**CS225: Pseudorandomness** Prof. Salil Vadhan

# Lecture 3: Sampling and Approximation Problems

*February 8, 2007*

Based on scribe notes by David Troiano, Grant Schoenebeck, and Brian Greenberg.

## 1 Sampling Problem

The power of randomization is well-known to statisticians. If we want to estimate the mean of some quantity over a large population, we can do so very efficiently by taking the average over a small random sample.

Formally, here is the computational problem we are interested in solving:

Sampling (aka $\varepsilon$-Approx Oracle Average): Given oracle access to a function $f : \{0,1\}^m \to [0,1]$, estimate $\mu(f) \stackrel{\text{def}}{=} \mathrm{E}[f(U_m)]$ to within an additive error of $\varepsilon$. That is, output an answer in the interval $[\mu - \varepsilon, \mu + \varepsilon]$.

And here is the algorithm:

**Algorithm for $\varepsilon$-Approx Oracle Average:** For an appropriate choice of $t$, choose $x_1, \ldots, x_t \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}^m$, and output $(\sum_i f(x_i))/t$.

By the Chernoff Bound from last lecture, we only need to take $t = O(\log(1/\delta)/\varepsilon^2)$ samples to have additive error at most $\varepsilon$ with probability at least $1 - \delta$. Note that for constant $\varepsilon$ and $\delta$, the sample size is independent of the size of the population ($2^m$), and we have running time $\mathrm{poly}(m)$ even for $\varepsilon = 1/\mathrm{poly}(m)$ and $\delta = 2^{-\mathrm{poly}(m)}$.

For this problem, we can *prove* that no deterministic algorithm can be nearly as efficient.

**Proposition 1** *Any deterministic algorithm solving* $(1/4)$-Approx Oracle Average *must make at least* $2^n/2$ *queries to its oracle.*

**Proof:** Suppose we have a deterministic algorithm $A$ that makes fewer than $2^n/2$ queries. Let $Q$ be the set of queries made by $A$ when all of its queries are answered by 0. Now define two functions

$$\begin{aligned} f_0(x) &= 0 \qquad \forall x \\ f_1(x) &= \begin{cases} 0 & x \in Q \\ 1 & x \notin Q \end{cases} \end{aligned}$$

Then $A$ gives the same answer on both $f_0$ and $f_1$ (since all the oracle queries return 0 in both cases), but $\mu(f_0) = 0$ and $\mu(f_1) > 1/2$, so the answer must have error greater than $1/4$ for at least one of the functions. ■

Thus, randomization provides an exponential savings for approximating the average of a function on a large domain. However, this does not show that $\mathbf{BPP} \neq \mathbf{P}$. There are two reasons for this:

1. $\varepsilon$-Approx Oracle Average is not a decision problem, and indeed it is not clear how to define languages that capture the complexity of approximation problems. However, below we will see how a slightly more general notion of decision problem does allow us to capture approximation problems such as this one.

2. More fundamentally, it does not involve the standard model of input as used in the definitions of **P** and **BPP**. Rather than the input being a string that is explicitly given to the algorithm (where we measure complexity in terms of the length of the string), the input is an exponential-sized oracle to which the algorithm is given random access. Even though this is not the classical notion of input, it is an interesting one that has received a lot of attention in recent years, because it allows for algorithms whose running time is sublinear (or even polylogarithmic) in the actual size of the input (e.g. $2^m$ in the example here). As in the example here, typically such algorithms require randomization and provide approximate answers. Ronitt Rubinfeld is teaching an entire course at MIT on sublinear-time algorithms this semester.

## 2 Promise Problems

Now we will try to find a variant of the $\varepsilon$-Approx Oracle Average problem that is closer to the **P** vs. **BPP** question. First, to obtain the standard notion of input, we consider functions that are presented in a concise form, as Boolean circuits $C : \{0,1\}^m \to \{0,1\}$ (analogous to the algebraic circuits defined last lecture, but now the inputs take on Boolean values and the computation gates are $\wedge$, $\vee$, and $\neg$).

Next, we need a more general notion of decision problem than languages: A *promise problem* $\Pi$ consists of a pair $(\Pi_Y, \Pi_N)$ of disjoint sets of strings, where $\Pi_Y$ is the set of yes instances and $\Pi_N$ is the set of no instances. The corresponding computational problem is: given a string that is "promised" to be in $\Pi_Y \cup \Pi_N$, decide which is the case. All of the complexity classes we have seen have natural promise-problem analogues, which we denote by **prP**, **prRP**, **prBPP**, etc.

Now we can consider the following problem problem $\varepsilon$-Approx Circuit Average:

$$\begin{aligned} \mathrm{CA}_Y^\varepsilon &= \{(C,p) : \mu(C) > p + \varepsilon\} \\ \mathrm{CA}_N^\varepsilon &= \{(C,p) : \mu(C) \leq p\} \end{aligned}$$

Here $\varepsilon$ can be a constant or a function of the input length. It turns out that this problem completely captures the power of probabilistic polynomial-time algorithms.

**Theorem 2** *For every function $\varepsilon$ such that $1/\mathrm{poly}(n) \leq \varepsilon(n) \leq 1 - 1/2^{n^{o(1)}}$, $\varepsilon$-Approx Circuit Average is* **prBPP**-*complete. That is, it is in* **prBPP** *and every promise problem in* **prBPP** *reduces to it.*

**Proof Sketch:** Inclusion in **prBPP**:

Hardness for **prBPP**: Given any promise problem $\Pi \in \mathbf{prBPP}$, we have a probabilistic polynomial-time algorithm $A$ that decides $\Pi$ with 2-sided error at most $2^{-n}$ on inputs of length $n$. We can view the output of $A(x;r)$ as a function of its input $x$ and its coin tosses $r$. Note that if $x$ is of length $n$, then we may assume that $r$ is of length at most $\mathrm{poly}(n)$ without loss of generality (why?). For any $n$, there is a $\mathrm{poly}(n)$-sized circuit $C(x;r)$ that simulates the computation of $A$ for inputs $x$ of length $n$, and moreover $C$ can be constructed in time $\mathrm{poly}(n)$. (See any text on complexity theory for a proof.) Let $C_x(r)$ be the circuit $C$ with $x$ hardwired in. Then the map $x \mapsto (C_x, 1/2^n)$ is a polynomial-time reduction from $\Pi$ to $\varepsilon$-Approx Circuit Average. Indeed, if $x \in \Pi_N$, then $A$ accepts with probability at most $1/2^n$, so $\mu(C_x) \leq 1/2^n$. And if $x \in \Pi_Y$, then $\mu(C_x) \geq 1 - 2^{-n} > 2^n + (1 - 2^{-(n')^{o(1)}})$, where $n' = |(C_x, 1/2^n)| = \mathrm{poly}(n)$. □

**Corollary 3** *$\varepsilon$-Approx Circuit Average is in* **prP** $\Leftrightarrow$ **prBPP** $=$ **prP** $\Rightarrow$ **BPP** $=$ **P**.

The proof of Proposition 1 does not extend to $\varepsilon$-Approx Circuit Average. Indeed, it's not even clear how to define the notion of 'query' for an algorithm that is given a circuit $C$ explicitly; it can do arbitrary computations that involve the internal structure of the circuit. Moreover, even if we restrict attention to algorithms that only use the input circuit $C$ as if it were an oracle (other than computing the input length $|(C,p)|$ to know how long it can run), there is no guarantee that the function $f_1$ constructed in the proof of Proposition 1 has a small circuit.

## 3 Approximate Counting to within Relative Error

Note that $\varepsilon$-Approx Circuit Average can be viewed as the problem of approximately counting the number of satisfying assignments of a circuit $C$ to within additive error $\varepsilon \cdot 2^m$, and a solution to this problem may give useless information for circuits that don't have very many satisfying assignments (e.g. circuits with fewer than $2^{m/2}$ satisfying assignments). Thus people typically study approximate counting to within *relative error*. For example, given a circuit $C$, output a number that is within a $(1+\varepsilon)$ factor of its number of satisfying assignments, $\#C$. Or the essentially equivalent decision problem $\varepsilon$-Approx #CSAT:

$$\begin{aligned}\mathrm{CSAT}^{\varepsilon}_Y &= \{(C,N) : \#C > (1+\varepsilon)\cdot N\}\\ \mathrm{CSAT}^{\varepsilon}_N &= \{(C,N) : \#C \leq N\}\end{aligned}$$

Unfortunately, this problem is **NP**-hard for general circuits (why?), so we do not expect a **prBPP** algorithm. However, there is a very pretty randomized algorithm if we restrict to formulas in *disjunctive normal form* (DNF) (i.e. an OR of clauses, where each clause is an AND of variables or their negations). Letting $\varepsilon$-Approx #DNF be the restriction of $\varepsilon$-Approx #CSAT to DNF formulas, we have:

**Theorem 4** *For every $\varepsilon(n) \geq 1/\mathrm{poly}(n)$, $\varepsilon$-Approx #DNF is in* **prBPP**.

**Proof:** It suffices to give a probabilistic polynomial-time algorithm that estimates the number of satisfying assignments to within a $1 \pm \varepsilon$ factor. Let $\varphi(x_1, \ldots, x_m)$ be the input DNF formula.

A first approach would be to apply random sampling as we have used above: Pick $t$ random assignments uniformly from $\{0,1\}^m$ and evaluate $\varphi$ on each. If $k$ of the assignments satisfy $\varphi$, output $(k/t) \cdot 2^m$. However, if $\#\varphi$ is small (e.g. $2^{m/2}$), random sampling will be unlikely to hit any satisfying assignments, and our estimate will be 0

The idea to get around this difficulty is to embed the set of satisfying assignments, $A$, in a smaller set $B$ so that sampling can be useful. Specifically, we will define sets $A'$ and $B$ satisfying the following properties:

1. $|A'| = |A|$
2. $A' \subseteq B$
3. $|A'| \geq |B|/\text{poly}(n)$, where $n = |\varphi|$.
4. We can decide membership in $A'$.
5. $|B|$ computable in polynomial time.
6. We can sample uniformly at random from $B$.

Letting $\ell$ be the number of clauses, we define $A'$ and $B$ as follows:

$$\begin{aligned} B &= \Big\{(i, \alpha) \in [\ell] \times \{0,1\}^m : \alpha \text{ satisfies the } i^{\text{th}} \text{ clause}\Big\} \\ A' &= \Big\{(i, \alpha) \in B : \alpha \text{ does not satisfy any clauses before the } i^{\text{th}} \text{ clause}\Big\} \end{aligned}$$

Now we verify the desired properties:

1. Clearly $|A| = |A'|$ since $A'$ only contains pairs $(i, \alpha)$ such that the first satisfying clause in $\alpha$ is the $i^{\text{th}}$ one.
2. Also, the size of $A'$ and $B$ can differ by at most a factor of $\ell$ by construction since for $A'$ we only look at the first satisfying clause and there can only be $m-1$ more elements in $B$ per assignment $\alpha$.
3. It is easy to decide membership in $A'$ in linear time.
4. $|B| = \sum_{i=1}^{\ell} 2^{m-m_i}$, where $m_i$ is the number of literals in clause $i$.
5. We can randomly sample from $B$ as follows. First pick a clause with probability proportional to the number satisfying assignments it has ($2^{m-m_i}$). Then, fixing those variables in the clause (e.g. if $x_j$ is in the clause, set $x_j = 1$, and if $\neg x_j$ is in the clause, set $x_j = 0$), assign the rest of the variables uniformly at random.

Putting this together, we deduce the following algorithm:

**Algorithm for $\varepsilon$-Approx #DNF:** Generate a random sample of $t = O(1/(\varepsilon/\ell)^2)$ points in $B$. Let $\hat{\mu}$ be the fraction of sample points that land in $A'$, and output $\hat{\mu} \cdot |B|$.

By the Chernoff bound, we have $\hat{\mu} \in [|A'|/|B| \pm \varepsilon/\ell]$ with high probability (where we write $[\alpha \pm \beta]$ to denote the interval $[\alpha - \beta, \alpha + \beta]$). Thus, with high probability the output of the algorithm satisfies:

$$\hat{\mu} \cdot |B| \in [|A'| \pm \varepsilon|B|/\ell] \subseteq [|A| \pm \varepsilon|A|].$$

■

There is no deterministic polynomial-time algorithms known for this problem. However, near the end of the course, we will give a quasipolynomial-time derandomization of the above algorithm (i.e. one in time $2^{\mathrm{polylog}(n)}$).

### 3.1 Finding Large Cuts in a Graph

**Definition 5** *For a graph $G = (V, E)$ and $S \subseteq V$, define* $\mathrm{cut}(S) = |\{(u, v) : u \in S, v \notin S\}|$.

We examine the problem of Max Cut: given $G$, find the largest cut. Solving this problem optimally is **NP**-hard (in contrast to Min Cut, which is in **P**). However, there is a simple randomized algorithm that finds a cut of expected size at least $|E|/2$ (which is 1/2-approximation to the optimal):

**Large Cut Algorithm:** Output a random subset $S \subseteq V$. That is, place each vertex $v$ in $S$ independently with probability 1/2.

To analyze this algorithm, consider any edge $e = (u, v)$. Then the probability that $e$ crosses the cut is 1/2. By linearity of expectations, we have:

$$\mathrm{E}[|\mathrm{cut}(S)|] = \sum_{e \in E} \Pr[e \text{ is cut}] = |E|/2.$$

This also serves as a proof, via the probabilistic method, that every graph (without self-loops) has a cut of size at least $|E|/2$.

Later we will see how to derandomize this algorithm. We note that there is a much more sophisticated randomized algorithm that finds a cut whose expected size is within a factor of .878 of the largest cut in the graph (and this algorithm can also be derandomized).