6.842 Randomness and Computation February 16, 2022

Lecture 6

Lecturer: Ronitt Rubinfeld *Scribe: David Cui, Zixuan Xu*

In this lecture, we look at the equivalence between approximate counting and almost uniform generation for downward self-reducible problems. We shall do this by giving a way to approximately count the number of satisfying assignments for a DNF formula (also known as the problem #DNF) by using the scheme we gave last lecture for uniformly generating satisfying assignments to a DNF formula. We then show that the relationship between approximate counting and uniform generation goes in both directions. After this we give some basic definitions in randomized complexity and introduce a naive derandomization algorithm. First, we recall some definitions.

Definition 1. *Let $L \subset \{0,1\}^*$ be a language and $f : L \to \mathbb{R}_{\geq 0}$ represent a counting problem on L. A **fully polynomial randomized approximation scheme (FPRAS) for** f is a probabilistic algorithm $\mathcal{A}$ such that when given input $x \in L$ and error parameter $\varepsilon > 0$ satisfies*

$$\Pr\left[\frac{f(x)}{1+\varepsilon} \leq \mathcal{A}(x,\varepsilon) \leq (1+\varepsilon)f(x)\right] \geq \frac{3}{4}$$

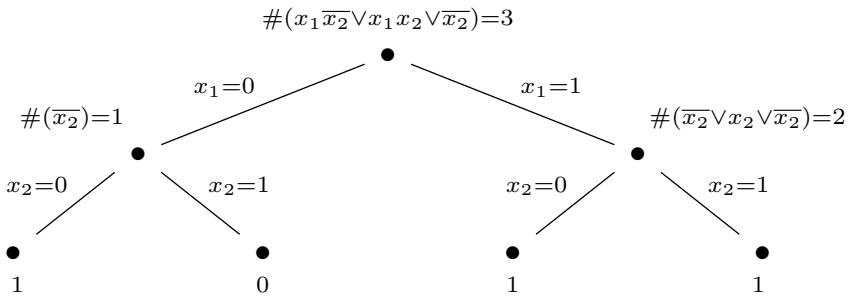
and runs in time polynomial in $|x|, \varepsilon^{-1}$.

Remark The confidence parameter $3/4$ can be amplified to $1-\delta$ at the cost of the algorithm running in time polynomial in $|x|$, ε^{-1}, and $\log \delta^{-1}$ instead.

Definition 2. *A language $L \subset \{0,1\}^*$ is **downward self-reducible** if there exists a polynomial time Turing machine V with oracle access to L such that V^L decides L and V on input x only makes queries to L on strings of length less than $|x|$.*

Roughly speaking this just says that there's a way to solve the problem via recursively solving subproblems. For example, **SAT** is downward self-reducible. We shall now see that #**DNF** is downward self-reducible.

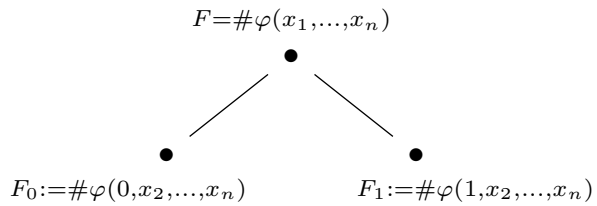
The key insight is to notice that given a DNF instance φ, the number of solutions $\#\varphi$ is given by the sum of the number of solutions of the DNF formulae $\varphi|_{x_1=0}$ and $\varphi|_{x_1=1}$. Hence, if we could solve the problem on smaller instances, then we can use this to build to the original instance. We can explicitly see this through the following example.



We'll call such a tree for a DNF formula φ a downward self-reducible (DSR) tree where the general procedure is to fix the ith variable at height i of the tree.

1 Approximate counting for #DNF

Given a DNF formula φ, consider the first two levels of its DSR tree.



Since **DNF** $\in$ **P**, we don't need to consider unsatisfiable DNF formulae. Namely, we can assume that $F \neq 0$. Now, let $S_b := F_b/F$ be the fraction of satisfying assignments such that $x_1 = b$ for $b \in \{0,1\}$. Then note that if $F_b \neq 0$, $F = F_b/S_b$. Without loss of generality suppose $F_0 \neq 0$.

Now, the main insight is that we have an algorithm for uniform generation for solutions to DNF formulae and hence we can estimate the value S_0! Furthermore, given that we know the value of F_0, we can turn this into an estimation for F (and hence forget about the F_1 subtree). But F_0 is also a DNF instance (with one less variable) so we can use the same trick on F_0 and recurse down until we've fixed all the variables.

1.1 FPRAS for #DNF

We now describe the algorithm. Let $\mathcal{U}$ be the uniform generator for solutions of DNF formulae. Now we define an estimator for S_b.

$\mathcal{E}(\varphi, b, k)$:

1 let $\mathbf{x}^1, \ldots, \mathbf{x}^k$ be k independent samples of $\mathcal{U}(\varphi)$
2 **return** $\frac{1}{k}\sum_{i=1}^{k} \mathbb{1}_{\mathbf{x}_1^i = b}$

The estimator algorithm $\mathcal{E}$ takes in a DNF formula φ and estimates, using k samples, the fraction of solutions of φ that have the first variable set to b. Now we are ready to describe the FPRAS for #**DNF**.

$\mathcal{A}_k(\varphi, \varepsilon)$:

1 **if** $\varphi \equiv 0$ or 1
2 **then return** val(φ)
3 $\widetilde{S}_0 := \mathcal{E}(\varphi, 0, k)$
4 $\widetilde{S}_1 := \mathcal{E}(\varphi, 1, k)$
5 **if** $\widetilde{S}_0 \geq \widetilde{S}_1$
6 **then return** $\mathcal{A}\left(\varphi|_{x_1=0}, \varepsilon\right)/\widetilde{S}_0$
7 **else return** $\mathcal{A}\left(\varphi|_{x_1=1}, \varepsilon\right)/\widetilde{S}_1$

This algorithm is as we described above. We recursively fix the variables of the DNF formula. Notice here we choose to go down the branch with the larger number of satisfying solutions. This is both a convenient way of ensuring the branch we go down actually *has* a satisfying assignment (so we don't have an issue of dividing by zero) but also a way to make sure our errors are not too large as we shall see in the analysis. When all the variables are fixed, we simply return the value of the formula. In our case, this will always be 1.

1.2 Analysis of the FPRAS

We first fix some notation. Let φ be a DNF formula of n variables. $\varphi|_{b_1 b_2,\ldots,b_k}$ for $1 \le k \le n$ will denote the DNF formula φ with $x_1 = b_1, x_2 = b_2, \ldots, x_k = b_k$, where $b_i \in \{0,1\}$. Let $F := \#\varphi$ and $F_{b_1 b_2 \ldots b_k} := \#\varphi|_{b_1 b_2 \ldots b_k}$. Let $S_{b_1 b_2 \ldots b_k} := F_{b_1 b_2 \ldots b_k}/F_{b_1 b_2 \ldots b_{k-1}}$. $\widetilde{S}_{b_1 b_2 \ldots b_k}$ and $\widetilde{F}_{b_1 b_2 \ldots b_k}$ shall denote the approximations of these corresponding values (i.e., what the algorithm outputs).

Theorem 3. *The algorithm $\mathcal{A}_k$ is an FPRAS for #**DNF** with error parameter ε and confidence parameter $1-\delta$ when $k \in O(n^3 \log(1/\delta)/\varepsilon^2)$.*

Proof. A standard application of the Chernoff bound shows that $O(n^2 \log(1/\delta)/\varepsilon^2)$ independent samples of $\mathcal{U}$ is sufficient for establishing

$$\Pr\left[S_{b_1 \ldots b_k} - \frac{\varepsilon}{12n} \le \widetilde{S}_{b_1 \ldots b_k} \le S_{b_1 \ldots b_k} + \frac{\varepsilon}{12n}\right] \ge 1 - \delta$$

for a single instance of $\widetilde{S}_{b_1 \ldots b_k}$. Hence by union-bound we need $O(n^3 \log(1/\delta)/\varepsilon^2)$ samples to establish

$$\Pr\left[S_{b_1 \ldots b_k} - \frac{\varepsilon}{12n} \le \widetilde{S}_{b_1 \ldots b_k} \le S_{b_1 \ldots b_k} + \frac{\varepsilon}{12n} \text{ for all } 1 \le k \le n,\, b_1, \ldots, b_k \in \{0,1\}\right] \ge 1 - \delta.$$

Now since $\widetilde{S}_{b_1 \ldots b_k} + \widetilde{S}_{b_1 \ldots \bar{b}_k} \ge S_{b_1 \ldots b_k} + S_{b_1 \ldots \bar{b}_k} - 2\frac{\varepsilon}{12n} = 1 - 2\frac{\varepsilon}{12n}$, we have $1/2 - \frac{\varepsilon}{12n} \le \widetilde{S}_{b_1 \ldots b_k} \le 1$. This means we can turn the additive error into a multiplicative one as follows. We fix ε to be small enough such that $1/2 - \frac{\varepsilon}{12n} \ge 1/3$ then

$$\widetilde{S}_{b_1 \ldots b_k} \le S_{b_1 \ldots b_k} + \frac{\varepsilon}{12n} = S_{b_1 \ldots b_k}\left(1 + \frac{\varepsilon}{12 S_{b_1 \ldots b_k} n}\right) \le S_{b_1 \ldots b_k}\left(1 + \frac{\varepsilon}{4n}\right),$$

$$\widetilde{S}_{b_1 \ldots b_k} \ge S_{b_1 \ldots b_k} - \frac{\varepsilon}{12n} = S_{b_1 \ldots b_k}\left(1 - \frac{\varepsilon}{12 S_{b_1 \ldots b_k} n}\right) \ge S_{b_1 \ldots b_k}\left(1 - \frac{\varepsilon}{4n}\right).$$

Now finally we estimate the error of $\widetilde{F}$. By construction of the algorithm $\mathcal{A}$ we have

$$\widetilde{F} = \frac{\widetilde{F}_{b_1}}{\widetilde{S}_{b_1}} = \frac{\widetilde{F}_{b_1 b_2}}{\widetilde{S}_{b_1}\widetilde{S}_{b_2}} = \cdots = \frac{1}{\prod_{k=1}^{n} \widetilde{S}_{b_1 \ldots b_k}}.$$

Then using the above estimation,

$$\begin{aligned}
\frac{1}{\prod_{k=1}^{n} \widetilde{S}_{b_1 \ldots b_k}} &\le \frac{1}{\prod_{k=1}^{n} S_{b_1 \ldots b_k}\left(1 - \frac{\varepsilon}{4n}\right)} \\
&\le \frac{(1 + \frac{\varepsilon}{2n})^n}{\prod_{k=1}^{n} S_{b_1 \ldots b_k}} \\
&= F\left(1 + \frac{\varepsilon}{2n}\right)^n \\
&\le F(1+\varepsilon).
\end{aligned}$$

where the second inequality follows from picking $\frac{\varepsilon}{12n} \le 1/6$ as above. Similarly, we can establish that

$$\frac{1}{\prod_{k=1}^{n} \widetilde{S}_{b_1 \ldots b_k}} \ge \frac{F}{1+\varepsilon}$$

and hence $\mathcal{A}$ is a FPRAS for #**DNF**. □

Now, the runtime of this algorithm is upper bounded by

$$\begin{aligned}
&\leq \#\text{ recursions} \cdot \#\text{ samples to get } \frac{\varepsilon}{12n} \text{ additive error} \cdot \text{runtime of uniform generator} \\
&\leq n \cdot \text{poly}\left(\left(\frac{\varepsilon}{12n}\right)^{-1}\left(\frac{1}{4n}\right)^{-1}\right) \cdot \text{poly}(n) \\
&= O(\text{poly}(n, 1/\varepsilon)).
\end{aligned}$$

We can bound the error of failure by union bound over all recursion levels as

$$\Pr[\text{algorithm fails}] \leq \sum_{i=1}^{n} \Pr[\text{bad estimate}] \leq n \cdot \frac{1}{4n} \leq \frac{1}{4}.$$

Thus we have our desired FPRAS for #DNF.

2 Uniform Generation via Counting

In general, such approximation scheme works for any downward self-reducible problems with a polynomial time uniform generator. In fact, for any downward self-reducible problem in **NP**, we have

$$\text{poly-time approximate counting \#solutions} \iff \text{poly-time almost uniform generation.}$$

For DNF, we just showed that uniform generation implies approximate counting.

Here we will show a simpler statement. Specifically we will show that perfect counting for #DNF implies perfect uniform generation. Given a perfect counter for #DNF, consider the following algorithm:

Recursive Algorithm:

- At $b_1 \dots b_i$, use counter to compute $r_0 = F_{b_1,\dots b_i 0}, r_1 = F_{b_1 \dots b_i 1}$.
- Go to $b_1 \dots b_{i+1}$ where $b_{i+1} = 0$ with probability $\frac{r_0}{r_0+r_1}$; otherwise go to $b_1 \dots b_{i+1}$ where $b_{i+1} = 1$.

Claim 4. *The above algorithm always reaches a satisfying assignment.*

Proof. By construction, the algorithm never take a branch with no satisfying assignments since the branch would be picked with probability 0. □

Claim 5. *The above algorithm outputs a satisfying assignment uniformly at random.*

Proof. Let $b_1, \dots, b_n$ be a satisfying assignment, then

$$\begin{aligned}
\Pr[\text{output } b_1, \dots, b_n] &= \frac{F_{b_1}}{F} \cdot \frac{F_{b_1 b_2}}{F_{b_1}} \cdot \frac{F_{b_1 b_2 b_2}}{F_{b_1 b_2}} \cdot \dots \cdot \frac{1}{F_{b_1 \dots b_n}} \\
&= \frac{1}{F},
\end{aligned}$$

which is the same for every satisfying assignment. □

We remark in the end that for approximate counters with approximation factor ε', we have

$$\Pr[\text{output } b_1, \dots, b_n] \leq \frac{1}{F} \cdot \left(\frac{1+\varepsilon'}{1-\varepsilon'}\right)^n \leq \frac{1}{F} \cdot \frac{1}{1-\varepsilon}$$

if we choose $\varepsilon' < \frac{\varepsilon}{2n}$. Then we can get close to uniform generation of satisfying assignments.

3 Randomized Complexity

Definition 6. *A* Language L *is a subset of* $\{0,1\}^*$.

E.g. $\{x \mid x$ is a CNF formula that is satisfiable$\}$.

Definition 7. P *is the class of languages* L *with polynomial time* deterministic *algorithm* A *such that on input* x*,*

$$A(x) = \begin{cases} \textit{accept} & \textit{if } x \in L \\ \textit{reject} & \textit{if } x \notin L. \end{cases}$$

Definition 8. RP *is the class of languages* L *with polynomial time* probabilistic *algorithm* A *such that on input* x*,*

$$\Pr[A \textit{ accepts} \mid x \in L] \geq 1/2,$$

$$\Pr[A \textit{ rejects} \mid x \notin L] = 1.$$

This is also called 1-sided error since A *always rejects if* $x \notin L$*.*

Definition 9. BPP *is the class of languages* L *with polynomial time* probabilistic *algorithm* A *such that on input* x*,*

$$\Pr[A \textit{ accepts} \mid x \in L] \geq 2/3,$$

$$\Pr[A \textit{ rejects} \mid x \notin L]2/3.$$

This is also called 2-sided error.

4 Derandomization via Enumeration

Now we present the simplest form of derandomization.

Given: randomized algorithm $\mathcal{A}$ for language L that runs in time $t(n)$ with $r(n)$ random bits, input x.
Deterministic Algorithm: run $\mathcal{A}$ on *every* possible random string of length $r(n)$, output majority answer.

Behavior Suppose $\mathcal{A} \in \mathbf{BPP}$, then

- If $x \in L$, then $\geq 2/3$ of random strings cause $\mathcal{A}$ to accept $\implies$ majority answer is accept.
- If $x \notin L$, then $\geq 2/3$ of random strings cause $\mathcal{A}$ to reject $\implies$ majority answer is reject.

Runtime

$$O(2^{r(n)} \cdot t(n)) \leq O(2^{t(n)} \cdot t(n)).$$

Note that we are simply using $r(n) \leq t(n)$ by assuming the computation model of a Turing machine, but if we have a better bound on $r(n)$, say $r(n) = O(\log n)$ and $t(n) = \text{poly}(n)$, this would improve the time bound to $\text{poly}(n)$.

Corollary 10. $\mathbf{BPP} \subseteq \mathbf{EXP} := \mathbf{DTIME}\left(\bigcup_c 2^{n^c}\right)$.