1 DNF formulas

**Definition 1** Given $n$ variables $x_1, \ldots, x_n$, a DNF formula $F = F_1 \lor F_2 \lor \ldots \lor F_m$ on $m$ clauses and $n$ variables is a boolean formula where each clause $F_i$ is of the form $F_i = y_1 \land y_2 \land \ldots$, and where the $y_j$ are literals $x_k$ or $\lnot x_k$.

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 $F = x_1 x_2 x_3 \lor \lnot x_1 x_2 x_4$, we could satisfy the first clause by choosing $x_1 = x_2 = T$, $x_3 = F$. As an aside, note that if the $\lor$ 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. $(x \lor y \lor \lnot z) \land (x \lor \lnot x \lor y)$. 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. $F_1 = x_1 x_2 \lnot x_3 \lor \lnot x_1 x_2 x_4$. 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 assignment 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 $\pi$ satisfying $F$:

1. **Step i:** Pick $i \in [m]$ with probability $\frac{|S_i|}{\sum |S_j|}$.
2. Then pick a random satisfying assignment $\pi$ of $F_i$.
3. **Step ii:** Compute $\ell = |\{ j \in \{1,2,\ldots,m\} : \pi \in S_j \}|$.
4. Then toss a coin with bias $1/\ell$.
   - If the coin is “Heads”, OUTPUT $\pi$ and halt.
   - Otherwise, restart at step i.

Intuitively, step i is the naive selection routine, and step ii compensates for assignments $\pi$ in several $S_i$: if $\pi$ is in $\ell$ different sets $S_i$, then each of these $S_i$ should be $1/\ell$ times as likely to select $\pi$ 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 \( \pi \). For a given \( \pi \), as before let \( \ell = |\{ j \in \{1, 2, \ldots, m \} : \pi \in S_j \}| \). By conditional probability,

\[
\Pr[\pi \text{ picked in step 1}] = \sum_{j \in [m] \text{ s.t. } \pi \in S_j} \Pr[\text{Step i picks clause } j] \frac{1}{|S_j|} = \frac{\sum_{j \in [m] \text{ s.t. } \pi \in S_j} |S_j|^{-1}}{\ell} = \frac{1}{\sum |S_j|^2}.
\]

So the probability that this loop iteration actually outputs \( \pi \) is \( \frac{1}{\ell \sum |S_j|^2} \), which is independent of \( \pi \).

Claim 3 The number of loops needed to choose \( \pi \) satisfies

\[ E[\# \text{ loops until OUTPUT}] \leq m. \]

Proof of Claim 3: For each \( \pi \) 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 \(|y| = O(\text{poly}(|x|))\).

2. There exists a polynomial time procedure for deciding if \((x, y) \in R\).

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

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 \( x \in L \).