Lecture 8

Lecturer: Ronitt Rubinfeld *Scribe: Matthew Ince*

1 DNF formulas

Definition 1 *Given n variables $x_1, \ldots, x_n$, a* DNF formula *$F = F_1 \vee F_2 \vee \ldots \vee F_m$ on m clauses and n variables is a boolean formula where each clause F_i is of the form $F_i = y_{j_1} \wedge y_{j_2} \wedge \ldots$, and where the y_j are literals x_k or $\overline{x_k}$.*

Our goal is to uniformly randomly generate satisfying assignments of *DNF formulas*. Every non-trivial DNF formula has a satisfying assignment, because satisfying the formula reduces to satisfying a single clause F_i. For instance, to satisfy the formula

$$F = x_1 x_2 \overline{x_3} \vee \overline{x_1} x_2 x_4,$$

we could satisfy the first clause by choosing $x_1 = x_2 = T$, $x_3 = F$. As an aside, note that if the $\vee$ are replaced by XORs $\oplus$, then F becomes a polynomial in the variables over $\mathbb{Z}_2$, and finding satisfying assignments reduces to random polynomial zero-finding.

Not surprisingly, generating satisfying assignments for a DNF formula is closely related to counting the number of such assignments. However, exact answers to this problem are difficult to obtain: the negation of a DNF formula is a so-called CNF formula, *e.g.* $(x \vee y \vee \overline{z}) \wedge (x \vee \overline{x} \vee y)$. CNF formulas are the subject of the famous $3CNF - SAT$ problem, which shows that finding satisfying assignments for CNF formulas with three variables per clause is NP-complete. Since counting the number of satisfying assignments of a DNF formula would reveal the existence of a satisfying assignment of its negation, counting the number of assignments is a problem of class #P.

We first find satisfying assignments when $m = 1$. In this case, F only has a single clause, *e.g.* $F_1 = x_1 x_2 \overline{x_3}$. We may generate all satisfying assignments of this clause by choosing $x_1 = T, x_2 = T, x_3 = F$, and arbitrary values for each other x_i. Note that there are 2^{n-3} satisfying assignments in all.

If we have more than one clause, we could simply pick a clause, then pick a random satisfying assigment for that clause. However, this procedure is biased toward assignments satisfying several different clauses. Because we want a uniform distribution of outputs, we use a slightly more complicated selection routine. For convenience, let S_i be the set of assignments satisfying F_i.

Algorithm A

To randomly generate π satisfying F:

Step i: Pick $i \in [m]$ with probability $\frac{|S_i|}{\sum |S_i|}$.
Then pick a random satisfying assignment π of F_i.

Step ii: Compute $\ell = |\{j \in \{1, 2, \ldots, m\} : \pi \in S_j\}|$.
Then toss a coin with bias $1/\ell$.
If the coin is "Heads", OUTPUT π and halt.
Otherwise, restart at step I.

Intuitively, step i is the naive selection routine, and step ii compensates for assignments π in several S_i: if π is in ℓ different sets S_i, then each of these S_i should be $1/\ell$ times as likely to select π to ensure a uniform distribution. We now prove some claims about algorithm A:

Claim 2 *Algorithm A outputs satisfying assignments uniformly at random.*

Proof of Claim 2: It suffices to show that each loop iteration is equally likely to output all satisfying assignments π. For a given π, as before let $\ell = |\{j \in \{1, 2, \ldots, m\} : \pi \in S_j\}|$. By conditional probability,

$$\begin{aligned}
\Pr[\pi \text{ picked in step 1}] &= \sum_{j \in [m] \ s.t. \ \pi \in S_j} \Pr[\text{Step i picks clause } j] \frac{1}{|S_j|} \\
&= \sum_{j \in [m] \ s.t. \ \pi \in S_j} \frac{|S_j|}{\sum |S_j|} \cdot \frac{1}{S_j} \\
&= \sum_{j \in [m] \ s.t. \ \pi \in S_j} (\sum |S_j|)^{-1} \\
&= \frac{\ell}{\sum |S_j|}.
\end{aligned}$$

So the probability that this loop iteration actually outputs π is $\frac{1}{\ell} \frac{\ell}{\sum |S_j|} = \frac{1}{\sum |S_j|}$, which is independent of π. ■

Claim 3 *The number of loops needed to choose π satisfies*

$$E[\#\ loops\ until\ OUTPUT] \leq m.$$

Proof of Claim 3: For each π examined, we have $\ell \leq m$, giving $1/\ell \geq 1/m$. A coin with bias p has $1/p$ expected runs until it outputs "Heads", so

$$E[\#\text{ loops}] = 1/bias \leq m.$$

■

2 P-relations

Definition 4 *Let R be a binary relation $R \subset \{0,1\}^* \times \{0,1\}^*$ on strings. We say R is a* P-relation *if*

1. *For each $(x, y) \in R$, we have $|y| = O(poly(|x|))$.*
2. *There exists a polynomial time procedure for deciding if $(x, y) \in R$.*

For example, consider $R_{SAT} = \{(x, y) \mid x \text{ a boolean formula, } y \text{ a satisfying assignment of } x\}$.

Claim 5 *We have $L \in NP$ if and only if there exists a P-relation R such that $x \in L$ holds if and only if there exists y with $(x, y) \in R$.*

The (trivial) proof of this fact comes next time. Note that y can be thought of as "corroborating" whether $x \in L$.