Randomness and Computation
or, “Randomized Algorithms”

Mary Cryan
School of Informatics
University of Edinburgh

The DNF counting problem

On Tuesday we showed that using the Naïve Monte Carlo method is infeasible (for any application, but specifically for DNF) if the number of solutions is a small fraction of the sampled set. For DNF this happens (say) when we have a small number of very large clauses. A random assignment is very unlikely to hit the good assignments.

However, we can develop an FPRAS for DNF if we refine our view of the sampling.

FPRAS for DNF counting

Our formula is

\[ F = C_1 \lor C_2 \lor \ldots \lor C_t, \]

where every \( C_i \) is a conjunction of literals.

- Every individual clause \( C_i \) may have positive literals \( x_j \) for some \( j \in [n] \) and negative literals \( \bar{x}_j \) for some \( j \in [n] \).
- In order to satisfy \( C_i \) we need every positive \( x_j \) in \( C_i \) to get the value 1, and every negative literal \( \bar{x}_j \) in \( C_i \) to get the value 0.
- If \( C_i \) contains the literals \( x_j, \bar{x}_j \) for the same \( j \in [n] \) (an opposing pair of literals), there is no assignment in \( \{0, 1\}^n \) which can satisfy clause \( C_i \).
- If \( C_i \) does not contain any opposing pair of literals, then \( C_i \) is satisfied by any assignment \( a \in \{0, 1\}^n \) which sets

\[ a_j = \begin{cases} 1 & C_i \text{ contains the positive literal } x_j \\ 0 & C_i \text{ contains the negative literal } \bar{x}_j \\ 0/1 & \text{neither } x_j \text{ nor } \bar{x}_j \text{ appear in } C_i \end{cases} \]

- Assuming \( C_i \) has \( \ell_i \) literals and no opposing pair, then there are exactly \( 2^{n-\ell_i} \) satisfying assignments for \( C_i \).

FPRAS for DNF counting

Our formula is

\[ F = C_1 \lor C_2 \lor \ldots \lor C_t, \]

where every \( C_i \) is a conjunction of literals.

For every clause \( C_i \), we define \( SC_i \) to be the set of \( 2^{n-\ell_i} \) assignments \( a \in \{0, 1\}^n \) which satisfy \( C_i \). We define

\[ U = \{ (i, a) \mid 1 \leq i \leq t \text{ and } a \in SC_i \}. \]

Notice a few things:

- The \( SC_i \) sets are not disjoint, as a satisfying assignment for one clause may also satisfy a different clause/clauses.
- So usually the number of satisfying assignments \( |\bigcup_{i=1}^t SC_i| \) for \( F \) is strictly less than \( |U| = \sum_{i=1}^t |SC_i| \).
- However, any satisfying assignment can be shared by at most \( t \) clauses, so we also have \( t \cdot |\bigcup_{i=1}^t SC_i| \geq |U| = \sum_{i=1}^t |SC_i| \).
### Sampling DNF satisfying assignments

The $t$-approximate relationship between the cardinalities $\left| \bigcup_{i=1}^t SC_i \right|$ and $\sum_{i=1}^t |SC_i|$ will help us sample.

We will assign "ownership" of a satisfying assignment $a \in \{0,1\}^n$ to the lowest-indexed clause $i$ such that $a \in SC_i$. Let $\overline{SC}_i$ be the set of all $a \in \{0,1\}^n$ assignments that satisfy clause $C_i$ but do not satisfy $C_i$ for any $i', i' < i$.

Then the number of satisfying assignments of $F$ is **exactly** $\sum_{i=1}^t |\overline{SC}_i|$. Also we have

$$\sum_{i=1}^t |\overline{SC}_i| \leq \sum_{i=1}^t |SC_i| \leq t \cdot \sum_{i=1}^t |\overline{SC}_i|.$$  

And also we know the size of each $|SC_i|$ is exactly $2^{n-\ell_i}$; hence we can easily compute the value of $|U| = \sum_{i=1}^t |SC_i|$.

RC (2018/19) – Lecture 17 – slide 6

### FPRAS for DNF counting

**Algorithm APPROX DNF**

1. $count \leftarrow 0$
2. $cardU \leftarrow 0$
3. for $i \leftarrow 1$ to $t$
   4. $cardU \leftarrow cardU + 2^{n-|C_i|}$
5. for $k \leftarrow 1$ to $m$
   6. Choose $i$ with probability $\frac{2^{n-|C_i|}}{cardU}$.
   7. Sample $a \in SC_i$ by setting the literals of $C_i$ to the required values, then randomly generating the other $n-|C_i|$ bits.
8. if $(a$ does not satisfy $C_i$ for any $i' < i$) then
   9. $count \leftarrow count + 1$
10. return $\frac{count}{m} \cdot |cardU|$.

Note $m$ is polynomial in $\frac{1}{\epsilon}, \ln \left( \frac{1}{\delta} \right)$ and the size of the input.

RC (2018/19) – Lecture 17 – slide 7

### Sampling DNF satisfying assignments

If we knew the value of the

$$\frac{\sum_{i=1}^t |\overline{SC}_i|}{\sum_{i=1}^t |SC_i|},$$

then we could just multiply this by the pre-computed $|U|$ to get the exact number of satisfying assignments. But we don't know this.

**Sampling:** We will sample Uniformly at random from $U$ ($m$ times), then check whether each sample also belongs to $\bigcup_{i=1}^t \overline{SC}_i$.

- Can't just choose $i$ uniformly from all the indices $1 \leq i \leq t$.
- Have to weight each $i$ according to the size of $SC_i$, which is the easily computable value $2^{n-\ell_i}$.
- Choose $i$ with probability $\frac{2^{n-\ell_i}}{\sum_{i=1}^t 2^{n-\ell_i}}$, then choose some $a \in SC_i$ (toss $n-\ell_i$ coins). Every element of $U$ is generated with probability $\frac{1}{\sum_{i=1}^t 2^{n-\ell_i}}$.
- Then check whether $a$ belongs to any $\overline{SC}_i$ with $i' < i$.

RC (2018/19) – Lecture 17 – slide 6

### FPRAS for DNF counting

**Theorem (Theorem 11.2)**

Our DNF counting algorithm gives a fully-polynomial randomized approximation scheme for the DNF counting problem if we set $m = \lceil \frac{2 e^{\ln(\frac{1}{\delta})}}{\epsilon^2} \rceil$.

**Proof.**

We will have an FPRAS if we can ensure that the value returned by APPROX DNF lies within $(1 \pm \epsilon)$ of $\frac{\sum_{i=1}^t |SC_i|}{|U|} = \frac{\sum_{i=1}^t |\overline{SC}_i|}{|U|}$.

We know that $\sum_{i=1}^t \frac{|\overline{SC}_i|}{|U|} \geq \frac{1}{t}$. The individual samples ($i$ chosen first, then $a$) are indicator variables with probability $\geq \frac{1}{t}$ of being 1.

Hence, by Chernoff, the probability of being more than $\epsilon$ from the true value after $m$ samples is at most

$$2e^{-e^{-\epsilon^2 m/(3t)}} \leq 2e^{-e^{-\frac{\epsilon^2}{3} \ln(\frac{1}{\delta})}} = 2e^{-\ln(\frac{1}{\delta})} = \delta.$$  

Note $m$ is polynomial in $\frac{1}{\epsilon^2}, \ln(\frac{1}{\delta})$ and the size of the input.  

RC (2018/19) – Lecture 17 – slide 8
From Sampling to Approximate Counting

We have seen an example of how uniform sampling from the target set can be used to obtain an FPRAS to approximately count the elements.

This is generally achievable for structures we want to count/sample (but not usually as straightforward as for DNF).

- Won’t always have an immediately-samplable “superset” like \( U \) whose cardinality is bigger by a low factor like \( T \).
- May need a series of sampling phases.
- Won’t always be able to do exact uniform sampling from the bigger set, that may sometimes be almost-uniform instead.

Definition (Definition 11.3)
Let \( \omega \) be the (random) output of a sampling algorithm for a problem if, given input \( x \) and a parameter \( \epsilon > 0 \), it generates a \( \epsilon \)-uniform sample of \( \Omega \) after running in time polynomial in \( \ln \frac{1}{\epsilon} \) and the size of \( x \).

RC (2018/19) – Lecture 17 – slide 9

From Sampling to Approximate Counting

Imagine that we have an “off the shelf” FPAUS for sampling independent sets of an input graph. We show how to create an FPRAS from this.

Definition
For a given undirected graph \( G = (V, E) \), the subset \( I \subseteq V \) is said to be an independent set if for every \( e \in E, e = (u, v) \), at most one of \( u, v \) lie in \( I \).

Definition
For a given graph \( G = (V, E) \) consider some ordering \( e_1, e_2, \ldots, e_m \) of the edges of \( E \). For every \( i = 1, \ldots, m \), set \( E_i = \bigcup_{j=1}^{i} \{ e_j \} \), and \( G_i = (V, E_i) \). Let \( \Omega(G_i) \) be the number of independent sets in \( G_i \).

Observe that \( G_0 \) is an \( n \)-vertex graph with no edges, and \( G_m \) is \( G \). Each \( G_{i+1} \) is \( G_i \) with an extra edge added.

RC (2018/19) – Lecture 17 – slide 10

From Sampling to Approximate Counting (Independent Sets)

Now consider the following telescoping product on the I.S.s of the different graphs:

\[
|\Omega(G)| = \frac{|\Omega(G_m)|}{|\Omega(G_{m-1})|} \times \frac{|\Omega(G_{m-1})|}{|\Omega(G_{m-2})|} \times \cdots \times \frac{|\Omega(G_1)|}{|\Omega(G_{0})|} \times |\Omega(G_0)|.
\]

- \( |\Omega(G_0)| = 2^n \) as every subset of \( V \) is an I.S. for \( G_0 \) (\( G_0 \) has no edges to worry about).
- We will show how to obtain close approximate values for each ratio \( r_i = \frac{|\Omega(G_i)|}{|\Omega(G_{i-1})|} \), for \( i = 1, \ldots, m \).
- If we write \( \tilde{r}_i \) for our approximation of the ratio \( r_i \), our estimate for the number of I.S.s will be

\[
2^n \prod_{i=1}^{m} \tilde{r}_i.
\]

RC (2018/19) – Lecture 17 – slide 11

From Sampling to Approximate Counting (Independent Sets)

Definition
Let \( \omega \) be the (random) output of a sampling algorithm for a finite sample space \( \Omega \). Then a sampling algorithm is said to generate an \( \epsilon \)-uniform sample of \( \Omega \) if for every \( S \subseteq \Omega \),

\[
|\Pr[\omega \in S] - \frac{|S|}{|\Omega|}| \leq \epsilon.
\]

A sampling algorithm is a fully-polynomial almost uniform sampler (FPAUS) for a problem if, given input \( x \) and a parameter \( \epsilon > 0 \), it generates an \( \epsilon \)-uniform sample of \( \Omega(x) \) after running in time polynomial in \( \ln \frac{1}{\epsilon} \) and the size of \( x \).

RC (2018/19) – Lecture 17 – slide 12
From Sampling to Approximate Counting (Independent Sets)

We will compute a \( \tilde{r}_i \) that is within \( \pm \frac{\epsilon}{2m} \) of the true value with probability at least \( 1 - \frac{\delta}{m} \), for each \( i, 1 \leq i \leq m \).

Our algorithm uses the assumed FPAUS as a subroutine in step 4.

Algorithm \textsc{EstimRatio}(\( G_{i-1} = (V, E_{i-1}); e_i \))

1. \( \text{count} \leftarrow 0 \)
2. \( G_i \leftarrow (V, E_{i-1} \cup \{e_i\}) \)
3. for \( k \leftarrow 1 \) to \( M = \lceil 1296m^2 e^{-2} \ln\left(\frac{2m}{\epsilon}\right) \rceil \)
4. \( \) Generate a \( \frac{\epsilon}{2m} \)-uniform sample from \( \Omega(G_{i-1}) \).
5. \( \) if (the sample generated is also an I.S for \( G_i \)) then
6. \( \text{count} \leftarrow \text{count} + 1 \)
7. \( \text{return} \tilde{r}_i \leftarrow \frac{\text{count}}{M} \)

From Sampling to Approximate Counting (Independent Sets)

\[
\left( 1 - \frac{\epsilon}{2m} \right)^m \leq \prod_{i=1}^{m} \frac{\tilde{r}_i}{r_i} \leq \left( 1 + \frac{\epsilon}{2m} \right)^m .
\]

Easy to show (for \( \epsilon < 1 \)) that \( (1 - \frac{\epsilon}{2m})^m \geq (1 - \epsilon) \) and \( (1 + \frac{\epsilon}{2m})^m \leq (1 + \epsilon) \), hence we have

\[
(1 - \epsilon) \leq \prod_{i=1}^{m} \frac{\tilde{r}_i}{r_i} \leq (1 + \epsilon) ,
\]

\[
(1 - \epsilon) \prod_{i=1}^{m} \tilde{r}_i \leq \prod_{i=1}^{m} r_i \leq (1 + \epsilon) \prod_{i=1}^{m} r_i .
\]

Hence the approximate value \( 2^n \prod_{i=1}^{m} \tilde{r}_i \) computed is within \( (1 \pm \epsilon) \) of the true value with probability \( \geq 1 - \delta \), and we have an FPRAS.

From Sampling to Approximate Counting (Independent Sets)

It is possible to show the following:

Lemma (Lemma 11.4)

When \( m \geq 1 \) and \( 0 < \epsilon \leq 1 \), Algorithm \textsc{EstimRatio} yields a \( \left( \frac{\epsilon}{2m}, \frac{\delta}{m} \right) \)-approximation for the quantity \( r_i \).

Longish proof so not doing in class.

With \( m \) runs of Algorithm \textsc{EstimRatio} (one for each \( \frac{\Omega(G_{i-1})}{|\Omega(G_{i-1})|} \)) we have estimates \( \tilde{r}_m, \tilde{r}_{m-1}, \ldots, \tilde{r}_2, \tilde{r}_1 \).

- By Lemma 11.4, \( \Pr[|\tilde{r}_i - r_i| > \frac{\epsilon}{2m}] \leq \frac{\delta}{m} \), for every \( 1 \leq i \leq m \).
  Hence (Union Bound on bad events) with probability \( 1 - \delta \), all \( \tilde{r}_i \) are within \( \frac{\epsilon}{2m} \) of their true values.

- So with probability \( 1 - \delta \), we have

\[
\left( 1 - \frac{\epsilon}{2m} \right)^m \leq \prod_{i=1}^{m} \frac{\tilde{r}_i}{r_i} \leq \left( 1 + \frac{\epsilon}{2m} \right)^m .
\]

Reading and Doing

Reading:

- Section 11.3 from the book.

Doing:

- Exercise 11.6 from the book.

- Supposed we wanted to come up with a “telescoping product” for the number of contingency tables \( \Sigma_{c,c} \). Can you think of a way of doing this? We need two things:

  - We need the number of ratios (to be approximated) in the sequence to be (smallish) polynomial in the input.
  - We need each ratio to be an inverse polynomial in the size of the input (and preferably not too small).

“Size of input” for c-tables is in terms of \( n, m \) and \( \lg(\max\{\max_i r_i, \max_j c_j\}) \).