Monte-Carlo Approximation Algorithms for Enumeration Problems

Karp, Luby and Madras (1989)
Outline

• DNF Counting problem
• A naïve Monte-Carlo algorithm (Algorithm 1)
• Generalization to the Union of Sets problem and Coverage Algorithm
• Coverage Algorithm for DNF Counting (Algorithm 2)
• Self-Adjusting Coverage Algorithm (Algorithm 3)
DNF Counting Problem

• **Input**: Propositional formula $F$, a disjunction of $m$ clauses $C_1, \ldots, C_m$

• Each $C_i$ is a conjunction of literals w.r.t $n$ Boolean variables $X_1, \ldots, X_n$

• **Output**: $\#F$ – number of truth assignments over $n$ variables which satisfy the formula $F$

• **#P-Complete** problem (i.e. if $P \neq NP$, then there is no poly-time algorithm for DNF counting problem)
\( \varepsilon, \delta \) – Approximation Algorithm

We consider the problem of approximating \( \#F \). An \( \varepsilon, \delta \) approximation algorithm for the DNF problem is a Monte-Carlo algorithm which on every input formula \( F, \varepsilon > 0, \delta > 0 \), outputs a number \( \tilde{Y} \) such that

\[
\Pr\left( (1 - \varepsilon) \#F \leq \tilde{Y} \leq (1 + \varepsilon) \#F \right) \geq 1 - \delta.
\]

The goal is to design an approximation algorithm for the DNF counting problem which runs in time polynomial in the length of \( F \).
\((\varepsilon, \delta)\)-FPRAS

**Definition 11.2:** A polynomial randomized approximation scheme (PRAS) for a counting problem \(\Pi\) is a randomized algorithm \(A\) that takes an input instance \(I\) and a real number \(\varepsilon > 0\), and in time polynomial in \(n = |I|\) produces an output \(A(I)\) such that

\[
\Pr \left[ (1 - \varepsilon)\#(I) \leq A(I) \leq (1 + \varepsilon)\#(I) \right] \geq \frac{3}{4}.
\]

A fully polynomial randomized approximation scheme (FPRAS) is a polynomial randomized approximation scheme whose running time is polynomially bounded in both \(n\) and \(1/\varepsilon\).

**Definition 11.3:** An \((\varepsilon, \delta)\)-FPRAS for a counting problem \(\Pi\) is a fully polynomial randomized approximation scheme that takes an input instance \(I\) and computes an \(\varepsilon\)-approximation to \(\#(I)\) with probability at least \(1 - \delta\) in time polynomial in \(n, 1/\varepsilon, \) and \(\log 1/\delta\).
Abstract Monte-Carlo algorithm

- Given finite set of known size $U$
- **Efficient** method for sampling uniformly from $U$
- Given $f: U \to \{0, 1\}$ and efficient method for computing $f(u)$, given $u$ in $U$
- $G = f^{-1}(1)$ (pre-image of 1 in $U$)
- Goal is to estimate $|G|$
- **One trial**: randomly choose $u$ in $U$, compute $f(u)$ and set $Y = |U| \cdot f(u)$
- Monte-Carlo algorithms consists in running $N$ trials
- $Y_i$ – value of $Y$ from $i$-th trial
- Output is $\hat{Y} \leftarrow \frac{\sum_{i=1}^{N} Y_i}{N}$
How large should $N$ be?

- We have $E[Y] = |G|$, so also $E[\tilde{Y}] = |G|$
- Need to choose value of $N$, to guarantee that the Monte-Carlo algorithm presented is an $\epsilon, \delta$ approximation algorithm

**Zero–One Estimator Theorem.** Let $\mu = |G|/|U|$. Let $\epsilon \leq 2$. If $N \geq (1/\mu) \cdot (4 \ln(2/\delta)/\epsilon^2)$ then the Monte-Carlo algorithm described above is an $\epsilon, \delta$ approximation algorithm.
Sufficient properties for Monte-Carlo trial

• For DNF Counting problem, in order to obtain an \( \varepsilon, \delta \) approximation algorithm, need the following sufficient properties to design a Monte-Carlo trial. Given formula \( F \), polynomial (time) means polynomial in the length of \( F \).

1. In polynomial time we can compute \( |U| \) and we can randomly choose members from \( U \) with the uniform probability distribution.

2. The function \( f \) is computable in polynomial time.

3. We can compute in polynomial time an upper bound \( B \) on \( |U|/|G| \) such that the value of \( B \) is polynomial.

If we can achieve these goals, then the polynomial time \( \varepsilon, \delta \) approximation algorithm consists of computing \( B \) and running \( N = B \cdot 4\ln(2/\delta)/\varepsilon^2 \) trials.
Algorithm 1 – Naïve Monte-Carlo

• **U** – the set of all possible truth assignments for the **n** variables
• **f** – the function that evaluates to 1 on the set of truth assignments that satisfy the formula **F**, 0 otherwise.

So, **G** is the set of truth assignments that satisfy **F**

• Want to estimate |**G**| = #**F**

• We notice that properties 1 and 2 in previous slide are true for this definition of a trial

• Property 3 not true, since |**U**| / |**G**| may be exponentially large in the length of **F**
Union of Sets problem – Coverage Algorithm

• Generalization of DNF Counting
• **Input**: Description of \( m \) sets, \( D_1, ..., D_m \)
• **Output**: \( |D| \), where \( D \) is the **union** of \( D_1, ..., D_m \)
• We design an \( \varepsilon, \delta \) approximation algorithm to estimate \( |D| \)
• We make the following assumptions:

1. For all \( i \) between 1 and \( m \), \( |D_i| \) can be easily computed.
2. For all \( i \) between 1 and \( m \), we can randomly choose an element \( s \in D_i \) such that the probability of choosing each such \( s \) is \( 1/|D_i| \).
3. Given any \( s \in D \) and any \( i \) between 1 and \( m \), it is easy to decide whether or not \( s \in D_i \).
Abstract Coverage Algorithm – Trial design

The following is a description of one trial of the algorithm. Let $U$ be the direct sum of the $D_i$, i.e., $U = D_1 \oplus \cdots \oplus D_m$. An element in $U$ is represented by a pair $(s, i)$, where $i$ is between 1 and $m$ and $s \in D_i$. Thus, $|U| = \sum_{i=1}^{m} |D_i|$. For each $s \in D$, define the coverage set to be

$$\text{cov}(s) = \{(s, i) : (s, i) \in U\}.$$

There is one element in $\text{cov}(s)$ for each set $D_i$ containing $s$. The coverage sets define a partition on $U$, where the size of each coverage set is at most $m$ and the number of coverage sets is exactly $|D|$. We define $f(s, i) = 1$ if $i$ is the smallest index such that $s \in D_i$ and $f(s, i) = 0$ otherwise. Let $G$ be the subset of $U$ for which $f$ takes on the value of 1. Since, for each $s \in D$, $f$ takes on the value 1 for exactly one element of $\text{cov}(s)$, $|G| = |D|$. 
Abstract Coverage Algorithm – One Trial

A trial proceeds as described in Section 1: choose \((s, i)\) at random from \(U\) (uniformly) and set \(Y \leftarrow |U| \cdot f(s, i)\). In more detail:

1. Randomly choose \(i \in \{1, \ldots, m\}\) such that \(i\) is chosen with probability \(|D_i|/|U|\).

2. Randomly choose \(s \in D_i\) such that \(s\) is chosen with probability \(1/|D_i|\).

Note. Steps 1 and 2 randomly choose \((s, i) \in U\) with probability \(1/|U|\).

3. Compute \(f(s, i)\); \(Y \leftarrow f(s, i) \cdot |U|\).
Abstract Coverage Algorithm - Complexity

advantage of the coverage algorithm over the naive algorithm is that \( m \geq 1/\mu \) (where \( \mu = |G|/|U| \)), and thus, applying the Zero–One Estimator Theorem, \( N = m \cdot 4 \ln(2/\delta)/\epsilon^2 \) trials suffice for an \( \epsilon, \delta \) approximation algorithm.
Algorithm 2
Coverage Algorithm for DNF Counting

- $D_i$ – set of truth assignments which satisfy clause $C_i$, $i = 1,\ldots,m$
- $D$ – union of $D_i$’s
- $\#F = |D|$
- Given $s$ in $D$, $\text{cov}(s)$ contains an element for each clause which $s$ satisfies
EXAMPLE 1. Let $F = C_1 \lor C_2 \lor C_3$, where $C_1 = X_1 \land X_2$, $C_2 = \overline{X_1}$ and $C_3 = \overline{X_3}$. Consider the following matrix. The rows of the matrix correspond to the elements of $D$ and the columns correspond to the $m$ clauses. A box in entry $(s, i)$ in this matrix indicates that $(s, i) \in U$. The set of boxes in row $s$ correspond to the elements in $\text{cov}(s)$. For each $(s, i) \in U$, if $f(s, i) = 1$ then the corresponding box is filled in and if $f(s, i) = 0$ then the corresponding box is left blank (Fig. 1).
To simplify the discussion of the implementation, we assume a simple representation of the formula $F$ which is not as compact as it could be. We leave it to the reader to derive a more compact representation when each clause contains a small number of the possible literals. $F$ is represented by an $m$ by $n$ matrix, where the $i, j$ entry in the matrix is 1 if literal $X_j$ appears in clause $i$, 0 if literal $\overline{X_j}$ appears in clause $i$, and empty otherwise.

The preprocessing for the coverage algorithm consists of the following:

For $i = 1, \ldots, m$, compute $|D_i| = 2^n - \# \text{ literals in } C_i$. Let $A$ be an integer array of length $m + 1$ such that $A_0 = 0$ and $A_i = \Sigma_{j=1}^i |D_i|$. Then $|U| = A_m$. The total time for the preprocessing step is $O(mn)$. 
Algorithm 2 – Trial definition

We run the algorithm for $N = m \cdot 4 \ln(2/\delta)/\varepsilon^2$ trials. A trial of the algorithm consists of the following steps:

1. Randomly choose $i \in \{1, \ldots, m\}$ such that $i$ is chosen with probability $|D_i|/|U|$. This is done by randomly choosing a number $r$ between $1$ and $|U|$ and using binary search to find the entry in $A$ such that $A_{i-1} < r \leq A_i$.

2. Randomly choose $s \in D_i$ such that $s$ is chosen with probability $1/|D_i|$. This is done by setting the truth assignments for the variables which appear in $C_i$ to satisfy clause $C_i$ and then choosing the truth values for the variables that do not appear in $C_i$ randomly to be “true” or “false” each with probability $\frac{1}{2}$.

(Note. Steps 1 and 2 randomly choose $(s, i) \in U$ with probability $1/|U|$.)
Algorithm 2 – Trial definition and Complexity

3. Compute \( f(s, i) \). Let \( j = \min\{ l : s \in D_l \} \). Checking whether \( s \in D_l \) consists of checking whether \( s \) satisfies \( C_j \). The value of \( j \) is computed by indexing sequentially through the clauses. Then, \( f(s, i) = 1 \) iff \( i = j \).

4. \( Y = f(s, i) \cdot |U| \).

The total time for steps 1 and 2 of a trial is \( O(n) \). The bottleneck in terms of time is step 3, the computation of \( f(s, i) \). For each clause \( C_l \), the time to check if \( s \) satisfies \( C_l \) is \( O(n) \). The total number of clauses which are checked in the worst case is \( m \). Thus, the total time for step 3 is \( O(mn) \). The total time for preprocessing plus all trials of the algorithm is \( O(nm^2 \cdot \ln(1/\delta)/\varepsilon^2) \).
Algorithm 3
Self-adjusting coverage algorithm

- $E[Y] = D$ if $f$ is any function which satisfies the following for each $s$ in $D$.

$$\sum_{(s, i) \in \text{cov}(s)} f(s, i) = 1.$$ 

We first describe a variant of the coverage algorithm for the union of sets problem, using random variables in place of $f$. For each $s \in D$, $f'(s)$ is a random variable such that $E[f'(s)] = 1/|\text{cov}(s)|$. A trial consists of randomly choosing $(s, i) \in U$, choosing a random value for $f'(s)$ and setting the estimator $Y$ to $|U| \cdot f'(s)$. It can be easily verified that $E[Y] = |D|$. 
Algorithm 3 – Computing $f'(s)$

For each $s \in D$, not only is $f'(s)$ a random variable, but also the amount of time to compute $f'(s)$ is a random variable $t(s)$. Given $s \in D$, $f'(s)$ is computed as follows:

```plaintext
    computing $f'(s)$
    $t(s) \leftarrow 0$
    repeat
        $t(s) \leftarrow t(s) + 1$
        randomly choose $j \in \{1, \ldots, m\}$ with probability $1/m$
    until $s \in D_j$
    $f'(s) \leftarrow t(s)/m$
```

At each iteration of the repeat loop, the probability that $s \in D_j$ is $|\text{cov}(s)|/m$. Thus, $t(s)$ is a geometric random variable and $E[t(s)] = m/|\text{cov}(s)|$. Thus, $E[f'(s)] = 1/|\text{cov}(s)|$. 
SELF-ADJUSTING COVERAGE ALGORITHM.

\[ \text{gtime} \leftarrow 0 \quad \{ \text{gtime counts the global number of steps executed} \} \]
\[ \text{TOTAL} \leftarrow 0 \]
\[ N_T \leftarrow 0 \]
\[ T \] is set as specified in Theorem I or Theorem II

\textbf{trial:}

randomly choose \((s, i) \in U\) with probability \(1/|U|\) \{as before\}
\[ t(s) \leftarrow 0 \]

\textbf{step:}

\[ t(s) \leftarrow t(s) + 1 \]
\[ \text{gtime} \leftarrow \text{gtime} + 1 \]

\textbf{If} \( \text{gtime} > T \) \textbf{then go to finish}

randomly choose \(j \in \{1, \ldots, m\}\) with probability \(1/m\)

\textbf{If} not \(s \in D_j\) \textbf{then go to step}

\[ \text{TOTAL} \leftarrow \text{gtime} \]
\[ N_T \leftarrow N_T + 1 \]
\[ f'(s) \leftarrow t(s)/m \]
\[ Y_{N_T} \leftarrow |U| \cdot f'(s) \]

\textbf{Go to trial}

\textbf{finish:}

\[ \bar{Y} \leftarrow \Sigma_{i=1}^{N_T} Y_i/N_T \]

(Equivalently, \( \bar{Y} \leftarrow (\text{TOTAL} \cdot |U|)/(m \cdot N_T) \))
\[ \bar{Y}' \leftarrow T \cdot |U|/m \cdot N_T \]
Algorithm 3 – Definition of single step

When we use this implementation of a trial of the coverage algorithm, the time per trial is dominated by the time spent executing the repeat loop in the computation of $f'$. In the rest of the discussion, one step is defined to be a single iteration of this repeat loop. The total running time of the algorithm is then the number of steps times the time it takes to execute a step. An upper bound on the time to execute a step of the algorithm is the time to randomly choose $(s, i) \in U$ (this is executed once for each trial of the algorithm, but an upper bound on the number of trials is the total number of steps) plus the time to randomly choose $j \in \{1, \ldots, m\}$ plus the time to determine if $s \in D_j$. 
Algorithm 3 – Derivation

Let $\mu = \frac{|D|}{|U|}$ and let $t$ be the random variable which is defined to be the number of steps in a trial. Then,

$$E[t] = \sum_{(s, i) \in U} \frac{E[t(s)]}{|U|} = \frac{m \cdot |D|}{|U|} = m \cdot \mu.$$  

An upper bound on $E[t]$ is $m$. We can prove a theorem analogous to the Zero–One Estimator Theorem which states that this variant of the coverage algorithm is an $\epsilon, \delta$ approximation algorithm when the number of trials is $O[m \cdot \ln(1/\delta)/\epsilon^2]$. However, this does not improve the total running time of the algorithm since the upper bound on the expected number of steps per trial is $m$. For instance, if this variant of the coverage algorithm is used for the DNF counting problem, the total running time of the entire algorithm is $O[nm^2 \cdot \ln(1/\delta)/\epsilon^2]$, which is no better than the time for the original coverage algorithm.
Algorithm 3 - Derivation

There is a nice tradeoff which we could take advantage of if we could compute $\mu$. Let $c$ be a constant and let

$$N(\mu) = \frac{c \cdot \ln(1/\delta)}{\mu \cdot \epsilon^2}.$$ 

For the coverage algorithm we showed that $N(\mu)$ trials are sufficient to guarantee an $\epsilon, \delta$ approximation algorithm for a small value of $c$. Let $T(\mu)$ be the random variable which is the total number of steps completed by the variant of the coverage algorithm when we execute $N(\mu)$ trials. Then,

$$E[T(\mu)] = N(\mu) \cdot E[t] = \frac{cm \cdot \ln(1/\delta)}{\epsilon^2}$$

and thus the expected number of steps for this algorithm is $m$ times less than the upper bound on the number of steps for the variant of the coverage algorithm described above. The problem is that we cannot implement this algorithm because it requires the computation of $\mu$ which is the quantity we are trying to estimate.
We overcome our inability to implement the algorithm which depends on the computation of $\mu$ by the following trick. Let $T = cm \ln(2/\delta)/\epsilon^2$, where $c$ is a constant to be specified later. We run the algorithm for $T$ steps and let the number of trials completed during these $T$ steps be a random variable $N_T$. Let $\text{TOTAL} \leq T$ be the total number of steps completed in the first $N_T$ trials. The average value of the estimates from the $N_T$ trials is exactly $\tilde{Y} = (\text{TOTAL} \cdot |U|)/(m \cdot N_T)$. Another very similar estimator of $|D|$ is $\tilde{Y}' = (T \cdot |U|)/(m \cdot N_T)$. In the Appendix we prove that for an appropriate choice of $c$ the algorithm is an $\epsilon, \delta$ approximation algorithm when using $\tilde{Y}$ or $\tilde{Y}'$ as the estimator. It turns out that the proof
Algorithm 3 - Derivation

appropriate choice of $c$ the algorithm is an $\epsilon, \delta$ approximation algorithm when using $\tilde{Y}$ or $\tilde{Y}'$ as the estimator. It turns out that the proof for $\tilde{Y}'$ is cleaner and simpler than the proof for $\tilde{Y}$. The intuition for why the algorithm is an $\epsilon, \delta$ approximation algorithm when using estimator $\tilde{Y}$ follows. The expected value of $N_T$ is approximately $T/E[t] = c \ln(2/\delta)/\mu \epsilon^2 = N(\mu)$. Thus, the intuition is that by fixing $T$, the number of trials completed by the algorithm self-adjusts so that with high probability enough trials are run to guarantee an $\epsilon, \delta$ approximation algorithm.
Algorithm 3- Theorems and Complexity

**Self-adjusting Coverage Algorithm Theorem I.** When $\varepsilon < 1$ and 

\[ T = \frac{(8 \cdot (1 + \varepsilon) \cdot m \ln(2/\delta))/\varepsilon^2}{}, \] 

the self-adjusting coverage algorithm is an $\varepsilon, \delta$ approximation algorithm when estimator $\tilde{Y}'$ is used.

**Self-adjusting Coverage Algorithm Theorem II.** When $\varepsilon < 1$ and 

\[ T = \frac{(8 \cdot (1 + \varepsilon) \cdot m \ln(3/\delta))/(1 - \varepsilon^2/8)\varepsilon^2}{}, \] 

the self-adjusting coverage algorithm is an $\varepsilon, \delta$ approximation algorithm when estimator $\tilde{Y}$ is used.

The time per step is dominated by the time to determine if truth assignment $s$ satisfies clause $C_j$; this can be executed in $O(n)$ time. Thus, the total running time of the self-adjusting coverage algorithm for the DNF counting problem is $O[nm \cdot \ln(2/\delta)/\varepsilon^2]$. 