COMPSCI 630: Randomized Algorithms April 9, 2018

Lecture 21: Approximate Counting I – DNF counting

Lecturer: Rong Ge *Scribe: Abe Frandsen*

1 Overview

In this lecture, we discuss a major application of Markov chains and random walks, namely approximate counting. We start off by defining the problem and giving some examples. Then we address some complexity issues and outline a FPRAS for approximate counting. Finally, we consider the problem of approximate DNF counting.

2 Introduction

A *counting problem* asks to find the number of elements in a specified set, rather than finding the best element (optimization problem) or determining if there exists an element (decision problem).

Example 1. Given a graph, count the number of matchings (or spanning trees, paths, etc.). If we can count the number of matchings, we can compute

$$Pr(\text{edge } e \text{ is part of a uniformly random matching}) = \frac{\#\text{ matchings with edge } e \text{ deleted}}{\#\text{ matchings in graph}},$$

and this can tell us how important an edge is.

Example 2. Given an object $S \subset \mathbb{R}^d$, compute the volume of S. We can turn this into a counting problem by, for example, creating a grid, and counting how many grid points are in S, which allows us to estimate its volume.

Example 3. Given a complicated distribution, compute marginals and expectations.

In general, we can think of a counting problem as computing

$$\sum_{i \in S} f(i) \qquad \left(\text{or } \int f(x)\,dx\right)$$

where $f(i)$ can be efficiently computed, but S may be very large, i.e. exponential in the relevant parameters.

3 Complexity of Counting

Definition 1 (#P (Sharp-P))**.** *#P is the set of counting problems where verifying a solution (i.e. computing $f(i)$) takes polynomial time.*

Observe that NP reduces to #P, since for NP problems, we just need to verify if there is or isn't a solution. It is believed that #P is strictly harder than NP. Below we list some examples of the complexity of decision problems versus counting problems.

Example	Decision	Counting
Spanning trees	P	P
bipartite matchings	P	#P-complete
3-SAT	NP-complete	#P-complete

4 Approximate Counting

Definition 2 (PRAS,FPRAS)**.** *A* polynomial-time randomized approximation scheme (PRAS) *is an algorithm that, given any* $\varepsilon > 0$ *and counting problem* I*, outputs* $A(I)$ *in polynomial-time (in the size of the problem) such that with probability at least* $3/4$

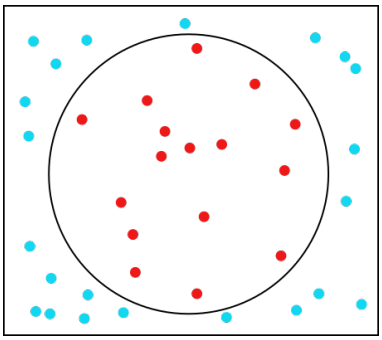
$$(1-\varepsilon)\#I \leq A(I) \leq (1+\varepsilon)\#I,$$

where $\#I$ *is the correct answer to the counting problem.*

A fully polynomial-time randomized approximation scheme (FPRAS) *is an algorithm that has the same guarantees as a PRAS, but runs in time polynomial in both the size of the problem and in* ε^{-1}.

Using the notion of Monte-Carlo sampling, we can outline a general idea for an FPRAS. To describe this approach, consider the classical problem:

Example 4. (Area of circle) Suppose we want to compute the area of a circle (we aren't allowed to use the well-known formula). We can put a bounding box around the cirle, then sample uniformly in the box (which is easy to do, since it is a product distribution). For each sampled point, we can quickly check whether it is in the circle or not by computing it's distance to the center. Finally, we return the fraction of points that lie in the circle.



In general, we describe this technique as follows:

- generate a sample i from S (i.e. the bounding box in our example)
- compute $f(i)$ (i.e. $f(i) = 1$ if i is in the circle in our example)
- estimate the average of $f(i)$ over the samples

However, there are some problems with this Monte-Carlo approach:

1. how do we choose S, and how do we sample from S? (Often, Markov chains are needed for this sampling.)

2. The estimate for the average of $f(i)$ may be very high variance. For example, if $f(i) \in \{0,1\}$, and $Pr(f(i)=1)$ is very small, then small additive error in the estimate for the average of $f(i)$ unfortunately can translate to large multiplicative error. In our previous example about the area of a circle, this could happen if the bounding box is much larger than the circle. On possible remedy to this issue is to decompose $Pr(f(i)=1)$ as a product of several probabilities, none of which are small, and estimate these component probabilities.

5 DNF Counting

We now study the particular problem of *DNF counting*. A *disjuntive normal form*, or DNF, formula is a Boolean formula that is structured as an OR of clauses $C_1 \vee C_2 \vee \cdots \vee C_m$, where each clause C_i is an AND of literals $C_i := L_{i,1} \wedge L_{i,2} \wedge \cdots \wedge L_{i,r_i}$, and a literal is either a variable or its negation (i.e. x_j or $\bar{x_j}$ for $1 \leq j \leq n$).

Example 5. The following is a DNF:

$$(x_1 \wedge \bar{x_3}) \vee (x_2 \wedge x_3 \wedge \bar{x_5}) \vee (x_4 \wedge x_5)$$

Observe that the entire DNF is true if just one of the clauses is true.

The *DNF Counting Problem* asks to count the number of satisfying assignments of a given DNF. For the rest of this lecture, let $\#X$ denote the solution to the DNF counting problem we are considering. Note that there are 2^n possible assignments to n Boolean variables. The exact counting problem is NP-complete, since if we could exactly count the number of satisfying assignments, we could exactly answer the question of whether the *negation* of the DNF has a satisfying assignment. But the negation of a DNF is a CNF, and it is well known that CNF satisfiability is NP-complete (c.f. 3-SAT). Despite this hardness result, we note that an FPRAS can't distinguish between 2^n vs $2^n - 1$ solutions (since that would require ε to be exponentially small), and so we can still hope for a FPRAS for this problem.

Consider the following **naive algorithm**: randomly generate an assignment $X \in \{0,1\}^n$, and check whether it satisfies the DNF. Do this multiple times to estimate $Pr(X \text{ satisfies the DNF})$, and then output $2^n \cdot Pr(X \text{ satisfies the DNF})$. As noted earlier, this basic Monte-Carlo approach will fail if $Pr(X \text{ satisfies the DNF})$ is very small.

Key idea: we will change the object of counting. We now count pairs (X,i), where X is a satisfying assignment to clause C_i in the DNF. Note that each X which satisfies the DNF will necessarily contribute at least one count to this new counting problem. Counting the number of such pairs $\#(X,i)$ is straightforward, since

$$\begin{aligned}\#(X,i) &= \sum_{j=1}^{m} (\#\text{ satisfying assignments for } C_j) \\ &= \sum_{j=1}^{m} 2^{n-r_j}\end{aligned}$$

(The summand 2^{n-r_j} occurs because for clause j, r_j variables must be fixed, leaving $n - r_j$ free variables.) Sampling a uniformly random such pair is also easy:

- first sample and index j with probability $2^{n-r_j}/\#(X,i)$
- next, sample a random satisfying assignment for C_j (by simply randomly sampling each of the free variables).

How can we use these ideas to solve the DNF counting problem? We know that $\#X \leq \#(X,i)$, but the right-hand-side can be much larger. We need some way to "remove" duplicates from $\#(X,i)$, but we can't do this by simply listing every pair, since there are exponentially many. Instead, we focus on a particular type of pair:

Definition 3. *We say the assignment-clause pair (X,i) is a* canonical pair *if i is the* first *index such that X satisfies clause C_i.*

Since each satisfying assignment can be mapped to a unique canonical pair, we have that

$$\#\text{canonical pairs} = \#X.$$

Furthermore, since there are only m clauses, any given satisfying assignment can contribute at most m pairs to $\#(X,j)$. Hence, we also have

$$\#\text{canonical pairs} \geq \frac{\#(X,i)}{m}.$$

With these observations in mind, we have the following **algorithm**: randomly generate pairs (X,j), estimate the fraction $\hat{p}$ of pairs that are canonical, and output $\#(X,i) \cdot \hat{p}$.

Claim 1. *Let p be the true fraction of pairs that are canonical. If we can estimate $\hat{p}$ with an accuracy satisfying $|\hat{p}-p| \leq \frac{\varepsilon}{m}$, then we have*

$$\frac{|\hat{p}-p|}{p} \leq \varepsilon.$$

Proof. This follows from the fact that $p \geq \frac{1}{m}$, as noted earlier. □

You can check that this claim gives the required accuracy guarantee for a FPRAS.