6.842 Randomness and Computation February 17, 2022

Lectures on Uniform Generation and Approximate Counting

Lecturer: Ronitt Rubinfield *Scribe: Yuchong Pan*

1 Uniform Generation for DNF

A *DNF formula* is a Boolean formula that consists of "OR of ANDs", e.g., $\varphi(x_1, x_2, x_3) = x_1\overline{x_2} \vee \overline{x_1}x_3$. The task is to output a random satisfying assignment to a DNF formula, uniformly over all satisfying assignments.

Consider the special case with only one conjunction. We can satisfy literals in the conjunction, and set the others randomly. For instance, for $\varphi(x_1, x_2, x_3) = x_1\overline{x_2}$, we set $x_1 = \mathrm{T}, x_2 = \mathrm{F}$, and set $x_3 \in \{\mathrm{T}, \mathrm{F}\}$ uniformly.

Now consider the two conjunction case, e.g., $\varphi(x_1, x_2, x_3) = x_1x_2 \vee x_3$. We give an algorithm attempt in Algorithm 1. However, two problems arise, as illustrated in Figure 1. Firstly, conjunctions have different numbers of assignments. Secondly, some assignments can be chosen in multiple ways. In particular, using Algorithm 1, the probability that TTF is chosen is $1/2 \cdot 1/2 = 1/4$, the probability that TTT is chosen is $1/2 \cdot 1/2 + 1/2 \cdot 1/4 = 3/8$, and that the probability that each of TFT, FTT, FFT is chosen is $1/2 \cdot 1/4 = 1/8$.

```
1 pick $i \in \{1, 2\}$ uniformly
2 set variables in conjunction $i$ to T
3 set others uniformly
```

Algorithm 1: An algorithm attempt of uniform generation for DNF with 2 conjunctions.

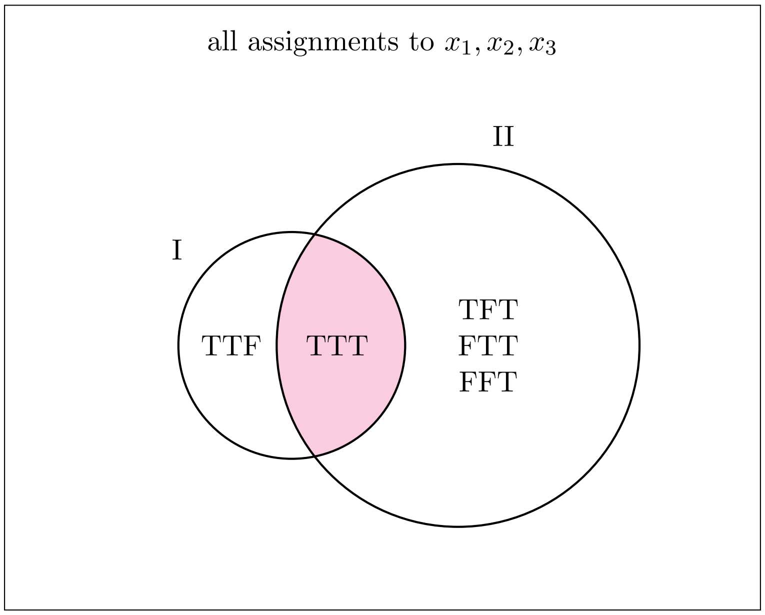

Figure 1: An illustration of problems in Algorithm 1.

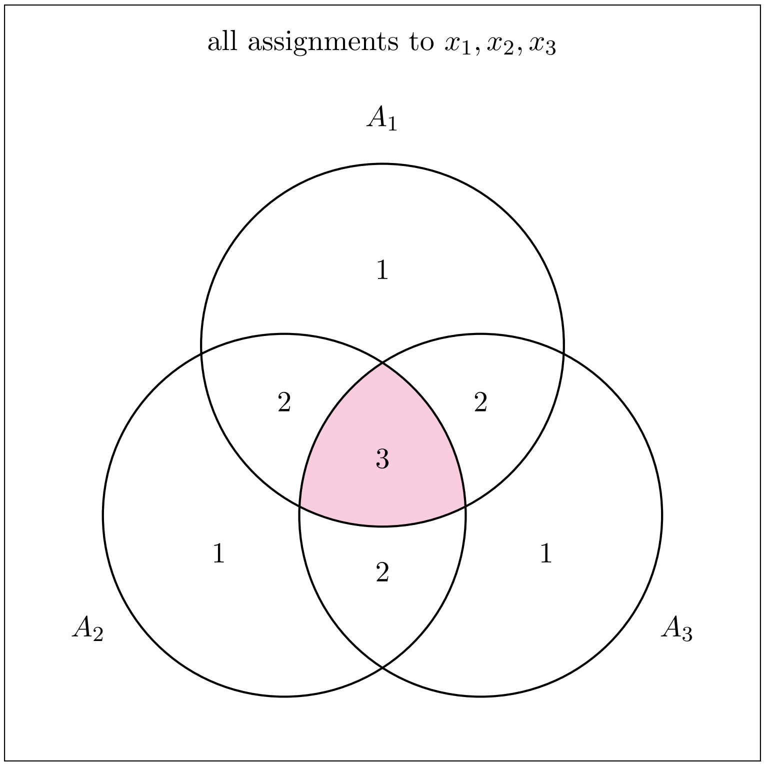

Figure 2: An example for rejection sampling, where the number in each region indicates the number of conjunctions each assignment satisfies.

The first problem is easy to fix; we simply replace line 2 in Algorithm 1 to "pick $i \in \{1, 2\}$ *proportional to the number of satisfying assignments for that conjunction.*" For the second problem, we introduce the notion of *rejection sampling.* Let m be the number of conjunctions in φ. For each $i \in [m]$, let $A_i = \{\bar{x} = (x_1, \ldots, x_n) : \bar{x}$ satisfies the i^{th} conjunction$\}$. Consider the example illustrated in Figure 2. In the shaded region in Figure 2, each assignment is 3 times more likely to be picked; therefore, we correct our algorithm by tossing a coin of bias $1/3$ to decide if to output the assignment. We summarize our final algorithm in Algorithm 2.

```
1 $A_i = \{\bar{x} = (x_1, \ldots, x_n) : \bar{x}$ satisfies the $i^{\text{th}}$ conjunction$\}$ for each $i \in [m]$
2 **repeat**
3     pick $i \in [m]$ with probability $|A_i| / \sum_{j=1}^{m} |A_j|$
4     pick $\bar{b} \in A_i$ uniformly
5     $t_{\bar{b}} \leftarrow |\{j \in [m] : \bar{b} \in A_j\}|$ // $t_{\bar{b}} \geq 1$ since $\bar{b}$ satisfies $C_i$
6     output $\bar{b}$ with probability $1/t_{\bar{b}}$
7 **until** *success*
```

Algorithm 2: Uniform generation for DNF, where the input is a DNF formula $\varphi = \bigvee_{i=1}^{m} C_i$ and C_i is a conjunction for each $i \in [m]$.

We show the uniformity of Algorithm 2. For each $\bar{b}$ that satisfies φ,

$$\mathbb{P}\left[\bar{b} \text{ is output in round } i\right] = \frac{1}{t_{\bar{b}}} \sum_{\substack{j \in [m] \\ \bar{b} \in A_j}} \mathbb{P}[\text{pick } j \text{ in round } i] \cdot \frac{1}{|A_j|}$$

$$
\begin{aligned}
&= \frac{1}{t_{\bar{b}}} \sum_{\substack{j \in [m] \\ \bar{b} \in A_j}} \frac{|A_j|}{\sum_{k=1}^m |A_k|} \cdot \frac{1}{|A_j|} \\
&= \frac{1}{t_{\bar{b}}} \sum_{\substack{j \in [m] \\ \bar{b} \in A_j}} \frac{1}{\sum_{k=1}^m |A_k|} \\
&= \frac{1}{\sum_{k=1}^m |A_k|}.
\end{aligned}
$$

Hence, the probability that $\bar{b}$ is output in round i is the same for all $\bar{b}$ satisfying φ, so the generation is uniform. Moreover, $\mathbb{P}[\text{loop success}] \geq 1/\max t_{\bar{b}} \geq 1/m$, so $\mathbb{E}[\#\text{ loops until success}] \leq m$. This shows that Algorithm 2 runs in polynomial time in expectation.

2 Counting Problems

Definition 1. Let #P denote the class of problems that count the number of accepting paths in a polynomial time nondeterministic Turing machines.

Definition 2. We say that a problem is #P-*complete* if

(a) it is in #P;

(b) every problem in #P has a polynomial time reduction to it.

Let #SAT denote the problem that counts the number of assignments satisfying a Boolean formula φ. Then #SAT is #P-complete. Similarly, let #DNF denote the problem that counts the number of assignments satisfying a DNF formula φ. Although DNF $\in$ P, we show below that #DNF $\notin$ P. However, it turns out that #DNF is #P-complete.

Proposition 3. *#DNF $\notin$ P.*

Proof. We show that if #DNF $\in$ P, then CNF $\in$ P. By De Morgan's law, a clause $\overline{A \vee B} = \overline{A} \wedge \overline{B}$, which is a conjunction. Therefore, Given a CNF formula φ in n variables, φ is satisfiable if and only if $\overline{\varphi}$ (which is a DNF formula) has at most $2^n - 1$ satisfying assignments. □

3 Approximate Counting and Downward Self-Reducibility

Definition 4. A *fully polynomial randomized approximation scheme (FPRAS)* for DNF is an algorithm which, given a DNF formula φ and $\varepsilon > 0$, outputs $y \in \mathbb{Z}_+$ such that $z/(1+\varepsilon) \leq y \leq z(1+\varepsilon)$ with probability at least 3/4 in time polynomial in $|\varphi$ and $1/\varepsilon$, where z is the number of satisfying assignments to φ.

In Problem Set 1 Problem 1, we shall show that Definition 4 implies that an FPRAS runs in time polynomial in $|\varphi|$, $1/\varepsilon$ and $\log \delta$ if we replace the probability bound 3/4 with $1 - \delta$.

Proposition 5. *If there exists an FPRAS for* #SAT*, then* SAT $\in$ BPP*.*

Proof. We give a probabilistic algorithm for SAT. Given a Boolean formula φ, we call the FPRAS on φ with $\varepsilon = 1/2$ (or any constant); if the output of the FPRAS is positive, then output "satisfiable"; else output "unsatisfiable."

If φ is satisfiable, then the number of satisfying assignments is at least 1, so $y \geq 1/(1+\varepsilon) > 0$, which impies that the algorithm outputs "satisfiable." Otherwise, the number of satisfying assignments is 0, so $y = 0$, which implies that the algorithm outputs "unsatisfiable." □

Downward self-reducibility (DSR) refers to the idea of solving a problem via solving its sub-problems. For DNF, we can compute the number of satisfying assignments using the DSR tree in Figure 3, where $F_{b_1,\ldots,b_k}$ denotes the number of satisfying assignments for $\varphi(b_1,\ldots,b_k,x_{k+1},\ldots,x_n)$.

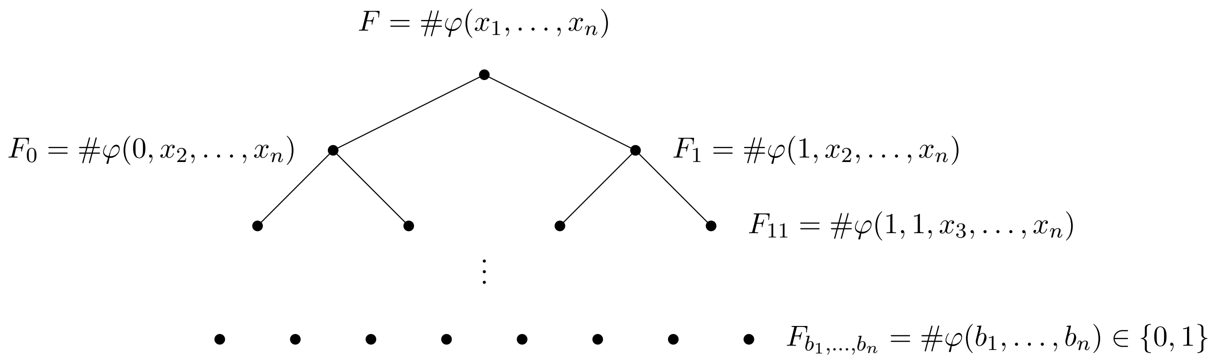
$F = \#\varphi(x_1,\ldots,x_n)$
$F_0 = \#\varphi(0,x_2,\ldots,x_n)$
$F_1 = \#\varphi(1,x_2,\ldots,x_n)$
$F_{11} = \#\varphi(1,1,x_3,\ldots,x_n)$
$F_{b_1,\ldots,b_n} = \#\varphi(b_1,\ldots,b_n) \in \{0,1\}$

Figure 3: The DSR tree for DNF.

WLOG consider the root of the DSR tree. Then $F = F_0 + F_1$. Let $S_1 = F_1/F$ be the fraction of satisfying assignments for which $x_1 = \mathrm{T}$. Then $F = F_1/S_1$, where F_1 can be computed recursively via $F_1 = F_{11}/S_{11}$, with S_{11} the fraction of satisfying assignments to $\varphi(1,x_2,\ldots,x_n)$ for which $x_2 = \mathrm{T}$. The main insights of approximate counting for DNF are:

(i) we can estimate S_1 via sampling;

(ii) we can uniformly generate satisfying assignments for DNF.

To estimate S_1, we use the simple algorithm in Algorithm 3.

```
1 uniformly generate $k$ satisfying assignments to $\varphi$
2 $\widetilde{S_1} \leftarrow (\#$ generated satisfying assignments with $x_1 = \mathrm{T})/k$
```

Algorithm 3: An algorithm that estimates S_1.

For $k \in \mathbb{Z}_+$ and $b_1,\ldots,b_k \in \{0,1\}$, define $S_{b_1,\ldots,b_n}$ to be the fraction of satisfying assignments to $\varphi(b_1,\ldots,b_{k-1},x_k,\ldots,x_n)$ for which $x_k = b_k$ (note that we use T/F and 1/0 interchangeably), and $\widetilde{S_{b_1,\ldots,b_k}}$ to be the estimate of $S_{b_1,\ldots,b_k}$ similarly computed as in Algorithm 3. For $b_1,\ldots,b_n \in \{0,1\}$,

$$F = \frac{F_{b_1}}{S_{b_1}} = \frac{1}{S_{b_1}} \cdot \frac{F_{b_1,b_2}}{S_{b_1,b_2}} = \frac{F_{b_1,b_2,b_3}}{S_{b_1}S_{b_1,b_2}S_{b_1,b_2,b_3}} = \ldots = \frac{1}{\prod_{i=1}^n S_{b_1,\ldots,b_i}}.$$

Two difficulties arise:

(i) What if $F_{b_1,\ldots,b_i} = 0$ for some $i \in [n]$?

(ii) Are the approximations of $S_{b_1,\ldots,b_i}$'s for $i \in [n]$ enough?

The solution to both of the difficulties are to always go down to the "large enough" child. We note that for the choice of the "larger" child, a small additive error implies a small multiplicative error. To see this, suppose that we estimate each $S_{b_1,\ldots,b_i}$ to within an additive error of $\varepsilon/(18n)$ (by Chernoff's bound, we need $\mathrm{poly}(n/\varepsilon)$ samples). Therefore, if $1/3 \le r \le 1$, then

$$r + \frac{\varepsilon}{18n} = r\left(1 + \frac{\varepsilon}{18nr}\right) \le r\left(1 + \frac{\varepsilon}{6n}\right),$$

$$r - \frac{\varepsilon}{18n} = r\left(1 - \frac{\varepsilon}{18nr}\right) \geq r\left(1 - \frac{\varepsilon}{6n}\right).$$

Therefore, the output is at most

$$\frac{F_{b_1}}{\widetilde{S_{b_1}}} \leq \dots \leq \frac{1}{\prod_{i=1}^{n} \widetilde{S_{b_1,\dots,b_i}}} \leq \frac{1}{\prod_{i=1}^{n} S_{b_1,\dots,b_n}\left(1 - \frac{\varphi}{6n}\right)} \leq \frac{\left(1 + \frac{\varepsilon}{3n}\right)^n}{\prod_{i=1}^{n} S_{b_1,\dots,b_n}} \leq F\left(1 + \frac{\varepsilon}{2}\right).$$

We can similarly show that the output is at least $F/(1+\varepsilon)$.

4 Equivalence Between Uniform Generation and Counting

Theorem 6 (Jerrum-Valiant-Vazirani)**.** *For any problem in NP that is downward self-reducible, one can approximately count the number of solutions to the problem in polynomial time if and only if one can perform generate solutions to the problem almost uniformly in polynomial time.*

In this note, we prove the easier case where both of uniform generation and counting are perfect. That perfect uniform generation implies perfect counting is proved in Section 3. For the other direction, given a counter for #DNF, we define a recursive algorithm in Algorithm 4 that performs uniform generation for DNF. For any satisfying assignment $b_1, \dots, b_n$, the probability that Algorithm 4 outputs $b_1, \dots, b_n$ is

$$\frac{F_{b_1}}{F} \cdot \frac{F_{b_1,b_2}}{F_{b_1}} \cdots \frac{1}{F_{b_1,\dots,b_{n-1}}} = \frac{1}{F}.$$

Therefore, the probability that every satisfying assignment is output by Algorithm 4 is equal, so Algorithm 4 is a uniform generator for DNF.

```
1 // at $b_1, \dots, b_i$
2 use the counter to compute $r_0 = F_{b_1,\dots,b_i,0}$ and $r_1 = F_{b_1,\dots,b_i,1}$
3 go left on the DSR tree with probability $r_0/(r_0 + r_1)$; else go right
4 output $b_1, \dots, b_n$ when reaching a leaf
```

Algorithm 4: A uniform generator for DNF given a counter.