CS49/249 (Randomized Algorithms), Spring 2021 : Lecture 7

Topic: Estimating #DNF : Importance Sampling

Disclaimer: These notes have not gone through scrutiny and in all probability contain errors. Please discuss in Piazza/email errors to deeparnab@dartmouth.edu

- A DNF formula is a Boolean formula with n variables and m *disjunctive* clauses $C_1, \ldots, C_m$; a clause is a disjunction if it is an AND of a collection of literals (a Boolean variable or its negation). The formula is an OR of these clauses. In English, a DNF formula evaluates to true if ***at least one*** of its clauses evaluates to true, and a disjunctive clause evaluates to true if ***all*** of its literals evaluates to true. It is convenient to think of ANDs as product, ORs as sum, negation as difference from 1, true as 1, false as 0, and a formula evaluates to true if its value is > 0. For example,

  $$\phi = x_1 x_2 \overline{x_3} + \overline{x_1} x_2 \overline{x_4} + \overline{x_2} x_3 x_4$$

  is a DNF formula with 4 variables and 3 clauses. It has many satisfying assignments: $(\mathsf{true}, \mathsf{true}, \mathsf{false}, \mathsf{true})$ or $(1, 1, 0, 1)$ is simply one.

  The #DNF problem, or the DNF counting problem, takes as input a DNF formula and returns the ***number*** of satisfying assignments ϕ has. This is a special case of the general counting problem : given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, how many $\mathbf{x} \in \{0,1\}^n$ leads to $f(\mathbf{x}) = 1$. Throughout, we will use N to denote the true number of satisfying assignments. Our goal is to obtain an (ε, δ)-multiplicative estimate of N, and we wish to find an algorithm which runs in polynomial time.

- ***Try 1: Rejection Sampling.*** We start by describing a "not-so-great" approach which is traditionally called *rejection sampling*. The idea is extremely simple: we sample an $\mathbf{x} \in \{0,1\}^n$ and check if $\phi(x) = 1$ or not. Here ϕ is the DNF formula which I am thinking of as a Boolean function. If it satisfies, we set $\widehat{\mathsf{est}} = 2^n$, otherwise we set $\widehat{\mathsf{est}} = 0$. The time taken to sample is precisely the time taken to evaluate $\phi(x) =$ or not, and this takes $O(m + n)$ time.

  What is $\mathbf{Exp}[\widehat{\mathsf{est}}]$? Well,

  $$\mathbf{Exp}[\widehat{\mathsf{est}}] = 2^n \cdot \mathbf{Pr}[\phi(\mathbf{x}) = 1] = 2^n \cdot \frac{\mathsf{N}}{2^n} = \mathsf{N}$$

  Thus, $\widehat{\mathsf{est}}$ is an unbiased estimate. To apply the boosting theorem to obtain an (ε, δ)-estimate, we need to figure out $\mathbf{Var}[\widehat{\mathsf{est}}]$. In particular, we are interested in the ratio of variance-to-squared-mean. We see,

  $$\mathbf{Var}[\widehat{\mathsf{est}}] = (2^n)^2 \cdot \frac{\mathsf{N}}{2^n} - \mathsf{N}^2 = \mathsf{N} \cdot (2^n - \mathsf{N}) \;\; \Rightarrow \;\; \frac{\mathbf{Var}[\widehat{\mathsf{est}}]}{\mathbf{Exp}^2[\widehat{\mathsf{est}}]} = \frac{2^n}{\mathsf{N}} - 1$$

  If N is (very) large, then we see that this number is not bad. However, even if $\mathsf{N} = 1.9^n$, then this ratio is exponentially large in n. Therefore, this estimate would need a lot of independent samples to give anything good. By the way, observe that the above process didn't use the fact that ϕ was a DNF at all, and would work for any Boolean function. The better estimate will use the structure of ϕ.

- ***Importance Sampling.*** The main reason the above rejection sampling algorithm doesn't work is because most of the time is spent rejecting. When $\mathsf{N} \ll 2^n$, then we are indeed trying to find the size of a (really tiny) needle in a haystack, and if we randomly reach out, then we will be clutching at straws (ha! ha!). The idea of importance sampling is to directly get to the needle.

To set up the stage, let's introduce two bits of notation. For clause C_i, let us denote $S_i := \{\mathbf{x} \in \{0,1\}^n : \mathbf{x} \text{ satisfies } C_i\}$ as the set of assignments which satisfies the ith clause. What we are interested in is (and this is where DNF-ness is used)

$$\mathsf{N} = \left| \bigcup_{i=1}^{m} S_i \right|$$

Observe two things: (a) $|S_i|$ is very easy to evaluate; indeed, $|S_i| = 2^{n-k_i}$ where k_i is the number of literals in C_i. This is because there is exactly one way to set the literals in C_i to set it to true, and the remaining variables can take whatever values they want, and (b) indeed, any $\mathbf{x} \in S_i$ can be sampled extremely easily as well. Furthermore, if all the S_i's were disjoint, then N would simply be the sum of the $|S_i|$'s, and we would be done. In general, the sum is a crude upper bound as the S_i's may not be disjoint at all. This motivates the next piece of notation.

For every assignment $\mathbf{x} \in \{0,1\}^n$, define $N(\mathbf{x})$ to be the number of clauses $\mathbf{x}$ satisfies. That is,

$$N(\mathbf{x}) := |\{j \in [m] : \mathbf{x} \in S_j\}|$$

Note that for any $\mathbf{x}$, the quantity $N(\mathbf{x})$ can be evaluated in $O(mn)$ time as well: we simply go over all clauses and count, and each clause takes $O(n)$ time to check.

The idea of *importance sampling* is the following: *the algorithm always samples an assignment $\mathbf{x} \in \bigcup_{i=1}^{m} S_i$, and then corrects the estimate if $N(\mathbf{x})$ is large.* Here is the full algorithm. This algorithm is due[1] to Richard Karp and Michael Luby from 1983.

```
1: procedure ESTIMATE-#DNF($\phi$):
2:     Set $S := \sum_{i=1}^{m} |S_i|$. ▷ Crude Upper Bound
3:     Sample clause $C_i$ with probability $p_i := \frac{|S_i|}{|S|}$. ▷ Importance Sampling
4:     Sample $\mathbf{x} \in S_i$ uar. ▷ Set literals in $C_i$ deterministically. Everything else, uar
5:     Evaluate $N(\mathbf{x}) = |\{j \in [m] : \mathbf{x} \in S_j\}|$. ▷ $O(mn)$ time; $N(\mathbf{x}) \geq 1$.
6:     return $\widehat{\mathsf{est}} = \frac{|S|}{N(\mathbf{x})}$. ▷ Correction.
```

Remark: *In fact, the same algorithm can be used to estimate the size of the union of a collection of sets as long as* three *properties hold: (a) we know $|S_i|$ for every set in the collection, (b) we can* sample *$e \in S_i$ for every set in the collection, and (c) for any element e, we can figure out how many sets contain this e very quickly. This viewpoint leads to more applications than just counting satisfying assignments to DNF formulas.*

- ***Analysis.*** We first show that $\widehat{\mathsf{est}}$ is an unbiased estimate, and then we show it doesn't have high variance when compared with the mean squared.

[1] *Monte-Carlo algorithms for enumeration and reliability problems.* R. M. Karp and M. Luby, Proc, 24th IEEE Symp. of Found. of Comp. (FOCS), 1983

Claim 1. $\mathbf{Exp}[\widehat{\mathsf{est}}] = \mathsf{N}$

Proof. First note that the sampled $\mathbf{x}$ in Line 4 is a satisfying assignment of ϕ because it lies in at least one of the S_i's. In particular, if we define $U := \bigcup_{i=1}^m S_i$, then we never sample $\mathbf{x} \notin U$. Therefore, when calculating the expectation of $\widehat{\mathsf{est}}$, we need only consider $\mathbf{x} \in U$.

$$\begin{aligned}
\mathbf{Exp}[\widehat{\mathsf{est}}] &= \sum_{\mathbf{x}\in U} \frac{S}{N(\mathbf{x})} \cdot \mathbf{Pr}[\mathbf{x} \text{ sampled in Line 4}] \\
&= \sum_{\mathbf{x}\in U} \frac{S}{N(\mathbf{x})} \cdot \sum_{i\in[m]:\mathbf{x}\in S_i} \underbrace{\mathbf{Pr}[\mathbf{x} \text{ sampled in Line 4} \mid i \text{ sampled in Line 3}]}_{\frac{1}{|S_i|}} \cdot \underbrace{\mathbf{Pr}[i \text{ sampled in Line 3}]}_{\frac{|S_i|}{S}} \\
&= \sum_{\mathbf{x}\in U} \frac{S}{N(\mathbf{x})} \cdot \underbrace{\sum_{i\in[m]:\mathbf{x}\in S_i} \frac{1}{S}}_{\frac{N(\mathbf{x})}{S}} = \sum_{\mathbf{x}\in U} 1 = |U| = \mathsf{N} \quad \square
\end{aligned}$$

Claim 2. $\mathbf{Var}[\widehat{\mathsf{est}}] \le m \cdot \mathbf{Exp}^2[\widehat{\mathsf{est}}]$

Proof. This follows rather crudely from (a) $\mathbf{Var}[\widehat{\mathsf{est}}] \le M \cdot \mathbf{Exp}[\widehat{\mathsf{est}}]$ where M is the *maximum* value $\widehat{\mathsf{est}}$ can take. Note, that since $N(\mathbf{x}) \ge 1$ in Line 5, we get that $\widehat{\mathsf{est}} \le S \le m \cdot \mathsf{N} = m\,\mathbf{Exp}[\widehat{\mathsf{est}}]$, where we have used another crude estimate that the sum of m cardinalities is at most m times the union. $\square$

Theorem 1. For any ε, δ, one can obtain an (ε, δ)-multiplicative approximation to N, the number of satisfying assignments to a DNF formula, with $O(\frac{m}{\varepsilon^2}\ln(1/\delta))$-runs of ESTIMATE-#DNF algorithm. This leads to a total running time of $O(nm^2\varepsilon^{-2}\ln(1/\delta))$.

Remark: *The above description is from the* conference *version of the paper. There is a journal version*[a] *which includes Neal Madras as an author. They give a slightly different estimation algorithm, which shaves off one m from the running time. This is a really nice algorithm, and I leave this as a reading project.*

[a] *Monte-Carlo approximation algorithms for enumeration problems.* R. M. Karp, M. Luby, and N. Madras, Journal of Algorithms, 10:429–448, 1989

Learning Tidbits:

- Algorithm Design: *Rejection sampling, although not ideal for #DNF, is something to be aware of as it is very general. Importance sampling, or non-uniform sampling, is also a very powerful tool. This is useful even when we want to do a uniform sampling over a universe to which we don't have direct access (see problem set for an example).*
- Analysis: *Sometimes, as in #DNF, to get an unbiased estimate, one needs to be mindful of what one returns. For instance, here once we sampled* $\mathbf{x}$*, we returned* $\frac{S}{N(\mathbf{x})}$.