15-855: Intensive Intro to Complexity Theory
Spring 2009

Approximate Counting/Sampling and the BCGKT Theorem

1 Approximate Counting

We begin this lecture by finishing the proof other Approximate Counting Theorem. Let's recall the statement and all the pieces we need for the proof.

Theorem 1.1. *There is a polynomial-time randomized algorithm* using an oracle for SAT *with the following behavior: Given a Boolean formula φ, with high probability the algorithm outputs a number m such that*

$$m/4 \leq \#\varphi \leq 4m.$$

The factor 4 can also be improved to $1 + 1/\text{poly}(n)$.

When $\#\varphi$ is small we observed that things are easier:

Lemma 1.2. *For any constant $c \in \mathbb{N}$, we can decide in P^{NP} whether or not $\#\varphi = c$.*

We also reduced to the problem of deciding yes/no whether $\#\varphi$ is near/far 2^k:

Definition 1.3. *Say that a number K is "near m" if $K \in [m/2, 2m]$ and that it's "far from m" if $K \notin [m/4, 4m]$.*

By solving the really small cases in P^{NP} and doing binary search otherwise, we reduced the Approximate Counting Theorem to the following:

Problem: Given are φ and $k > 10$. Distinguish in $\mathsf{BPP}^{\mathsf{NP}}$ whether $\#\varphi$ is near 2^k or is far from 2^k.

As in the Valiant-Vazirani Theorem the only tool we need in the proof is easy-to-compute pairwise-independent hash families. Here is an equivalent definition to the one from last time:

Definition 1.4. *Let $\mathcal{H}$ be a class of functions $X \to Y$. We say that $\mathcal{H}$ is a* pairwise-independent hash family *if:*

- $\mathbf{Pr}_{h \in \mathcal{H}}[f(x) = y] = 1/|Y|$ *for all $x \in X$, $y \in Y$.*
- *The events $h(x) = y$ and $h(x') = y'$ are independent for all $x \neq x'$ and y, y'.*

We also observed that if $\mathcal{H}$ is the set of all functions $h : \{0,1\}^n \to \{0,1\}^k$ of the form

$$h(x) = (r_1 \cdot x, \ldots, r_k \cdot x) \oplus b,$$

where $r_1, \ldots, r_k \in \{0,1\}^n$, $b \in \{0,1\}^k$, then it's easy to choose a random h and express "$h(x) = y$" with an $O(kn)$-size circuit.

1.1 The solution: pairwise-independent hashing plus Chebyshev's Inequality

To solve the "Problem" above, we use the same basic trick as in the Valiant-Vazirani Theorem. But instead of hashing into a space of size right around 2^k, let's hash into a space that's about 500 times smaller. Specifically, take the hash function's range to be

$$Y := \{0,1\}^{k-9},$$

so that $|Y| = 2^k/512$. For a given $x \in \{0,1\}^n$, let A_x denote the event that $h(x) = \overline{0}$ and let I_x denote the 0-1 indicator random variable for A_x. Define also the random variable

$$C = \sum_{x \in S} I_x = \#\{x \in S : h(x) = \overline{0}\}.$$

Since $h(x)$ is uniformly distributed on Y, we have $\mathbf{Pr}[A_x] = 1/|Y| = 512/2^k$. Hence by linearity of expectation,

$$\mu := \mathop{\mathbf{E}}_{h}[C] = 512 \cdot \frac{|S|}{2^k}.$$

Key Observation 1: If $|S|$ is "near 2^k", then μ is "near 512" — i.e., in the range $[256, 1024]$. On the other hand, if $|S|$ is "far from 2^k", then μ "far from 512" — i.e., *outside* the range $[128, 2048]$.

Key Observation 2: Chebyshev's Inequality says that

$$\mathbf{Pr}\left[|C - \mu| \geq 2\text{stddev}[C]\right] \leq 1/2^2 = 1/4.$$

In other words, there's at least a 3/4 chance that C is within 2 standard deviations of its mean μ.

Key Observation 3: The fact that h is chosen from a "pairwise-independent hash family" implies that the the indicator random variables I_x are pairwise independent: for $x \neq x'$ we have

$$\mathbf{E}[I_x I_{x'}] = \mathbf{Pr}[h(x) = \overline{0} \wedge h(x') = \overline{0}]\mathbf{Pr}[h(x) = \overline{0}]\mathbf{Pr}[h(x') = \overline{0}] = \mathbf{E}[I_x]\mathbf{E}[I_{x'}].$$

Exercise 1.5. *Let $C = \sum_{x \in S} I_x$ be a sum of pairwise-independent indicator random variables. Then $\mathbf{Var}[C] \leq \mathbf{E}[C]$.*

Hint: Expand out $\mathbf{E}[(\sum_{x \in S} I_x)^2]$.

Combining Observations 2, 3, and the exercise, we have that

$$\mathbf{Pr}[\mu - 2\sqrt{\mu} \leq C \leq \mu + 2\sqrt{\mu}] \geq 3/4.$$

I leave it to you to run the numbers and check that we may conclude:

$$|S| \text{ near } 2^k \quad \Rightarrow \quad 256 \leq \mu \leq 1024 \quad \Rightarrow \quad \mathbf{Pr}[200 \leq C \leq 1100] \geq 3/4;$$

$$|S| \text{ far from } 2^k \quad \Rightarrow \quad \mu < 128 \text{ or } \mu > 2048 \quad \Rightarrow \quad \mathbf{Pr}[200 \leq C \leq 1100] \leq 1/4.$$

Finally, by Lemma (1.2), we can use the NP oracle to *deterministically* decide if

$$\#(\varphi(x) \wedge h(x) = \overline{0}) \in [200, 1100] \quad \Leftrightarrow \quad C \in [200, 1100].$$

Actually, Lemma (1.2) was only about counting satisfying assignments to *formulas*, but we can also use to it count satisfying assignments to *circuits*, by first running the parsimonious reduction from #CIRCUIT-SAT to #SAT.

Hence we can distinguish whether $|S|$ is near 2^k or far from 2^k with probability at least $3/4$. As usual, this can efficiently be boosted up to $1 - 2^{-n^{10}}$ by repeating $\text{poly}(n)$ times and taking the majority answer.

2 Counting and sampling

Let C be a circuit and let $S(C) = \{x : C(x) = 1\}$, the satisfying assignments for C. We have just seen that we can approximate $|S(C)|$ to within any $1 + 1/\text{poly}(n)$ factor with high probability in $\mathsf{BPP}^{\mathsf{NP}}$.

Related question: Could we also *randomly sample* a string b from $S(C)$, approximately uniformly?

The answer, due to Jerrum, Vazirani, and Vazirani, is "yes" and the proof is quite easy. Let's first drop the "approximately" business and see a reduction from *exactly* counting $|S(C)|$ (assumed nonzero) to to *exactly* uniformly sampling from $S(C)$. The trick is just "downward self-reducibility" again. Let

$$S_0 = \{x' \in \{0,1\}^{n-1} : (0, x') \in S(C)\} = S(C_{x_1 \leftarrow 0}),$$
$$S_1 = \{x' \in \{0,1\}^{n-1} : (0, x') \in S(C)\} = S(C_{x_1 \leftarrow 1}),$$

where $C_{x_1 \leftarrow b}$ denotes $(n-1)$-input circuit gotten by fixing the first input bit of C to bit b. Now pick

$$b_1 = \begin{cases} 0 & \text{with probability } |S_0|/|S(C)|, \\ 1 & \text{with probability } |S_1|/|S(C)|. \end{cases}$$

Great — now b_1 is set with the correct probability. Now we would like to pick b_2 to be the next bit of b with the correct conditional probability. Define

$$S_{b_1,0} = \{x' \in \{0,1\}^{n-2} : (b_1, 0, x') \in S(C)\} = S(C_{x_1 \leftarrow b_1, x_2 \leftarrow 0}),$$
$$S_{b_1,1} = \{x' \in \{0,1\}^{n-2} : (b_1, 1, x') \in S(C)\} = S(C_{x_1 \leftarrow b_1, x_2 \leftarrow 1}).$$

Then pick

$$b_2 = \begin{cases} 0 & \text{with probability } |S_{b_1,0}|/|S_{b_1}|, \\ 1 & \text{with probability } |S_{b_1,1}|/|S_{b_1}|. \end{cases}$$

Etc., picking all the remaining bits $b_3, \ldots, b_n$. Clearly the string b ultimately chosen is satisfying for C, and

$$\mathbf{Pr}[b] = \frac{|S_{b_1}|}{|S(C)|} \cdot \frac{|S_{b_1,b_2}|}{|S_{b_1}|} \cdot \cdots \cdot \frac{|S_{b_1,\ldots,b_n}|}{|S_{b_1,\ldots,b_{n-1}}|} = \frac{|S_{b_1,\ldots,b_n}|}{|S(C)|} = \frac{1}{|S(C)|}, \tag{1}$$

as desired.

Now for the approximately business: If each estimate of an $|S|$ is off by a multiplicative factor of at most $1 + \epsilon$, then we see immediately from (1) that the probability of picking a particular b will be off by a multiplicative factor of at most $(1 + \epsilon)^{2n}$. Since we can make ϵ and arbitrarily

small polynomial in n, we can get the sampling error similarly small. (Also, the probability our estimation procedure fails, $2^{-n^{10}}$, is negligible.) Hence:

Theorem 2.1. *For any constant c, there is a poly-time randomized procedure with access to a* SAT *oracle which, given a satisfiable circuit C, outputs a nearly uniformly random string $x \in S(C)$: each such string is output with probability within a factor $1 + 1/n^c$ of $1/|S(C)|$.*

Remark: There is also a straightforward reduction from approximate sampling to approximate counting.

2.1 Exact counting and sampling?

Here is an oddity. The Approximate Sampling Theorem, Theorem 2.1, has been improved by Bellare, Goldreich, and Petrank: They give a $\mathsf{ZPP}^{\mathsf{NP}}$ which samples *exactly* from the uniform distribution on $S(C)$. On the other hand, it is known that if you could *exactly* count $S(C)$ in $\mathsf{BPP}^{\mathsf{NP}}$ then the Polynomial Time Hierarchy collapses (which we don't believe)! This follows immediately from Toda's Theorem, which we will prove later.

So although Approximate Counting and Approximate Sampling are easily interreducible, Exact Sampling is almost surely easier than Exact Counting. You might think about why the reduction from Approximate Counting to Approximate Sampling fails in the Exact Case.

3 Learning Polynomial Circuits

Let's finish with one more cool example of what we could do with an oracle for SAT: we could learn any unknown function with a poly-size circuit! This is the BCGKT Theorem, proved by Bshouty, Cleve, Gavaldà, Kannan, and Tamon.

Theorem 3.1. *Polynomial-size circuits are learnable in* $\mathsf{ZPP}^{\mathsf{NP}}$.

More precisely, consider the following scenario. "Nature" holds on to $f : \{0,1\}^n \to \{0,1\}$, promised to be computable by some circuit of size at most n^k. An algorithm trying to "learn" f can give a poly-size "hypothesis" circuit $H : \{0,1\}^n \to \{0,1\}$ to Nature and ask, "Does H correctly compute f?" Nature either answers "yes" in which case the learner is done, or answers "no" and provides a *counterexample*: a string $x \in \{0,1\}^n$ such that $f(x) \neq H(x)$. (Since the learner can evaluate $H(x)$, it learns that $f(x) = \overline{H(x)}$; this is the only way it learns values of f.) The BCGKT Theorem is:

Theorem 3.2. *There is an $n^{O(k)}$-time zero-error randomized algorithm with access to a SAT oracle which outputs a circuit H computing f.*

Before we get into the proof, a quick comment on the zero-error aspect: It's easy to see that in this learning scenario we can always achieve zero-error because the algorithm can "check its answer". More precisely, if an algorithm can come up with a correct H with probability $3/4$, it can first ask Nature if H is correct. If not, it simply runs itself again.

3.1 The outline

We will now give the outline of the BCGKT algorithm, which is a version of the "Halving Algorithm" from Learning Theory. At all times the algorithm maintains two sets, called CE and ITR.

The set CE ("CounterExamples") consists of all counterexamples the algorithm has been given so far by Nature. This set starts out empty and is maintained *explicitly* by the algorithm. Note that the algorithm can also remember the value $f(x)$ for each $x \in \text{CE}$.

The set ITR ("InTheRunning") consists of all circuits C of size at most n^k which correctly compute f on all $x \in \text{CE}$; these are the circuits that are still "in the running" as possible computers of f. Obviously ITR will be huge (it starts out as the set of all circuits of size at most n^k), so it is maintained *implicitly*. Note that as the algorithm adds to CE the set ITR shrinks; it is always nonempty though, since the circuit computing f is always in ITR.

Given a current state for ITR, we will also be interested in the function $\text{Maj(ITR)} : \{0,1\}^n \to \{0,1\}$, defined by

$$\text{Maj(ITR)}(x) = \text{Majority}_{C \in \text{ITR}} C(x).$$

Here now is the outline of the BCGKT Algorithm, with a bogus first step:

1. Let H be a circuit of size $O(n^{k+1})$ which exactly computes Maj(ITR). (???)
2. Ask Nature if H computes f. If yes, we're done.
3. If no, add Nature's counterexample x into CE and go back to Step 1.

3.2 The proof, modulo (???)

Obviously, Step 1 makes no sense — but other than that, we claim this is a correct, efficient (and deterministic, oracle-free) algorithm. To see this, suppose that after running for some time we produce $H = \text{Maj(ITR)}$ and Nature informs us that $H \not\equiv f$. So we get a counterexample x with

$$H(x) = \text{Maj(ITR)}(x) \neq f(x).$$

This means that at least $1/2$ of all circuits in ITR compute f wrongly on x. By definition, all such circuits are then dropped "out of the running". In other words, whenever we get a counterexample the set ITR shrinks by a factor of $1/2$.

To start with, ITR consisted of all circuits of size at most n^k. Even by a very rough count, there are at most $2^{O(s^2)}$ circuits of size at most s. So the set ITR can shrink by a factor of $1/2$ at most $O(\log 2^{O(n^{2k})}) = O(n^{2k})$ many times. Since the algorithm does $n^{O(k)}$ work in each step (writing down/computing H), the algorithm runs deterministically and oracle-free for time at most $n^{O(k)}$ before it finds some H computing f.

3.3 How to do (???)

Obviously, though, we need to implement (???) or something like it. As-is we can't, since there's no reason to believe there is a smallish circuit computing Maj(ITR). The idea is that the proof didn't *really* need the fact that H computes Maj(ITR) per se. What it *really* needed is this:

$$H(x) \neq f(x) \quad \Rightarrow \quad \text{ITR shrinks by a constant factor when } x \text{ is added to CE.} \tag{2}$$

Claim 3.3. *Suppose H is any circuit such that $H(y) = \text{Maj(ITR)}(y)$ for all y where there a "supermajority"; i.e., at least $2/3$ of the circuits C in* ITR *have the same value $C(y)$. Then* (2) *still holds.*

Proof. Suppose x is a string with $H(x) \neq f(x)$. Then

$$\begin{aligned} &\text{supermajority of } C \in \text{ITR have } C(x) = f(x) \\ &\qquad \Rightarrow \text{supermajority of } C \in \text{ITR have } C(x) \neq H(x) \qquad \text{— contradiction.} \end{aligned}$$

Hence at least $1/3$ of all $C \in$ ITR must compute $f(x)$ wrongly, and hence |ITR| shrinks by a factor of $2/3$ when x is added to CE. □

So it suffices to get a circuit H which agrees with the "supermajorities" of ITR.

Claim 3.4. *Suppose we draw circuits $C_1, \ldots, C_m$ uniformly at random (with replacement) from* ITR *and let H be a circuit computing Maj$(C_1(x), \ldots, C_m(x))$. If $m = O(n)$ then with high probability, H agree with all supermajorities. Further, we can make H a size $O(mn^k) = O(n^{k+1})$ circuit.*

Proof. The claim about H's size is clear (Majority has linear-sized circuits), so it suffices to prove the claim about supermajorities. This is also easy. Let x be any string where the circuits in ITR have a supermajority. The expected fraction of circuits C_i agreeing with this supermajority on x is at least $2/3$, by definition. If $m \gg n$, the empirical fraction will be very close to the expected fraction. In particular, it's easily shown that the probability that fewer than $1/2$ of the circuits agree with the supermajority on x (and hence $H(x)$ disagrees) is $\ll 2^{-n}$. We can now take a union-bound over all 2^n strings. □

Hence the line (???) can actually be executed with high probability, well enough so that (2) holds, so long as we can sample circuits uniformly from ITR. In fact, it's easy to see that the argument still works if we only sample *approximately* uniformly — the probability of drawing a circuit from the supermajority might go down from $2/3$ to, say, .66, but that doesn't affect the argument in any essential way.

But given the set CE (and the associated values $f(x)$ for each $x \in$ CE), the algorithm can write down a poly-size circuit D, whose *inputs* are circuits of C of size at most n^k, and which is defined by $D(C) = 1$ iff $C \in$ ITR. This is easy, since checking $C \in$ ITR just involves checking if $C(x) = f(x)$ for each of the (polynomially many) $x \in$ CE. Hence we can sample circuits in ITR approximately uniformly by applying the Approximate Sampling Theorem to D! This completes the proof.