, and 6, $R_{1/3}(f) \geq \Omega(\widetilde{\deg}_{1/3}(f))$. So a potential method for arguing that a function has large randomized query complexity is by lower bounding its approximate degree. Unfortunately approximate degree is not always easy to calculate, and the lower bounds it gives are sometimes far from tight. Nevertheless it is important for two reasons. One will be given in the next lecture. For the other one, keep reading.

## 4 Polynomial equivalence of complexity measures

We introduced several complexity measures for Boolean functions: deterministic and randomized query complexity, degree, sensitivity, and approximate degree. We also saw an example of a gap between randomized and deterministic query complexity. It turns out that this gap can never be too large!

Say two measures $M$ and $M'$ are *polynomially equivalent* if there exist constants $c, C > 0$ such that $\Omega(M(f)^c) \leq M'(f) \leq O(M(f)^C)$ for all $f\colon \{0,1\}^n \to \{0,1\}$ and all $n$.

**Theorem 7.** *$D$, $R_0$, $R_{1/3}$,* $\deg$*,* $\widetilde{\deg}$*, and* sens *are all polynomially equivalent.*

Thus sensitivity, which is a very "local" measure and generally easy to bound, at least from below, tells us what the query complexity of a function is up to polynomial equivalence.

To prove Theorem 7 we need to talk about one more measure. The *block sensitivity* of a function is the largest number of blocks the set of variables can be partitioned in so that at some input $x$, flipping all the variables in any given block changes the value of the function. Alternatively, you can think of block sensitivity as the maximum (regular) sensitivity among all functions obtained by potentially identifying different variables with one another:

$$\mathrm{bsens}(f) = \max_{(i_1,\ldots,i_n)} \mathrm{sens}(f(x_{i_1},\ldots,x_{i_n})),$$

where the maximum is taken over all sequences of indices with possible repetitions. In particular, block sensitivity is at least as large as sensitivity.

**Claim 8.** $D(f) \leq \mathrm{bsens}(f) \cdot \deg(f)^2$

**Claim 9** (The sensitivity theorem)**.** $\deg(f) \leq \mathrm{sens}(f)^2$.

**Claim 10.** $\mathrm{sens}(f) \leq \mathrm{bsens}(f) \leq \Omega\big(\widetilde{\deg}_{1/3}(f)^2\big)$.

We prove Claims 8 and 9 today and Claim 6 in the next lecture. Using them we can easily prove Theorem 7:

$$\deg^{1/4} \overset{9}{\leq} \mathrm{sens}^{1/2} \overset{10}{\preceq} \widetilde{\deg}_{1/3} \overset{6}{\preceq} R_{1/3} \leq R_0 \leq D \overset{8}{\leq} \deg^2 \cdot \mathrm{bsens} \overset{10}{\preceq} \deg^2 \cdot \widetilde{\deg}_{1/3}^2 \leq \deg^4.$$

Here $a \preceq b$ stands for $a(f) = O(b(f))$, and the number above each inequality refers to the Claim that it follows from. All the measures of interest are polynomially equivalent to degree, so they are polynomially equivalent to one another.

*Proof of Claim 8.* We prove the claim by induction on the degree of $f$. The base case $\deg(f) = 0$ checks out. Now suppose the claim is true for all functions of degree less than $\deg(f)$. We show it is true for $f$.

Let $M$ be any monomial of $f$ of maximal degree. If $\rho$ restricts the inputs outside $M$ to value 1 then $f|_\rho$ still contains the monomial $M$ (as it cannot cancel out with any of the restricted monomials) so it is not the constant function. Therefore there must exist a subset $B$ of $M$ so that $f$ flips its value at input $x = 1^n$ when the inputs in $B$ are flipped.

Now let $\mathcal{M}$ be a maximal set of monomials of $f$ of maximal degree that do not share any variables. Since each monomial in $\mathcal{M}$ contains a subset that flips its value at $1^n$, the size of $\mathcal{M}$ can be at most $\text{bsens}(f)$. Since each monomial in $\mathcal{M}$ has $\deg(f)$ variables, $\mathcal{M}$ covers at most $\deg(f)\text{bsens}(f)$ variables. Since $\mathcal{M}$ is maximal, after these variables are restricted to any possible set of values $\rho$ the degree of $f|_\rho$ must strictly decrease, while its block sensitivity cannot increase. By inductive hypothesis $D(f|_\rho) \leq \deg(f|_\rho)^2\text{bsens}(f|_\rho) \leq (\deg(f) - 1)^2\text{bsens}(f)$, so $D(f) \leq \deg(f)\text{bsens}(f) + (\deg(f) - 1)^2\text{bsens}(f) \leq \deg(f)^2\text{bsens}(f)$, completing the induction. □

## 5 Proof of the Sensitivity Theorem

For the proof fo Claim 9 it is better to think of $f$ as a function that takes values $1/-1$ instead of $0/1$. This changes neither the sensitivity nor the degree (as the change of representation can be implemented by the linear shift $y \to 1 - 2y$).

It is enough to prove the theorem for functions $f\colon \{-1, 1\}^n \to \{-1, 1\}$ of maximum degree $n$: If we take a maximal degree monomial of $f$ and restrict the values outside to 1 then the sensitivity can only go down while the degree stays the same. In Fourier language, we may assume that the coefficient $\hat{f}_{[n]}$ of the monomial $\prod_{i\in[n]} x_i$ is nonzero.

Next, instead of looking at the function $f\colon \{-1, 1\}^n \to \{-1, 1\}$ of degree $n$ we switch to the function $g(x) = f(x) \cdot \prod_{i=1}^n x_i$. This transformation has the effect at flipping the value of $f$ at all inputs that contain an odd number of $-1$'s. Since $x_i^2 = 1$ the coefficient of each monomial of $g$ equals to the coefficient of the complementary monomial of $f$:

$$g(x) = \sum_S \hat{f}_S \prod_{i\notin S} x_i$$

In particular, the average value $\mathrm{E}[g(x)]$ of $g$ equals the coefficient $\hat{f}_{[n]}$, so it is nonzero.

Finally, it will be convenient to think of the set $\{-1, 1\}^n$ as vertices of the *hypercube graph* $H_n$, the pairs that differ in exactly one input coordinate as edges of this graph, and the function $g$ as a coloring of the vertices of this graph with the colors $-1$ and 1. Then the assumption $\mathrm{E}[g(x)] \neq 0$ means that one of the colors covers strictly more than half the vertices. Let's call it the *dominant color.* Let $G$ be the subgraph of $H_n$ the hypercube induced by the vertices of the dominant color. Then the degree of any vertex in this graph can be at most $\text{sens}(f)$: At most $\text{sens}(f)$ neighbors of $x$ flip the value in $f$, so at most this many vertices can have the same color in $G$.

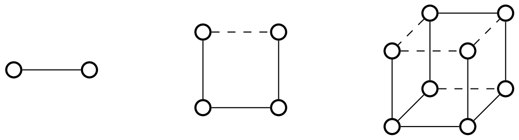
These changes in perspective reduce Claim 9 to the following statement about subgraphs of the hypercube:

**Lemma 11.** *Any induced subgraph of $H_n$ with strictly more than $2^{n-1}$ vertices has a vertex of degree at least $\sqrt{n}$.*

To prove Lemma 11 we need to talk about weighted graphs. A *bounded weighting* of a graph $G$ is an assignment of weights between $-1$ and 1 to the edges of $G$ (and zero weights to the non-edges). Recall that the adjacency matrix of a weighted graph is a symmetric matrix whose $(i, j)$-th entry is the weight of edge $(i, j)$. The eigenvalues of a (weighted) graph are the eigenvalues of its adjacency matrix. (Since the matrix is symmetric all eigenvalues are real and the corresponding eigenvectors are real and orthonormal.)

**Claim 12.** *The hypercube $H_n$ has a bounded weighting all of whose eigenvalues are $-\sqrt{n}$ or $\sqrt{n}$.*

All of the weights in fact have value 1 or $-1$. This is what the desired weighting looks like in the first three hypercubes. Solid and dashed lines indicate weight 1 and $-1$, respectively:



Each weighting is built recursively by assigning the previous one to the bottom part of the hypercube, its negation to the top part, and weight-1 edges to the matching that connects the two.

*Proof.* If the vertices are ordered lexicographically, the adjacency matrix is defined recursively by the formula

$$W_n = \begin{bmatrix} W_{n-1} & I \\ I & -W_{n-1} \end{bmatrix}$$

with base case $W_0 = 0$. Its square $W_n^2$ satisfies the recursion

$$W_n^2 = \begin{bmatrix} W_{n-1}^2 + I & 0 \\ 0 & W_{n-1}^2 + I \end{bmatrix} = \begin{bmatrix} W_{n-1}^2 & 0 \\ 0 & W_{n-1}^2 \end{bmatrix} + I.$$

Unwinding the recursion gives $W_n^2 = nI$, so all the eigenvalues of $W_n^2$ are equal to $n$. Since $W_n$ has real eigenvalues all of them must equal $-\sqrt{n}$ or $\sqrt{n}$. □

**Claim 13.** *If a (weighted) graph has an eigenvalue of multiplicity $m > 1$, then the subgraph obtained by removing any vertex has the same eigenvalue with multiplicity at least $m-1$.*

*Proof.* Suppose $x$ is an eigenvector of $A$, and assume its $i$-th entry $x_i$ is zero. Then $x$ is also an eigenvector with the same eigenvalue of the matrix obtained by zeroing out the $i$-th row and column of $A$. If $A$ is an adjacency matrix, then $x$ with the $i$-th entry removed is an eigenvector with the same eigenvalue of the graph obtained by removing the $i$-th vertex.

If the graph has an eigenvalue of multiplicity $m$ then the corresponding space of eigenvectors is $m$-dimensional. It must contain an $(m-1)$-dimensional subspace that is zero at the position of the removed vertex. After removing the vertex and the corresponding entry in the eigenvectors this $(m-1)$-dimensional space still remains a space of eigenvectors with the same eigenvalue, so the eigenvalue must have multiplicity at least $m-1$. □

**Claim 14.** *The maximum degree of a graph is at least as large as the largest eigenvalue of any of its bounded weightings.*

*Proof.* Let $d$ be the maximum degree and $x, \lambda$ be an eigenvector-value pair of the weighted adjacency matrix $A$. Since $Ax = \lambda x$, for every entry $x_i$ of $x$, $\lambda x_i$ must equal some bounded linear combination of at most $d$ other $x_j$'s. If $x_i$ is the largest entry of $x$ then the linear combination can reach its target only if the number of summands $d$ is at least as large as $|\lambda|$. □

*Proof of Lemma 11.* Take the bounded weighting of $H_n$ from the first claim. At least one of the eigenvalues $-\sqrt{n}$ or $\sqrt{n}$ has multiplicity $2^{n-1}$. After removing the fewer than $2^{n-1}$ vertices outside the induced subgraph the second claim says that at least one of these eigenvalues survives. By the third claim the maximum degree must be at least as large as it. □

## 6 Query complexity of partial functions

A *partial function* is a function that is only defined over a subset of all inputs. In the case of 0/1-valued functions over the Boolean cube (i.e. decision problems), a partial function is a function $f\colon \Pi \to \{0,1\}$ where $\Pi$ is a subset of $\{0,1\}^n$ called *the promise.* When evaluating the function we only care about being correct on inputs in the promise.

The deterministic and randomized query complexities of partial functions can be very different and Theorem 7 does not hold for them. For example, consider the set $\Pi$ consisting only of strings that have at most $n/3$ zeros or at least $n/3$ ones. The approximate majority partial function is majority on strings satisfying the promise:

$$APXMAJ(x) = \begin{cases} 1, & \text{if } x \text{ has at least } 2n/3 \text{ ones,} \\ 0, & \text{if } x \text{ has at most } n/3 \text{ ones.} \end{cases}$$

Approximate majority requires $n/3$ deterministic queries, but its Monte Carlo randomized query complexity is constant. If you sample say 21 random bits of $x$ and take their majority, then it is likely to be correct on every input that satisfies the promise (it is like sampling a coin that is promised to be heavily biased). So the gap between deterministic and randomized query complexity can be very large for partial functions.

Nevertheless, although Theorem 7 fails, several of the claims we proved along the way hold for partial functions as well. These include all claims from Sections 2 and 3. It also includes the important Claim 10 which we will prove next time. An illustrative example in this context is the approximate OR function:

$$APXOR(x) = \begin{cases} 1, & \text{if } x \text{ has exactly one 1,} \\ 0, & \text{if } x \text{ is the all-zero string.} \end{cases}$$

This partial function has sensitivity and therefore deterministic query complexity $n$. What about its randomized query complexity? Intuitively, a randomized algorithm still needs to make a lot of queries to locate the lonely 1. Since the 1 is in a random position, an algorithm that makes $q$ queries can spot the 1 with probability at most $q/n$, so it cannot distinguish between the two types of inputs with constant advantage unless $q$ is linear in $n$.

A pair of distributions $X_0$, $X_1$ is *$\delta$-indistinguishable* by algorithm $A$ if $\Pr[A(X_1) = 1] - \Pr[A(X_0) = 1] \leq \delta$.

**Lemma 15.** *Suppose there exists a pair of distributions $X_0$, $X_1$ supported on $f^{-1}(0)$ and $f^{-1}(1)$ that are $\delta$-indistinguishable by deterministic decision trees of depth at most $d$. Then no randomized decision tree of depth at most $d$ can compute $f$ with error smaller than $(1-\delta)/2$.*

This is essentially an equivalence: If the conclusion holds then there exists $X_0, X_1$ as in the assumption that are $2\delta$-indistinguishable. So Lemma 15 provides a universal method for lower bounding randomized query complexity.

*Proof.* Let $X$ take value $X_0$ with probability $1/2$ and $X_1$ with probability $1/2$. Viewing $R$ as a distribution over deterministic decision trees $T$, we can write

$$\begin{aligned}
\min_{x\in\Pi} \Pr[R(x) = f(x)] &\leq \mathrm{E}_X \Pr_T[T(X) = f(X)] \\
&= \mathrm{E}_T \Pr_X[T(X) = f(X)] \qquad \text{by linearity of expectation} \\
&= \mathrm{E}_T\left[\tfrac{1}{2}\Pr[T(X_1) = f(X_1)] + \tfrac{1}{2}\Pr[T(X_0) = f(X_0)]\right] \\
&= \mathrm{E}_T\left[\tfrac{1}{2}\Pr[T(X_1) = 1] + \tfrac{1}{2}\Pr[T(X_0) = 0]\right] \\
&= \mathrm{E}_T\left[\tfrac{1}{2}\Pr[T(X_1) = 1] + \tfrac{1}{2}(1 - \Pr[T(X_0) = 1])\right] \\
&\leq \mathrm{E}_T\left[\tfrac{1}{2} + \tfrac{1}{2}\delta\right] \\
&= \frac{1+\delta}{2}. \qquad \square
\end{aligned}$$

For the approximate OR function, the distributions of interest $X_0$ and $X_1$ are uniform over the 0 and 1 inputs of $f$, respectively. In that case get that $q/n \leq 1 - 2\varepsilon$ so the best advantage of a $q$-query randomized algorithm is at most $1/2 + q/2n$.

What is the degree of the approximate OR function? This is a trick question as there are two possible definitions of degree that are sensible for partial functions. The first answer is that the degree is 1 because $APXOR$ matches the value of the linear function $x_1 + \cdots + x_n$ on all points of interest.[1] However this function cannot represent the computation of any decision tree because it outputs non-binary values on inputs that are *outside* the promise $\Pi$. In contrast decision trees always output 0 or 1, even on inputs outside the promise.

This motivates an alternative definition which requires not only that the polynomial represents the function on inputs satisfying the promise, but also that it outputs 0 or 1 on inputs outside the promise.

**Definition 16.** The ($\varepsilon$-approximate) degree of a partial function $f\colon \Pi \to \{0,1\}$ is the best possible ($\varepsilon$-approximate) degree among all total functions from $\{0,1\}^n \to \{0,1\}$ that extend $f$.

In the next lecture we will show that the 1/3-approximate degree of both $OR$ and $APXOR$ are $\Theta(\sqrt{n})$. As their randomized query complexity is $\Omega(n)$, Claim 6 is not very tight in this case. In contrast, the approximate degree of the $MAJORITY$ function on $n$ bits is $\Omega(n)$, while the approximate degree of $APXMAJ$ is not known.

## References

The systematic study of query complexity started with a paper by Nisan and Szegedy from 1994 called *On the degree of Boolean functions as real polynomials.* Virtually all the results here are from that paper with the exception of the Sensitivity Theorem (Claim 9). Nisan and Szegedy conjectured that sensitivity was polynomially related to the other measures but the proof was only given by Huang a few months ago! The graph-theoretic perspective was suggested by Gotsman and Linial. A good pre-Sensitivity Theorem source is this survey of Buhrman and de Wolf.

There is a rich algorithmic field called *property testing* that studies the (randomized) query complexity of combinatorial promise problems that are defined by their proximity to a given property. See Section 10.1.2 of Goldreich's textbook for an introduction.

[1] For $-1/1$ valued inputs it is $n/2 - (x_1 + \cdots + x_n)/2$.