0368.4163 Randomness and Computation	March 25, 2009

Lecture 5

Lecturer: Ronitt Rubinfeld *Scribe: Inbal Marhaim, Naama Ben-Aroya*

Today

- Uniform generation of DNF satisfying assignments
- Uniform generation and approximate counting

1 Disjunctive Normal Form

Definition 1 *In Boolean logic, a* disjunctive normal form *(DNF) is a standardization (or normalization) of a logical formula which is a disjunction of conjunctive clauses, such as:*

$$F = x_1\bar{x_2} \vee x_2x_4\bar{x_5} \vee \dots$$

1.1 Questions

Question 1 *Is the formula satisfiable?*

Yes, unless $F = 0$.

Question 2 ***The counting problem:*** *How many assignments satisfy F?*

Let $\#F$ denote the number of satisfying assignments to the formula F.
First we notice that

$$F \text{ is a DNF} \iff \bar{F} \text{ is a CNF}$$

therefore

$$\#F < 2^n \iff \bar{F} \text{ is satisfiable}$$

Since 'Is CNF formula is satisfiable?' know to be hard (NP-complete) - the counting problem for DNF is also hard.

Question 3 *Can we generate a uniformly random satisfying assignment for a DNF formula?*

2 Generate a Random Satisfying Assignment

Naive algorithm: Pick a random assignment.

- If satisfies F, output it.
- Else, repeat.

Problem: run time: $\Omega(\frac{1}{p})$, where $p = \frac{\#F}{2^n}$.

2.1 One-Term DNF

Example:

$$F = x_1\bar{x_2}$$

Random satisfying assignment: set $x_1 = 1, x_2 = 0$, set all other variables randomly.
In general: set the variables of the clause to satisfy it and all other variables randomly. For one-term DNF it easy to compute $\#F$: if it's (only) clause has k variables then $\#F = 2^{n-k}$.

2.2 Two-Term DNF

Let S_i denote the set of satisfying assignments of F_i.

2.2.1 Disjoint Sets: $S_1 \cap S_2 = \emptyset$

In this case: $\#F = |S_1 \cup S_2| = 2^{n-k_1} + 2^{n-k_2}$. Uniform generation: Pick S_i with probability $\frac{|S_i|}{|S_1|+|S_2|}$. Output random element of S_i.

2.2.2 Non Disjoint Sets: $S_1 \cap S_2 \neq \emptyset$

The previous algorithm is OK for assignments from $S_1 \triangle S_2$ $(= (S_1 \setminus S_2) \cup (S_2 \setminus S_1))$, but assignments from $S_1 \cap S_2$ get twice as much probability.

Uniform generation: Pick S_i with probability $\frac{|S_i|}{|S_1|+|S_2|}$. Pick random element of S_i.

- If in $S_1 \triangle S_2$ then output it.
- Else (it is in $S_1 \cap S_2$): toss a coin. If 'Heads' - output, else do whole procedure again.

2.3 General Case: $F = F_1 \vee F_2 \vee \ldots \vee F_m$

Algorithm A:

Step 1: {Pretend S_i's disjoint to pick element}
Pick clause $i \in [m]$ with probability $\frac{|S_i|}{\sum |S_j|}$.
Then pick a random satisfying assignment π to clause i.
Step 2: {Compensate for intersections}
Compute $\ell = |\{j \in [m] : \pi \in S_j\}|$ (= number of clauses satisfied by π).
Then toss a coin with bias $\frac{1}{\ell}$.
If 'Heads' - output π and halt.
Else, start over.

Claim 2 *Algorithm A outputs each satisfying assignment π with same probability.*

Proof

$$\begin{aligned} \Pr[\text{pick } \pi \text{ in Step 1}] &= \sum_{\text{clause i that satisfied by } \pi} \Pr[\text{pick clause i}] \cdot \Pr[\text{output } \pi | \text{ picked } i] \\ &= \sum_i \frac{|S_i|}{\sum |S_j|} \cdot \frac{1}{|S_i|} = \frac{\ell}{\sum |S_j|} \end{aligned}$$

$$\Pr[\text{output } \pi] = \frac{\ell}{\sum |S_j|} \cdot \frac{1}{\ell} = \frac{1}{\sum |S_j|}$$

So the probability to pick a random assignment is a constant, moreover it does not depend on the number of clauses it satisfied. ■

Claim 3

$$E[\# \textit{ loops until output something}] \leq m.$$

Proof Idea Since all biases in Step 2 are $\geq \frac{1}{m}$. ■

3 P-relation

Definition 4 *$R \subseteq \{0,1\}^* \times \{0,1\}^*$ binary relation on strings. R is a* p-relation *if*

- $\forall (x,y) \in R : |y| = poly(|x|)$.
- $\exists$ *poly-time decision procedure for deciding whether* $(x,y) \in R$.

Examples:

- $R_{SAT} = \{(x,y)|x$ is a formula, y is a satisfying assignment to $x\}$
- $R_{matching} = \{(x,y)|x$ is a graph, y is a perfect matching in $x\}$

Theorem 5 *$L \in NP$ iff $\exists$p-relation R s.t.*

$$x \in L \iff \exists y \text{ s.t. } (x,y) \in R$$

.

Definition 6 *A function $f : \{0,1\}^* \to \mathbb{N}$ is in $\#P$ iff $\exists$ p-relation R s.t. $\forall x$, $f(x) = |\{y|(x,y) \in R\}|$.*

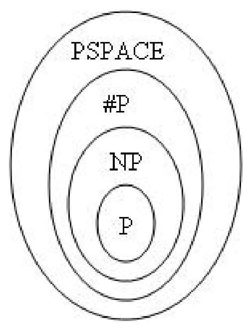

Figure 1: #P complexity class

$f(x)$ is the number of witnesses for an input x, which can each be validated in polynomial time.
It was shown that the counting problem for DNF (i.e. How many different variable assignments will satisfy a given DNF formula?) is $\#P$-complete.
Examples:

- $\#SAT$ - The number of satisfying assignments to a given formula
- $\#Matching$ - The number of perfect matches[1] in a given graph

Definition 7 *A is an* approximate counter *for f if*

$$\forall x \Pr[f(x)/(1+\epsilon) \leq A(x) \leq f(x)(1+\epsilon)] \geq 3/4$$

If the runtime of A is polynomial in $1/\epsilon$, $|x|$, then A is a fully polynomial randomized approximation scheme (FPRAS)

[1] A matching is a set of pairwise non-adjacent edges; that is, no two edges share a common vertex. A perfect matching is a matching which matches all vertices of the graph.

* Note that in Homework 1, we have shown that 3/4 can be replaced by any $1-\delta$ ($0<\delta<1$), repeating $A(x)$ $O(log 1/\delta)$ times.

Observation 8 *There is no FPRAS for #CNF (assuming $NP \neq RP$). If there were, then we could give a randomized algorithm to solve SAT in polynomial time.*

- *$SAT(x)$=yes $\Rightarrow$ $\#CNF(x) > 0$ $\Rightarrow$FPRAS$(x) > 0$, with probability $\geq 3/4$.*
- *$SAT(x)$=no $\Rightarrow$ $\#CNF(x) = 0$ $\Rightarrow$FPRAS$(x) = 0$, with probability $\geq 3/4$.*

Therefore, if we had an FPRAS for #CNF, we could answer SAT correctly with probability at least 3/4.

Definition 9 *R is a p-relation, $f : \{0,1\}^* \to \mathbb{N}$ s.t. $f(x) = |\{y|(x,y) \in R\}|$. M is a* uniform - generator *for R if on input x*

- *M's runtime is polynomial in $|x|$*
- *if $f(x) > 0$ then $\forall y$ s.t. $(x,y) \in R$ $\Pr[M \text{ outputs } y] = 1/f(x)$*
- *M does not output y s.t. $(x,y) \notin R$ (M might not output anything)*

Example: The random assignment generator for DNF we saw. The probability the generator will return an assignment is equal for all the assignments, therefore it is equal to $1/f(x)$, while $f(x)$ is the number of assignments.

Definition 10 *R is a p-relation, $f : \{0,1\}^* \to \mathbb{N}$ s.t. $f(x) = |\{y|(x,y) \in R\}|$. M is an* almost uniform generator (AUG) *for R if on input x*

- *M's runtime is polynomial in $|x|$, $[1/\epsilon]$*
- *if $f(x) > 0$ then $\forall y$ s.t. $(x,y) \in R$ $1/f(x)(1+\epsilon) \leq \Pr[M \text{ outputs } y] \leq (1+\epsilon)/f(x)$*
- *M does not output y s.t. $(x,y) \notin R$ (M might not output anything in this case)*

Definition 11 (Jerrum, Valiant, Vazirani) *A problem is called* self-reducible *if it can be solved via recursion on smaller problems of the same type.*

The formal definition can be found in Jerrum, Valiant and Vazirani's work.
Example: SAT - $\varphi(x_1, \ldots, x_n)$ is satisfiable if $\varphi(0, x_2, \ldots, x_n)$ or $\varphi(1, x_2, \ldots, x_n)$ is satisfiable.

Note: the running time of each recursive step must be polynomial but the number of recursions can be large (even exponential).

4 FPRAS and AUG for SAT

Theorem 12 (Jerrum, Valiant, Vazirani) *R is a self-reducible p-relation, then $\exists$ FPRAS for R iff $\exists$ AUG for R.*

This theorem implies that if one can approximate the number of solutions of a problem, he can also generate uniformly a random solution for the problem and vice versa.

Proof Idea We will prove the theorem for the #SAT problem

4.1 Building an AUG from a FPRAS

First we will assume that there is an exact counter for #SAT, and we will use it to build an AUG for #SAT.

We will use the fact that SAT is self-reducible, and define a self-recursion tree for a SAT formula: Let $b_1, \ldots, b_j$ be the index of the node with setting of prefix of $x_1, \ldots, x_j$ s.t. $x_i = b_i \forall i \in [j]$.
$F_{b_1,\ldots,b_j}$ is a new formula on $n - j$ variables.

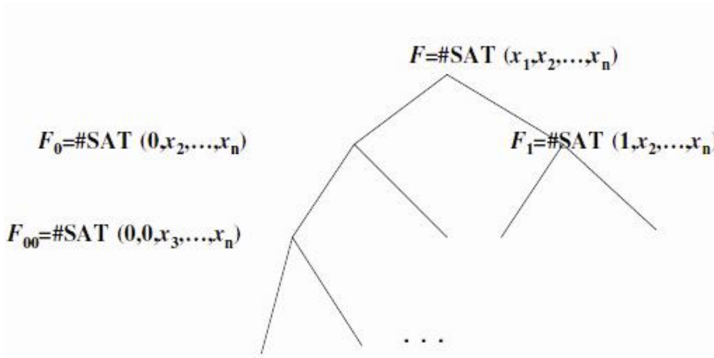

Figure 2: Self-Reducibility Tree

We will use the #SAT exact counter to guide a "random walk" from the root to a leaf.
Let $b_1, \ldots, b_k$ be the current node.

- Use the exact counter and compute
  $f_0 = F_{b_1,\ldots,b_k,0} = \#SAT\varphi(b_1, \ldots, b_k, 0, x_{k+2}, \ldots, x_n)$
  $f_1 = F_{b_1,\ldots,b_k,1} = \#SAT\varphi(b_1, \ldots, b_k, 1, x_{k+2}, \ldots, x_n)$
- Go left with probability $f_0/(f_0 + f_1)$, right with probability $f_1/(f_0 + f_1)$
- If $f_0 = 0$ and $f_1 = 0$ stop and output nothing
- If reached a leaf output it

Notice that if there are satisfying assignments the counter will return a number that is bigger than zero for at least one of the sides, therefore the algorithm will always choose a sub-tree which contains satisfying assignments, and will eventually output one.

The result is a proper AUG since:

- The counter's runtime is polynomial in $|x|, 1/\epsilon$, it is called $2 * n$ times, so the total runtime is polynomial in $|x|, 1/\epsilon$
- $\forall$ SAT assignment $b_1, \ldots, b_n$,
  $\Pr[\text{output } b_1, \ldots, b_n] = \frac{F_{b_1}}{F_{b_1}+F_{\overline{b_1}}} \cdot \frac{F_{b_1 b_2}}{F_{b_1 b_2}+F_{b_1 \overline{b_2}}} \cdot \frac{F_{b_1 b_2 b_3}}{F_{b_1 b_2 b_3}+F_{b_1 b_2 \overline{b_3}}} \cdot \ldots = \frac{1}{F}$
- If the formula has no assignments both f_0 and f_1 will be zero and M will stop and return nothing

We proved that using an exact counter we can build a proper AUG. What if we use a FPRAS with parameter ϵ'? In this case each multiplicand in the equation will be multiplied by $(1 + \epsilon')^2$ at most (the numerator can be multiplies by $(1+\epsilon')$ at most and the denominator can be multiplied by $\frac{1}{(1+\epsilon')}$ at least), and by $\frac{1}{(1+\epsilon')^2}$ at least. So an approximate counter yields each satisfying assignment with probability $\forall y$ s.t. $(x,y) \in R$

$$\frac{1}{(1+\epsilon')^{2n}F} \leq \Pr[M \text{ outputs } y] \leq \frac{(1+\epsilon')^{2n}}{F}.$$

Thus, if we want an ϵ-FPRAS for uniform generation, then it is sufficient to call the uniform counting ϵ'-FPRAS where $\epsilon' < \epsilon/2n$.

4.2 Building a FPRAS from an AUG

We want to estimate the number of satisfying assignments using an almost uniform generator. We only need an accuracy of $(1+\epsilon)$ multiplication factor, so even if one of the sub-problems, say F_1, has only one member and the AUG will always return assignments which belong to F_0, this will still fit the required accuracy.

We will use an AUG to estimate the number of satisfying assignments:

- Pick k random satisfying assignments using the AUG
- Compute
  S_{b_1} = the fraction of samples beginning with b_1
  $S_{b_1 b_2}$ = the fraction of samples beginning with $b_1 b_2$
  ...
  $S_{b_1 b_2 \ldots b_n}$ = the fraction of samples beginning with $b_1 b_2 \ldots b_n$
- Estimate F via $\frac{F_{b_1}}{S_{b_1}}$ $(S_{b_1} \approx \frac{F_{b_1}}{F})$:
  $\hat{F} = \frac{\hat{F_{b_1}}}{S_{b_1}} = \frac{\hat{F_{b_1 b_2}}}{S_{b_1} S_{b_1 b_2}} = \frac{\hat{F_{b_1 b_2 b_3}}}{S_{b_1} S_{b_1 b_2} S_{b_1 b_2 b_3}} = \ldots = \frac{1}{S_{b_1} S_{b_1 b_2} \ldots S_{b_1 b_2 \ldots b_n}}$

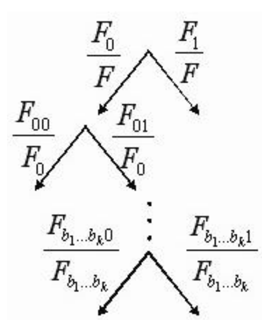

Figure 3: A "Random Walk" Illustration

This is a recursive process which simulates a "walk" on the recursion tree of the formula. In each step of the recursion the algorithm should choose a "direction"- on which sub-tree to continue the computation. We will choose $b_i \in \{0,1\}$ in each step such that maximizes the fraction of samples in its sub-tree $S_{b_1 \ldots b_i}$. In order to ensure a good estimation of F we need:

1. to reach a satisfying leaf - We will always reach a satisfying leaf (unless the formula is unsatisfiable) since we always recurse on the sub-tree with maximum samples.

2. $S_{b_1,\ldots,b_k}$ to be a good approximation - We always recurse on the sub-tree with maximum samples. When one side has probability $1/2 - \alpha$ and the other side has probability $1/2 + \alpha$ (for small α), we can show that the probability for sampling error is less then $e^{-2k\alpha^2}$ (by Chernoff bound). Therefore, for greater α we have smaller probability for picking the wrong sub-tree, and if α is very small then both sides has probability close to 1/2 and picking each side will generate good approximation.

The result is a FPRAS with $\epsilon' = \frac{\epsilon}{2n}$. ■

Conclusion: We showed a way to generate a random satisfying assignment for DNF. Using the above theorem we proved that we have an approximate algorithm for #DNF.