15-859(M): Randomized Algorithms **Lecturer:** Shuchi Chawla
Topic: Sampling, Medians of Means method and DNF counting **Date:** October 6, 2004
Scribe: Florin Oprea

8.1 Introduction

In this lecture we will consider the problem of counting the number of solutions of a problem in NP. This is obviously NP-hard for problems that are NP-complete, but not necessarily easy for problems in P. The class of such counting problems is denoted by $\#P$.

Definition 8.1.1 *The class $\#P$ (called "Sharp-P") is the class of all functions f for which there exists an NP language L with a non-deterministic Turing machine M such that $f(x)$ equals the number of accepting paths in M on input x.*

Counting problems have several interesting applications: estimating the volume of a body in space, counting the number of perfect matchings of a graph, counting the number of satisfying assignments for a DNF formula, approximating the permanent of a matrix. In this lecture we will look at the last two.

A counting problem, in its most general form, can be described as follows: Given a universe U and a set $S \subset U$, determine the size of S. When it is easy to determine the size of U, then this problem translates to estimating the ratio $\frac{|S|}{|U|}$ in an efficient way. For functions in the class $\#P$, we can think of the universe U as being the set of all solutions to the problem, or the set of all accepting paths.

Example: #SAT is a canonical problem in #P (i.e., it is #P -complete). A SAT formula is a formula in CNF form: $(x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_4 \vee \bar{x_1} \vee x_5) \wedge \ldots$. #SAT estimates the number of satisfying assignments for a SAT formula.

8.2 The DNF Counting Problem

A DNF formula is a conjunction of clauses, each of which is a disjunction of literals of the form x or $\bar{x}$. An example is: $(x_1 \wedge x_2 \wedge \bar{x_3}) \vee \ldots$. Our goal is to estimate the number of assignments to the variables that satisfy this formula.

Note that it is easy to determine whether a DNF formula is satisfiable or not — the formula is unsatisfiable if and only if each clause in it contains a contradiction. We will now show the surprising result that #DNF is in fact $\#P$ complete. This indicates that counting can be much harder than the corresponding decision problem.

Theorem 8.2.1 *#DNF is $\#P$ complete*

Proof: We reduce #SAT to this problem. Consider a CNF formular f of polynomial size. Let $g = \bar{f}$. Note that g can be expressed in DNF form with the same size as f.[1] Now any assignment

[1]Think about why this is true.

that satisfies f does not satisfy g and vice versa. Thus we have $2^n - \#g = \#f$, and solving $\#g$ gives an answer to $\#f$, which is a $\#P$ complete problem. ■

Although we do not know how to compute #DNF in polynomial time, our aim will be to estimate it to within a reasonable accuracy. In order to do so, our first attempt is to use sampling. Let U be the universe of possible assignments for a DNF formula f, and $S \subseteq U$ the set of satisfying assignments. The algorithm is the following:

1. Repeat the following t times:
   (a) Pick x uniformly at random from U.
   (b) If x belong to S, count 1.
   (c) If not, count 0.
2. Take an average of the above counts and return that as an estimate of $\frac{|S|}{|U|}$.

Let $p = \frac{|S|}{|U|}$ and consider the random variable X_i denote the output of i-th iteration. Clearly $\mathbf{E}[X_i] = \frac{|S|}{|U|} = p$.

Let X denote our estimate at the end of the algorithm, that is, $X = \frac{1}{t}\sum_{i=1}^{t} X_i$. Then $\mathbf{E}[X] = p$ and one can use the value $X \cdot |U|$ as an estimator for the size of $|S|$. There exists a trade-off between the value of t and the accuracy of the estimator. To determine after how many steps it is safe to stop the algorithm we'll use Chebyshev's inequality.

Chebyshev:

$\mathbf{Pr}[|X-\mu| \geq y\sigma] \leq \frac{1}{y^2}$, or using $y = \frac{\epsilon\mu}{\sigma}$, $\quad \mathbf{Pr}[|X-\mu| \geq \epsilon\mu] \leq \frac{\sigma^2}{\epsilon^2\mu^2}$

The first and second moments of the estimator x are given as follows.

$$\begin{aligned} \mu(X) &= \frac{|S|}{|U|} \\ \sigma^2(X_i) &= p(1-p) \\ \sigma^2(X) &= \frac{1}{t^2}\sum_{i=1}^{t}\sigma^2(X_i) = \frac{tp(1-p)}{t^2} = \frac{p(1-p)}{t} \end{aligned}$$

Using these bounds, we can bound the probability that our estimate is very far from the true value of $\frac{|S|}{|U|}$.

$$\begin{aligned} \mathbf{Pr}[\text{failure}] &= \mathbf{Pr}[\text{estimate is off by more than } 1+\epsilon] = \\ &\leq \frac{p(1-p)}{t\epsilon^2p^2} = \frac{1-p}{t\epsilon^2 p} \end{aligned}$$

So it is sufficient to take $t = \frac{4(1-p)}{\epsilon^2 p}$ to have a failure probability of at most $\frac{1}{4}$. In general, after t steps of sampling from a random variable with mean μ and variance σ^2, we have:

$$\begin{aligned}\mu_t &= \mu\\ \sigma_t &= \frac{\sigma^2}{t}\\ \mathbf{Pr}[\text{failure}] &\leq \frac{\sigma^2}{t\epsilon^2\mu^2} < \frac{1}{4} \text{ for } t = \left(\frac{4\sigma^2}{\epsilon^2\mu^2}\right)\end{aligned}$$

An estimator with the property $\mathbf{E}[X] = \mu$ (where μ is the quantity to be estimated) is called an *unbiased estimator*. Thus X given by the above algorithm is an unbiased estimator. The quantity $\frac{\sigma^2}{\mu^2}$ is called the *critical ratio*.

If $\frac{\sigma^2}{\mu^2}$ is poly(size of problem, $\frac{1}{\epsilon}$) the above algorithm is a FPRAS (fully polynomial randomized approximation scheme).

Definition 8.2.2 *A fully polynomial randomized approximation scheme (FPRAS) is a randomized algorithm A running in time* poly $(n, \frac{1}{\epsilon})$ *such that:*

$$\mathbf{Pr}[(1-\epsilon)f(x) \leq A(x) \leq (1+\epsilon)f(x)] \geq \frac{3}{4}$$

The algorithm A is called a PRAS if its running time is poly n.

As mentioned before, if the critical ratio is polynomial in n the above algorithm is a FPRAS for $\mu = \mathbf{E}[X]$. If we want $\mathbf{Pr}[\text{failure}] \leq \frac{\sigma^2}{t\epsilon^2\mu^2} < \delta$ we can take $t = \frac{\sigma^2}{\delta\epsilon^2\mu^2}$. When δ is large, choosing t this way is sufficient. However, if δ is very small, t becomes large. To do better we would like t to be of the order of $O(log(\frac{1}{\delta}))$.

Here is an algorithm that achieves this. This technique is known as *the median of means method.*

Denoting by A the previous algorithm, we will run $2s+1$ copies of A for some value of s to be determined later. Denote by $A^i(x)$ the output of the i-th instance of A. Our new algorithm outputs the **median** of all these $2s+1$ values.

A natural question here is why we don't take the mean. Note however that the algorithm A is not guaranteed to be within $(1+\epsilon)$ of μ with probability 1, so there may be cases when its output is far away from the true value of μ. Taking the mean in this case might have a bad impact on the final estimate, and our accuracy decreases only as $\frac{1}{t}$ and not as $\frac{1}{\exp(t)}$.

Let us now compute the probability of failure of the new algorithm. First note that failure in this case means that the median fell outside of the $[(1-\epsilon)\mu, (1+\epsilon)\mu]$ interval, an event which happens when at least $s+1$ instances of A have failed. We can use this fact to upper bound the probability of failure as follows:

$$\begin{aligned}
\mathbf{Pr}[\text{new algorithm fails}] &\leq \mathbf{Pr}[s+1 \text{ runs of algorithm of A fail}] \\
&\leq \sum_{i \geq s+1} \binom{2s+1}{i} \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{2s+1-i} \\
&\leq \left(\frac{1}{4}\right)^{s+1} \left(\frac{3}{4}\right)^s \sum_{i \geq s+1} \binom{2s+1}{i} \\
&= \frac{1}{2^{2\delta+2}} \left(\frac{3}{4}\right)^s 2^{2s} \\
&\leq \left(\frac{3}{4}\right)^s
\end{aligned}$$

We can now choose $s = \log_{\frac{3}{4}} \frac{1}{\delta}$ to obtained the desired bound.

Note that the new estimator may be biased, however we boosted the probability that $A(x) \in [(1-\epsilon)\mu, (1+\epsilon)\mu]$.

Let us see how can we apply the previous results to #DNF. Recall that U denotes the set of all assignments and S denotes the set of satisfying assignments, $\frac{|S|}{|U|} = p$. Then after $t \simeq \Omega(\frac{1-p}{\epsilon^2 p})$ steps, the probability of failure is below $\frac{1}{4}$.

Now, if p is large, the algorithm has a high chance of succeeding, however when p is small (say $\frac{1}{2^n}$), then it may happen that no element from S is sampled within any polynomial number of steps. This is an inherent problem to this approach, due to the fact that the size of the sampling universe is very large. In the following subsection we will see a different sampling technique, called *importance sampling*. The idea will be to shrink the size of the sampling universe to only a sub-population of interest, thereby increasing the value of p.

8.2.1 Importance Sampling

This method is due to Karp, Luby and Madras.

We assume that $\frac{\sigma^2}{\mu^2}$ is small. The algorithm will boost p by "decreasing" the size of the universe U.

Let S_i be the set of assignments satisfying clause C_i, $i = 1 \dots m$. Then $S = \bigcup_{i=1}^{m} S_i$ is the set whose size we need to estimate.

Define $U' = \{(i, a_i) | a_i \in S_i\}$. Then $|U'| = \sum_i |S_i|$. The sets $|S_i|$ have the following properties:

1. For all i, $|S_i|$ is computable in polynomial time.
2. It is possible to sample uniformly at random from each S_i.
3. For any $a \in U$ it can be determined in polynomial time whether $a \in S_i$.

To understand how the sampling process works we will use the following construction. Imagine a table with the rows indexed by the elements of U and the columns indexed by the sets S_i.

$$\begin{array}{cccccc} & S_1 & S_2 & S_3 & \dots & S_m \\ a_1 & \overline{*} & & * & & \\ a_2 & \overline{*} & * & & & \\ a_3 & & \overline{*} & & & \\ \vdots & & & & & \end{array}$$

We place a $*$ at the intersection of each row corresponding to an element of $a \in U$ and a column corresponding to a set S_i such that $a \in S_i$ (implying that a satisfies clause C_i). Thus all assignments satisfying a clause will have a corresponding row containing at least one $*$. To determine the size of S it would be sufficient to count all such rows. For that we use the following procedure: for each row containing at least one star, make one of its stars a "special" star ($\overline{*}$). Now the number of special stars will be equal to the size of $|S|$.

To count the number of special stars we use the same idea as before: sample a $*$ uniformly at random from the set of all $*$'s (U') and then test if it is special. If so, count 1, otherwise count 0. After a number of steps stop and output $X \cdot |U'|$, where X is the total number of 1's.

To complete the description of the algorithm we need two more things:

(i) A method to pick up a $*$ uniformly at random from the table.

(ii) Determine when a star is special.

For (ii) we define a $*$ as being **special**, if it is the first one in its row. Now it becomes easy to test in polynomial time that a $*$ is special by using property 3 from above: if the corresponding assignment satisfies the $*$'s column but no other column before that, then it is special. Otherwise it is not.

For (i) (picking up a $*$ u.a.r.) we use the following algorithm:

- Compute $|S_i|$.
- Pick i with probability $\frac{|S_i|}{\sum\limits_i |S_i|}$.
- Pick a random assignment satisfying C_i.

This is again easy to do, by the properties of the sets $|S_i|$.

There is one more thing we need to check before we are done: the fact that p is sufficiently large. But,

$$p = \frac{|S|}{|U'|} = \frac{\#\text{ of special } *' s}{\#\text{ of } *' s} \geq \frac{1}{m}.$$

Therefore, after $t \simeq \Omega(\frac{m}{\epsilon^2})$ steps we have a fairly good estimator for the size of S.

8.3 Permanent of a matrix M

Let M be a $n \times n$ $\{0,1\}$ matrix. Its permanent is defined as follows:

$$Perm(M) = \sum_{\pi \in S_n} \prod_{i=1}^{n} M_{i\pi(i)}$$

This is very similar to the determinant of M less the sign of each permutation π.

$$Det(M) = \sum_{\pi \in S_n} sgn(\pi) \prod_{i=1}^{n} M_{i\pi(i)}$$

Computing Perm is #P-complete. Note that Perm is equivalent to counting the number of perfect matchings in a bipartite graph. If we take the two sets of vertices to be the row numbers and the column numbers respectively, each non-zero term in the permanent corresponds to a perfect matching in the associated bipartite graph G_M.

An unbiased estimator for Perm

Replace each 1 in M by +1 or -1 independently at random and let M' be the newly obtained matrix. Denote by $X = (Det(M'))^2$.

Claim 8.3.1 $\mathbf{E}[X]$=*Perm(M)*

Proof: Let S be the set of permutations for which $\prod_{i=1}^{n} M_{i\pi(i)} = 1$. Then $Perm(M) = |S|$.

For each $\pi_i \in S$, denote by $X_i = sgn(\pi_i) \prod_{j=1}^{n} M'_{j\pi_i(j)} \in \{-1, 1\}$.

Then we have the following:

$$\begin{aligned}
Det(M') &= \sum_{\pi_i \in S} X_i. \\
X &= (Det(M'))^2 = \sum_i {X_i}^2 + \sum_{i \neq j} X_i X_j \\
\mathbf{E}[X] &= \sum_i \mathbf{E}\left[{X_i}^2\right] + \sum_{i \neq j} \mathbf{E}[X_i]\mathbf{E}[X_j] \\
&= |S| + 0 \\
&= Perm(M)
\end{aligned}$$

where we used the fact that the variables X_i and X_j are pairwise independent for $i \neq j$ and each of them has zero mean. ■

This estimator has a high variance, therefore in pactice we use something different. This will be presented later in the class.