Randomized Methods in Computation*
Lecture 5: Approximate Counting and Uniform Generation

Notes taken by Oded Lachish, Eran Ofek, Udi Wieder

Spring 2001

Summary: This lecture discusses quantitative problems related to NP relations. Namely, given an element in an NP language, we ask how many witnesses this element has. We use Hash functions in order to prove that given an oracle to NP and an instance in an NP language, it is possible to approximate the number of witnesses for that instance.

1 Introduction

Given a set S, a few natural questions one can ask about S can be divided into two categories.

1. Qualitative Questions:
   - Decision Question - is S empty or not? $S \stackrel{?}{=} \phi$
   - Search Query - produce an element of S. Find y such that $y \in S$.
2. Quantitative Questions:
   - Counting - What is the cardinality of S. $|S| =?$.
   - Uniform Generation - Generate an element of S with uniform probability among all possibilities.

Typically quantitative questions are harder to answer than qualitative ones. If one can count the number of elements in S then one can surely tell whether S is empty or not. We investigate what happens the other way around, namely, when being able to answer qualitative questions, what can be said about the ability to answer the quantitative ones?

1.1 The Model

We are given a family of sets $\{S_x\}_{x\in\{0,1\}^*}$ where:

- $S_x \subseteq \{0,1\}^*$. Furthermore there exists a polynomial $p(\cdot)$ such that the length of elements in S_x is at most $p(|x|)$.
- Given x and y it is easy (=done in polynomial time) to check whether $y \in S_x$.

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

In other words, the family $\{S_x\}_{x\in\{0,1\}^*}$ defines an NP relation. Recall:

Definition 1 (NP relation) *: An NP relation is a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ such that:*

- *R is polynomial time decidable.*
- *There exists a polynomial $p(\cdot)$ such that for every $(x,y) \in R$, it holds that $|y| \leq p(|x|)$.*

We let S_x be the witness set of x; i.e. $S_x \stackrel{\triangle}{=} \{y : (x,y) \in R\}$. The witness set of x with respect to relation R is also denoted by $R(x)$.
Now we can use this notation to rephrase the problems at hand. The Decision Question is the usual NP decision problem, namely given x check if $\exists y$ such that $(x,y) \in R$. In other words, check if $R(x) \stackrel{?}{=} \phi$.
The Search Problem can be phrased: given x find y such that $y \in R(x)$.
The Counting Problem: given x compute $|R(x)|$.
The Generation Problem: given x, generate y uniformly in $R(x)$.

1.2 How close are the quantitative questions to the qualitative questions?

First we investigate the connection between the two qualitative questions. How close are the Decision Problem and the Search Problem? Surly, given an oracle to the Search Problem, it is easy to solve the Decision Problem. A well known result shows that for NP-complete relations the other way around holds as well. In other words:

Fact 1.1 *(Self Reducibility of NP) Given an NP-relation R, an element x and an oracle access to NP decision questions, it is possible in polynomial time to find y such that $y \in R(x)$.*

Note that the oracle can answer any query which is an NP decision problem, and not necessarily queries regarding the specific relation R. In case R is NP-complete, we can use an oracle to $L(R) \stackrel{\triangle}{=} \{x : R(x) \neq \phi\}$ instead of an oracle to NP.

Fact 1.1 demonstrates a tight connection between the two qualitative questions. In the rest of the lecture we show that in some sense the quantitative questions are not much harder than the qualitative ones. Namely, given an oracle to the Decision Question, it is possible to approximate the number of witnesses of an element. In the next lecture we will present a result in the spirit of Fact 1.1 that demonstrates a tight connection between counting and uniform generation.
****

2 Approximating the size of $R(x)$ with an oracle access to NP

Goal: For a fixed NP-relation R, provide an algorithm that, given an element x and $\epsilon > 0$, runs in time $poly\left(\frac{|x|}{\epsilon}\right)$ and satisfies:

$$\Pr\left[Alg^{NP}(x) \notin (1 \pm \epsilon)|R(x)| \leq \frac{1}{3}\right].$$

The tools we use are a family of pairwise independent Hash functions $H_{n,m}$, that hashes $\{0,1\}^n \mapsto \{0,1\}^m$ $(m < n)$, and an oracle for any NP complete language. The strategy for solving the problem is first to show how to find a solution to the case in which $|R(x)|$ is small. (This is done using an oracle to NP). Then show how to reduce the general case (a large witness set), to a case in which the witness set is small. (This is done using the hash functions). Using a family of pair-wise independent hash functions has a price: there is a constant probability that the hash function chosen does not behave well, we denote this probability by δ. When the chosen hash function behaves well, the answer of the algorithm will be in the interval $(1 \pm \epsilon) \cdot |R(x)|$. Where ϵ is called the deviation parameter, and δ is called the error probability. Thus, for an NP relation R and an input x, the algorithm we describe efficiently produces a result that behaves as follows:

$$Prob\left[Alg^{\mathcal{NP}}(x) \notin (1 \pm \epsilon)|R(x)|\right] < \delta$$

where the probability is over the coin tosses of the algorithm.

2.1 The algorithmic scheme

Assume that $|x| = n$ and that $R(x) \subseteq \{0,1\}^n$ (this assumption is with out loss of generality, since padding can bring the element x and the witnesses to the same size). Let $l = O(\lg n)$.

Following is a sketch of the algorithm (which is actually an algorithmic scheme):

Algorithm A:

1. If $|R(x)| < 2^l$ output $|R(x)|$.
2. If $|R(x)| \geq 2^l$ then
   For $i = 1$ to $n - l$
   - Select $h \in_R H_{n,i}$.
   - If $|R(x) \cap h^{-1}(0^i)| < 2^l$ then
     output $2^i \cdot |R(x) \cap h^{-1}(0^i)|$ and Halt.

In Stage 1 we handle the case in which $|R(x)|$ is small. In the iterations of Stage 2 we hash the witness set into an increasingly larger set, until the number of witnesses that are mapped to 0^i is small. Specifically, if the hash function maps $\{0,1\}^n$ to $\{0,1\}^i$, we expect it to map $\frac{|R(x)|}{2^i}$ witnesses to each element in $\{0,1\}^i$. Once 2^i is the 'right' number, we have that a small number of elements are mapped into 0^i. (0^i was chosen arbitrarily). This case is similar to the case of a small a witness set, so is identifiable, and can be counted exactly. As a result the size of the set of witnesses mapped to 0 is used to approximate the entire number of witnesses.

2.2 Implementation of the algorithmic scheme

How to perform Stage (1)**:** We check if $|R(x)| < 2^l$ by defining the following NP relation:

$$R' = \{x, (y_1, \ldots, y_{2^l}) : y_1 \neq \ldots \neq y_{2^l} \text{ and } (x, y_1), \ldots (x, y_{2^l}) \in R\}.$$

That is, $L(R') = \{x : \exists y'(x, y') \in R'\}$ contains x iff x has at least 2^l *distinct* witnesses (i.e. $|R(x)| \geq 2^l$). Since R is an NP relation we know that the length of each y_i is polynomially bounded by the length of x. Since 2^l is polynomial in $|x|$ we have that the length of $(y_1, y_2, \ldots, y_{2^l})$ is also polynomially bounded by $|x|$. We conclude that R' is an NP relation.

Thus, to check whether $|R(x)| \geq 2^l$ we query the oracle whether $R'(x) \stackrel{?}{=} \phi$. If the oracle answers 'NO' then $|R(x)| < 2^l$. To compute the exact size of $|R(x)|$ define the following NP relation:

$$R'' = \{(x, 1^t), (y_1, \ldots, y_t) : y_1 \neq \ldots \neq y_t \text{ and } (x, y_1), \ldots (x, y_t) \in R\}.$$

That is, the language $L(R'')$ contains $(x, 1^t)$ iff x has at least t *distinct* witnesses (i.e. $|R(x)| \geq t$). In order to ensure that R'' is indeed an NP relation, we have to ensure that the length of the witnesses is polynomially bounded by the length of the instances. Note that t is represented in unary, therefore the length of the instance $(x, 1^t)$ is $|x| + t$. The witnesses are composed of t y_i's, whose lengths are polynomially bounded by $|x|$. Therefore, for every t, the length of $(y_1, y_2, \ldots, y_t)$ is polynomially bounded by $|(x, 1^t)|$ and we conclude that R'' is an NP relation. Thus, to find the exact size of $R(x)$, we iteratively query the oracle on the decision problem of $R''(x, 1^t)$ for $t = 1 \ldots 2^l$ and find the largest t for which $(x, 1^t) \in L(R'')$. Note that 2^l is polynomial in n, thus the whole operation runs in polynomial time.

How to perform stage (2): For each iteration i define the following NP relation:

$$R''' = \{(x, h), (y_1, \ldots, y_{2^l}) : y_1 \neq \ldots \neq y_t \text{ and } h(y_1) = h(y_2) = \cdots = h(y_{2^l}) = 0^i$$
$$\text{and } (x, y_1), \ldots, (x, y_{2^l}) \in R\}$$

That is, $L(R''')$ contains (x, h) iff x is in $L(R)$, and it has at least 2^l distinct witnesses mapped to 0^i by h. Note that the hash function can be evaluated in time that is polynomial in $|x|$. Therefore it is possible to verify in polynomial time whether $((x, h), (y_1, y_2, \ldots, y_{2^l})) \in R'''$. Thus $L(R''') \in \mathcal{NP}$. Furthermore, since h can be represented in space which is polynomial in $|x|$, the length of (x, h) is polynomial in $|x|$, and it is possible to produce an NP query for R''' in polynomial time (in $|x|$).
Querying the oracle about $R'''(x)$ enables us to identify when less than 2^l distinct witnesses were mapped to 0^i by h. In this case, finding the exact number of such witnesses, is done in a similar manner to the first case when $|R(x)|$ was small.

2.3 Analysis of the algorithm

The parameter l is chosen to be big enough so that $\epsilon \geq 10 \cdot 2^{-l/2}$ and also $2^l \geq n^2$. Note that since the running time is polynomial in 2^l, it follows that the running time is polynomial in $\max\{\text{n}, \frac{1}{\epsilon}\}$. This means that ϵ can be at best small as $\frac{1}{poly(n)}$, otherwise the running time would not be polynomial in n.
Let t be such that $2^t = |R(x)|$ (for sake of simplicity and without loss of generality assume t is an integer).

General strategy of the proof:
We will show that the algorithm is not likely to terminate before iteration $t - l$; that is, with high probability it will reach iteration $t - l$. Then, we show that given that A reached iteration $t - l + 1$ it is very likely that A will terminate in this iteration. This implies that the algorithm is likely to terminate in one of the iterations $\{t - l, t - l + 1\}$. The last step is to show that in both iterations ($i = t - l$ or $i = t - l + 1$) the value $|R(x) \cap h^{-1}(0^i)| \cdot 2^i$ is an ϵ approximation of $|R(x)|$ with high probability. Thus if A terminates in these iterations (as expected) it will output an ϵ approximation of $|R(x)|$ with high probability.

Recall the hashing lemma from the previous lecture:

Lemma 2.1 (Hashing lemma) *Let $H = \{h : \{0,1\}^n \to \{0,1\}^i\}$ be a family of pairwise independent hash functions. Let $I \subseteq \{0,1\}^n$ and $\alpha \in \{0,1\}^i$,and let $\rho = \frac{|I|}{2^i}$. Then*

$$\Pr_h\left[\left||I \cap h^{-1}(\alpha)| - \rho\right| \geq \epsilon \cdot \rho\right] \leq \frac{1}{\epsilon^2 \cdot \rho}$$

where h is uniformly selected in H.

We will use this lemma to prove that algorithm A will return a good approximation of $|R(x)|$, with high probability. We do this by examining the value $|R(x) \cap h^{-1}(0^i)|$ in various iterations of A. Let us fix some i (iteration number in the running of A).

Recall that since $h \in H$ is pairwise independent, it follows that

$$E_h\left[|R(x) \cap h^{-1}(0^i)|\right] = \frac{|R(x)|}{2^i} \tag{1}$$

We denote $\rho \stackrel{\text{def}}{=} \frac{|R(x)|}{2^i}$. For every $y \in R(x)$ denote by X_y the following random variable:

$$X_y = X_y(h) = \begin{cases} 1 & h(y) = 0^i, \\ 0 & otherwise. \end{cases}$$

where the probability space is of all choices of $h \in H$.
Recall that by the properties of H, in iteration i we have $\Pr_h[X_y = 1] = \frac{1}{2^i}$. Notice that

$$|R(x) \cap h^{-1}(0^i)| = \sum_{y \in R(x)} X_y$$

Thus, by linearity of the expectation:

$$E_h\left[|R(x) \cap h^{-1}(0^i)|\right] = \sum_{y \in R(x)} E_h(X_y)$$

$$= \sum_{y \in R(x)} \Pr[X_y = 1] = \frac{|R(x)|}{2^i} \quad = \quad \rho$$

which indeed agrees with equation (1).

First we show that the algorithm A is not likely to terminate in any of the iterations $i \in \{1, 2, .., t-l-1\}$.

Claim 2.2 *For every $i \in \{1, 2, .., t-l-1\}$ the probability that A terminates at iteration i is smaller than $\frac{1}{10n}$.*

Proof:

$$\rho = \frac{|R(x)|}{2^i} \geq \frac{2^t}{2^{t-l-1}} \geq 2^{l+1}$$

We wish to bound the probability that algorithm A terminates at iteration i, so we need to bound the probability that $|R(x) \cap h^{-1}(0^i)| < 2^l$. Notice that if $|R(x) \cap h^{-1}(0^i)| < 2^l$ then $||R(x) \cap h^{-1}(0^i)| - \rho| > \frac{1}{2} \cdot \rho$ (this is true since $\rho \geq 2^{l+1}$). By using the hashing lemma we can bound the probability that $||R(x) \cap h^{-1}(0^i)| - \rho| > \frac{1}{2} \cdot \rho$:

(The set $R(x) \subseteq \{0,1\}^n$ plays the roll of the set I from the lemma).

$$\Pr_h\left[||R(x) \cap h^{-1}(0^i)| - \rho| > \frac{1}{2} \cdot \rho\right] \leq \frac{1}{(\frac{1}{2})^2 \cdot \rho}$$

$$\leq \frac{1}{(\frac{1}{2})^2 \cdot 2^{l+1}} = 2^{-l+1} \leq 2 \cdot \frac{1}{n^2} \leq \frac{1}{10 \cdot n}$$

(for big enough n's). ■

Now we will show that A is not likely to continue beyond iteration $t - l + 1$.

Claim 2.3 *Given that A reached iteration $t-l+1$, the probability that A terminates at this iteration is at least* 0.9.

Proof: Here, we have

$$\rho = \frac{|R(x)|}{2^{t-l+1}} = \frac{2^t}{2^{t-l+1}} = 2^{l-1}$$

We wish to bound the probability that algorithm A does not terminate in this iteration. This happens only if $|R(x) \cap h^{-1}(0^i)| \geq 2^l$, but this event is contained in the event that $||R(x) \cap h^{-1}(0^i)| - \rho| \geq \rho$ (since $\rho = 2^{l-1}$). We use the hashing lemma to bound this probability:

$$\Pr_h \left[\left| |R(x) \cap h^{-1}(0^i)| - \rho \right| \right] \geq 1 \cdot \rho] \leq \frac{1}{\rho}$$

$$= \frac{1}{2^{l-1}} = 2^{-l+1} = 2 \cdot \frac{1}{n^2} \leq 0.1$$

(for large enough n's) ■

It remains to show that given that A terminates at iteration $t-l$ or $t-l+1$ it is likely that A outputs an ϵ approximation of $R(x)$.

Claim 2.4 *Assume that A terminates at iteration $t-l$ or $t-l+1$, then*

$$\Pr_h \left[A(x) \notin (1 \pm \epsilon) \cdot |R(x)| \right] \leq \frac{1}{10}$$

Proof: Here, we have

$$\rho = \frac{|R(x)|}{2^i} \geq \frac{2^t}{2^{t-l+1}} = 2^{l-1}$$

Since $\frac{A(x)}{2^i} = |R(x) \cap h^{-1}(0^i)|$, it follows that the following two conditions are equivalent:

$$|A(x) - |R(x)|| \geq \epsilon \cdot |R(x)| \tag{2}$$

$$\left| |R(x) \cap h^{-1}(0^i)| - \frac{|R(x)|}{2^i} \right| \geq \epsilon \cdot \frac{|R(x)|}{2^i} \tag{3}$$

Equation (3) can be written as:

$$\left| |R(x) \cap h^{-1}(0^i)| - \rho \right| \geq \epsilon \cdot \rho$$

Using the hashing lemma, we have

$$\begin{aligned}
\Pr_h \left[\left| |R(x) \cap h^{-1}(0^i)| - \rho \right| \geq \epsilon \cdot \rho \right] &\leq \frac{1}{\epsilon^2 \cdot \rho} \\
&\leq \frac{1}{\epsilon^2 \cdot 2^{l-1}} \\
&\leq \frac{1}{(10 \cdot 2^{l/2})^2 \cdot 2^{l-1}} \leq \frac{1}{10}
\end{aligned}$$

Using the equivalence of equations (2) and (3), the claim follows. ■

Corollary 2.5 *The probability that A outputs an ϵ approximation of $|R(x)|$ is at least $\frac{2}{3}$.*

Proof: The probability that A terminates before iteration $t-l$ is at most $\frac{1}{10}$ (using the union bound with Claim 1). It follows that the probability for algorithm A to terminate at iteration $t-l$ or at iteration $t-l+1$ is at least 0.8, because

$$\Pr[\text{A terminates at iteration t-l or t-l+1}] =$$

$$\Pr\left[\begin{array}{l}\text{A reaches,}\\ \text{iteration t-l}\end{array}\right]\cdot\Pr\left[\begin{array}{l}\text{A terminates at,}\\ \text{iteration t-l or t-l+1}\end{array}\;\middle|\;\begin{array}{l}\text{A reaches,}\\ \text{iteration t-l}\end{array}\right]$$

$$\geq 0.9\cdot 0.9 > 0.8$$

Using again the union bound with Claim 3 we conclude that the probability that A terminates at iterations $t-l$ or $t-l+1$ and outputs an ϵ approximation is at least $0.8-0.1>\frac{2}{3}$. ■

THE END OF THE PAPER ***********************

2.4 Amplifying the success probability

In the previous section we saw that we can approximate the size of the witness set with deviation ϵ, and in running time that is polynomial in $\frac{1}{\epsilon}$. The success probability of the approximation was shown to be larger than $\frac{2}{3}$. Like in many other probabilistic algorithms, we wish to amplify the success probability. We will show that in polynomial time, it is possible to reduce the error probability to be exponentially small.

We start tackling this problem in a standard way: assume we wish the error probability to be smaller than δ. We set $t \triangleq O(\log(\frac{1}{\delta}))$. The algorithm A is invoked t times, each invocation with fresh random coins, and we receive t (possibly different) answers. The question is what to do with these t answers?

At first thought one might think of taking the average of these results. On second thought this idea should be excluded, since the wrong solutions might be very wrong. If it happens that the algorithm miserably fails, it might give an answer that is so far from the true one, that it would skew the average. Consider the following illustrative example: Assume $|R(x)| = 2^{\frac{n}{2}}$, and $\Pr[A(x) = 2^{\frac{n}{2}}] = \frac{2}{3}$, yet $\Pr[A(x) = 2^n] = \frac{1}{3}$. That is, even though A is correct most of time, if it fails it fails miserably. Now the average is $E\left(A(x)\right) = \frac{2}{3}\cdot 2^{\frac{n}{2}} + \frac{1}{3}\cdot 2^n = \Theta(2^n)$, which is very far from the correct answer.

A much better idea is to use the median of the t results. Let A' be the following algorithm:

Algorithm A':

1. Set $t \triangleq O(\log(\frac{1}{\delta}))$
2. Invoke the algorithm A, t times, each invocation with fresh random coins, and receive t answers.
3. Output the median of the t answers.

Claim 2.6 *Prob* $[A'(x) \notin (1\pm\epsilon)\cdot|R(x)|] \leq \delta$

Proof: We use the following fact:

Fact 2.7 *If the median of t numbers is outside the range (a,b) then at least $\frac{t}{2}$ of the numbers are outside the range (a,b).*

Define random variables

$$X_i = \begin{cases} 1 & \text{the } i\text{'th invocation of A gives a value in } (1\pm\epsilon)\cdot|R(x)|, \\ 0 & \textit{otherwise.} \end{cases}$$

Note that $\Pr[X_i = 1] \geq \frac{2}{3}$, therefore $E(\sum_{i=1}^{t} X_i) \geq \frac{2t}{3}$.
According to Fact 2.7, if A' fails then at least half of the invocations of A failed. This means that (in case of failure) $\sum_{i=1}^{t} X_i \leq \frac{t}{2}$ and thus the sum is very far from it's expectation. We use the Chernoff bound to show that this is very unlikely.
According to Fact 2.7 we have:

$$\begin{aligned} \Pr[A'(x) \notin (1\pm\epsilon)\cdot|R(x)|] &\leq \Pr\left[\sum_{i=1}^{t} X_i \leq \frac{t}{2}\right] \\ &\leq \Pr\left[\left|\sum_{i=1}^{t} X_i - E(\sum_{i=1}^{t} X_i)\right| \geq \frac{t}{6}\right] \\ &\leq \exp\left(-\left(\frac{1}{6}\right)^2 \cdot t\right) \end{aligned}$$

where the last inequality is due to Chernoff Bound.
Thus, setting $t = 36\ln\left(\frac{1}{\delta}\right)$ we get $\Pr[A'(x) \notin (1\pm\epsilon)\cdot|R(x)|] \leq \delta$, and the claim follows. ■