Randomness and Computation
or, “Randomized Algorithms”

Mary Cryan

School of Informatics
University of Edinburgh
We’ve already met the concept of a Monte Carlo Algorithm, which uses randomness during its computation to compute a value which is an approximation to the correct answer (satisfying some approximation guarantee, with high probability).

The Monte Carlo Method is a method for estimating values which exploits a relationship between (approximate) counting and (almost-uniform) sampling.
The Monte Carlo Method

Most common scenario for the Monte Carlo Method arises when the value we want to estimate is the count of the number of combinatorial structures satisfying a given criterion.

We will usually rely on a close relationship between the problem of counting the number of combinatorial structures and sampling one of the structures uniformly at random.

Of course, in this setting, “the set of structures" means the set of structures according to some input. With contingency tables this input would be the number of rows $m$, and the number of columns $n$, and the specific lists of row sums $r = (r_1, \ldots, r_m)$ and column sums $c = (c_1, \ldots, c_n)$.

A Markov chain can sometimes be employed to do the sampling.

Other example “count the different combinatorial structures" include the set of proper $k$-colourings (of a given input graph $G = (V, E)$), the number of satisfying assignments (of a given DNF formula $\phi$), etc.
Suppose we live in a world where \( \pi \)'s value is unknown. We estimate:

**Algorithm** \( \text{ESTIMATEPI}(m) \)

1. \( \text{count} \leftarrow 0 \)
2. \( \text{for } i \leftarrow 1 \text{ to } m \)
3. draw \( (X, Y) \) uniformly at random from the square
   
   \( \text{ie draw each of } X, Y \text{ uniformly at random from the continuous distribution on } [-1, 1] \)
4. \( \text{if } X^2 + Y^2 \leq 1 \text{ then} \)
5. \( \text{count} \leftarrow \text{count} + 1 \)
6. \( \text{return } \frac{4 \cdot \text{count}}{m} \)
Monte Carlo Method - cute example

Can let $Z_i$ be the indicator variable for the "$i$-th" $(X, Y)$ lying inside the circle. Then for $Z = \sum_{i=1}^{m} Z_i$,

$$E[Z] = \sum_{i=1}^{m} E[Z_i] = m \frac{\pi \cdot 1^2}{2^2} = \frac{\pi m}{4}.$$ 

Hence $Z' = \frac{4Z}{m}$ is an estimate for the unknown value $\pi$.

Better estimate the higher $m$ is. By Chernoff (4.6) if we have $m$ samples, then for arbitrary $\epsilon \in (0, 1)$,

$$\Pr[|Z' - E[Z']| \geq \epsilon \pi] = \Pr \left[ |Z - \frac{\pi m}{4} | \geq \frac{\epsilon \pi m}{4} \right]$$

$$= \Pr[Z - E[Z] \geq \epsilon E[Z]]$$

$$\leq 2e^{-\epsilon^2 \pi m/12}.$$ 

RC (2016/17) – Lecture 16 – slide 5
Monte Carlo Method - cute example

Definition (Definition 10.1)
A randomized algorithm for estimating a (positive) quantity $V$ (usually depending on certain input parameters) is said to give an $(\epsilon, \delta)$ approximation if its output $X$ satisfies

$$\Pr[|X - V| \leq \epsilon V] \geq 1 - \delta.$$ 

We know that for given $\epsilon \in (0, 1)$, that if we take $m$ samples, then Algorithm ESTIMATEPI gives an

$$(\epsilon, 2e^{-e^{2}\pi m/12})$$

approximation.

We need $2e^{-e^{2}\pi m/12} \leq \delta$, equivalent to having $e^{-e^{2}\pi m/12} \leq \frac{\delta}{2}$, equivalent to having $\frac{e^{2}\pi m}{12} \geq \ln(\frac{2}{\delta})$, equivalent to $m \geq \frac{12 \ln(\frac{2}{\delta})}{\pi e^{2}}$. 

RC (2016/17) – Lecture 16 – slide 6
Monte Carlo Method

**Theorem (Theorem 10.1)**

Let $X_1, \ldots, X_m$ be independent and identically distributed indicator random variables (i.e., Bernoulli with a fixed parameter), and $\mu = \sum_{i=1}^{m} E[X_i]$. Then if $m \geq \frac{3 \ln \left( \frac{2}{\delta} \right)}{\epsilon^2 \mu}$, we have

$$\Pr \left( \left| \frac{1}{m} \sum_{i=1}^{m} X_i - \mu \right| \geq \epsilon \mu \right) \leq \delta.$$

So for this $m$, sampling gives a $(\epsilon, \delta)$-approximation of $\mu$.

**Definition (Definition 10.2)**

A fully polynomial randomized approximation scheme (FPRAS) for a problem is a randomized algorithm for which, given an input $x$ and any parameters $\epsilon, \delta$ with $0 < \epsilon, \delta < 1$ the algorithm outputs an $(\epsilon, \delta)$-approximation to the true value $V(x)$ in time polynomial in $\frac{1}{\epsilon}$, in $\ln \left( \frac{1}{\delta} \right)$ and in the size of $x$. 
Monte Carlo Method

*The Monte Carlo Method involves taking a sequence of independent and identical samples* $X_1, \ldots, X_m$ *such that* $\mathbb{E}[X_i] = V$, *with* $m$ *set large enough (see Theorem 10.1) to guarantee we have an* $(\epsilon, \delta)$-*approximation.*

The book discusses the reasons for using the Monte Carlo method. They discuss the situation of wanting to find “approximate” solutions for computational problems which are NP-hard to solve exactly (don’t believe that NP-hard problems have polynomial-time algorithms).

More common in fact is the use of Monte Carlo in approximating the “count” of $\#P$-complete (“hard to count exactly in polynomial time”) problems like proper $k$-colourings, contingency tables, etc. These will be from situations where the *decision problem* (“finding one”) is polynomial-time.
The DNF counting problem

An alternative normal form for propositional logical formulae is Disjunctive Normal Form (DNF), where each clause is now a conjunction ($\land$) or literals, and we have disjunctions ($\lor$) at the top-level. For example:

$$(x_1 \land \neg x_2 \land x_3) \lor (x_2 \land x_4) \lor (\neg x_1 \land x_3 \land x_4).$$

We are interested in counting the number of satisfying assignments to a given DNF formula.

- It is NP-hard to compute the exact number of satisfying assignments for a DNF, as this would solve the (NP-hard) problem of SAT (we can design a DNF which represents the negation of the SAT formula $\phi$, and has $2^n$ satisfying assignments $\Leftrightarrow$ $\phi$ was satisfiable).

- Hence counting DNF assignments is $\#P$-complete.

- However, a DNF usually has some/many satisfying assignments, and we aim to approximately count.
The DNF counting problem - Naïve Approach

For a given DNF formula $F$ over $n$ variables, let $c(F)$ denote the number of satisfying assignments to $x_1, \ldots, x_n$. 

$c(F)$ will be 0 only if it is the case that every clause contains $x_i$ and $\bar{x}_i$ for some $i$. Easy to notice this before we start. We eliminate any of these definitely unsatisfiable clauses before we start.

Naïve approach to counting DNF assignments is to sample $m$ uniform random assignments to $x_1, \ldots, x_n$ (from the set $\{0, 1\}^n$) and check whether $F$ is satisfied for each sample. The random variable $X_i$ will be 1 if the $i$-th trial satisfies $F$, 0 otherwise). Then we estimate the fraction of these to satisfy $F$ as $\frac{\sum_{i=1}^{m} X_i}{m}$, then we return

$$2^n \frac{\sum_{i=1}^{m} X_i}{m}$$

as the estimate of the total number of satisfying assignments.
The DNF counting problem - Naïve Approach

In order for
\[ 2^n \sum_{i=1}^{m} X_i \]
to be an \((\epsilon, \delta)\) - approximation for \(c(F)\), we require that we have

\[ \left| 2^n \sum_{i=1}^{m} X_i - c(F) \right| \leq \epsilon \cdot c(F) \quad \text{which happens } \Leftrightarrow \]
\[ \left| \sum_{i=1}^{m} X_i - \frac{mc(F)}{2^n} \right| \leq \epsilon \cdot \frac{mc(F)}{2^n} \]

and by Chernoff this holds \(\Leftrightarrow\) we have

\[ m \geq \frac{3 \cdot 2^n \ln \left( \frac{2}{\delta} \right)}{\epsilon^2 c(F)} \]

But if it is the case that \(c(F)\) is much much smaller than \(2^n\), then we need a huge number of samples (logical . . . needle in haystack).
The DNF counting problem

Problem with using the Naïve Monte Carlo method is that it is infeasible (for any application) 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.

On Friday, we will see a Monte Carlo algorithm which incorporates knowledge about “satisfying assignments per clause" to give an FPRAS for DNF.
Reading and Doing

Reading

▶ Sections 10.1, 10.2 from the book.
▶ We will continue with Section 4.2.2, 4.3 on Friday.

Doing

▶ Exercises 10.3, 10.4 from the book.