MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.265/15.070J Lecture 17 — Apr 19, SP17

Lecturer: Guy Bresler — Scribe notes by Chulhee Yun

Counting Problems

Content.

1. Counting problems
2. Counting colorings via sampling
3. Approximate counting and sampling
4. "All or nothing" theorem

The lecture is based on [Jerrum-Sinclair '89], [Jerrum-Valiant-Vazirani '86].

1 Correction from the last lecture

Theorem 1 (Continuous-time M.C. Mixing)**.** *Let $\{X_k\}$ be a (discrete-time) Markov chain with transition matrix P, N_t be a Poisson process of rate 1. Continuous-time chain is defined as $W_t \triangleq X_{N_t}$, and heat kernel H_t of W_t is defined as $H_t(x,y) \triangleq \Pr[W_t = y \mid W_0 = x]$.*

1. *If $d_{\mathrm{TV}}(P_k(x,\cdot),\pi) \le \epsilon$, then $d_{\mathrm{TV}}(H_t(x,\cdot),\pi) \le \epsilon$, for $t \ge k + O_\epsilon(\sqrt{k})$.*
2. *If $P(y,y) \ge \delta > 0$ for all y, $d_{\mathrm{TV}}(H_{t_0}(x,\cdot),\pi) \le \epsilon \Rightarrow d_{\mathrm{TV}}(P_k(x,\cdot),\pi) \le \epsilon + \epsilon'(\delta,t_0)$ where $k = \lfloor t_0/\delta \rfloor$.*

2 Counting problems

Definition 1 (#P)**.** *#P is the class of functions $f : \Sigma^* \to \mathbb{N}$ such that there exists polynomial-time non-deterministic Turing machine M which has $f(x)$ accepting computation paths on input x.*

Informally speaking, #P problems are counting problems associated with NP problems.

Examples. For the following NP problems,

- Does there exist a Hamiltonian cycle on G?
- Is there a SAT assignment for a conjunctive normal form (CNF) formula?

the corresponding #P problems are:

- How many Hamiltonian cycles are there on G?
- How many SAT assignments are there for a CNF formula?

It is known that #P problems are super hard. Toda's theorem says that #P problems are at least as hard as any problem in the entire polynomial hierarchy.

Definition 2 (fpras)**.** *Let $f \in \#P$, meaning that $f(x)$ is the number of solutions associated with x. A **fully-polynomial randomized approximation scheme (fpras)** for f is a randomized algorithm such that, on input (x, ϵ), outputs Z satisfying*

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

and runs in time $\mathrm{poly}(|x|, \epsilon^{-1})$.

Claim 1. *If there exists an fpras for f, then we can boost confidence to $1-\delta$ by running the algorithm $\log \delta^{-1}$ times and taking median.*

3 Counting colorings via sampling

We now consider the example of counting the number of q-coloring of graph via sampling. The input if the graph $G = (V, E)$ and the number of colors q. Let $n = |V|$, and $m = |E|$. The maximal degree of G is denoted Δ, and we assume that $q > 2\Delta + 1$. We learned from previous lectures that given a graph G, we can produce a uniformly random sample over all feasible q-colorings on G (e.g. Glauber dynamics).

Let us consider a sequence of graphs

$$G_0 = (V, E_0), G_1 = (V, E_1), \ldots, G_m = (V, E_m)$$

with increasing edge sets

$$E_0 = \emptyset, E_i = E_{i-1} \cup \{e_i\} \text{ for } i = 1, \ldots, m, \text{ and } E_m = E.$$

Note that any q-coloring on G_i is also valid for G_{i-1} because there are less restrictions in G_{i-1}.

Let $\Omega(G)$ denote the number of q-colorings on G. Then we have the following equality:

$$|\Omega(G)| = |\Omega(G_0)| \times \prod_{i=1}^{m} \frac{|\Omega(G_i)|}{|\Omega(G_{i-1})|},$$

where we have $|\Omega(G_0)| = q^n$ because there is no edge in G_0.

The ratio $\frac{|\Omega(G_i)|}{|\Omega(G_{i-1})|}$ can be estimated by sampling. We sample uniformly at random over q-colorings on G_{i-1}, and compute the portion of the samples that are also valid in G_i.

Claim 2. $\frac{|\Omega(G_i)|}{|\Omega(G_{i-1})|} \geq \frac{\Delta+1}{\Delta+2} \geq \frac{3}{4}$, *for* $\Delta \geq 2$.

Proof. Consider a coloring in $\Omega(G_{i-1}) \setminus \Omega(G_i)$. It is valid in G_{i-1} but not in G_i because there is a color conflict in the newly introduced edge $e_i = (k, l)$. If we change the color of k to a valid one, we can get a coloring in $\Omega(G_i)$. Since the maximal degree of G is Δ, there are at least $q - \Delta$ colors that k can choose. So,

$$|\Omega(G_i)| \geq (q - \Delta)|\Omega(G_{i-1}) \setminus \Omega(G_i)| \geq (\Delta + 1)(|\Omega(G_{i-1})| - |\Omega(G_i)|).$$

Arranging the inequality proves the claim. □

Now, we want to estimate each ratio $\frac{|\Omega(G_i)|}{|\Omega(G_{i-1})|}$ within factor $1 \pm \frac{\epsilon}{2m}$, so as to make sure that the total error is at most $(1 \pm \frac{\epsilon}{2m})^m \in (1 - \epsilon, 1 + \epsilon)$.

Define iid samples $Y_j = \mathbf{1}\{\text{random } q\text{-coloring of } G_{i-1} \text{ is valid for } G_i\}$. How many samples do we need? We use a Chebyshev inequality:

$$\Pr\left[\left|\frac{1}{t}\sum_{j=1}^{t} Y_j - \mathbb{E}[Y_1]\right| \geq \frac{\epsilon}{2m}\mathbb{E}[Y_1]\right] \leq \frac{\mathrm{var}(Y_1)}{\mathbb{E}[Y_1]^2}\frac{4m^2}{t\epsilon^2} \leq \frac{1}{\mathbb{E}[Y_1]}\frac{4m^2}{t\epsilon^2} \leq \frac{16m^2}{3t\epsilon^2}.$$

We take $t = \frac{cm^3}{\epsilon^2}$ for an appropriate constant c, then we can make the RHS of above inequality become $\leq \frac{1}{4m}$, so as to make sure that the total probability of error of all m estimates be less than or equal to $m \cdot \frac{1}{4m} = \frac{1}{4}$.

Finally the total run time is $O(m^4\epsilon^{-2}n\log n)$, where $O(n\log n)$ are for computing each sample, $O(m^3\epsilon^{-2})$ comes from t, and we have m ratio to estimate. If we sample one coloring from G_0 and continue by updating this for subsequent graphs in the sequence, then we can save a factor of m, and the total run time becomes $O(m^3\epsilon^{-2}n\log n)$.

4 Approximate sampling and counting

Theorem 2. *For all self-reducible NP problems, there exists an fpras for corresponding counting problem if and only if there exists a polynomial-time algorithm for approximate uniform sampling.*

This theorem states that (approximate) uniform sampling and (approximate) counting are exactly equivalent.

To see what a self-reducible NP problem is, consider an example of a SAT problem:

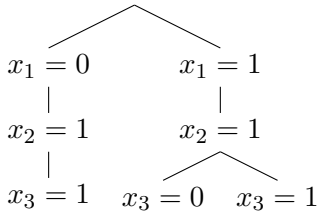
$$(\bar{x}_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_3)$$

Self-reducibility in SAP problem refers to the property that if we fix a variable, then we get a new SAT formula. For example, if we fix the values of x_1, we get

$$\begin{aligned} (1 \vee x_2) \wedge (0 \vee x_2) \wedge (0 \vee x_3) &= x_2 \wedge x_3 && \text{if } x_1 = 0 \\ (0 \vee x_2) \wedge (1 \vee x_2) \wedge (1 \vee x_3) &= x_2 && \text{if } x_1 = 1 \end{aligned}$$

which are again SAT formulas.

Using this property, we can draw a "self-reducibility tree" representing feasible assignments to the SAT formula. In this example, here is how the tree looks like:



Note that each leaf corresponds to a possible assignment for SAT formula, and each internal node is another SAT formula, where a subtree rooted at level i has variables $x_1, \ldots, x_i$ fixed.

Proof of Theorem. We now prove the theorem in the context of this tree. We do this by proving implications in each direction.

(Sampling $\Rightarrow$ fpras) In proving this, we make two simplifications:

- Exact sampling from uniform, rather than approximately uniform.
- Tree has branching factor of 2.

Let N be the number of solutions, which is equal to the number of leaves. Let N_1 be the number of leaves in the left subtree. Then define

$$r = \frac{\text{\# leaves in left subtree}}{\text{total \# of leaves}} = \frac{N_1}{N}.$$

Without loss of generality, we assume that $r \geq \frac{1}{2}$. Then we have $N = \frac{N_1}{r}$, so our estimate $\hat{N}$ for N is

$$\hat{N} = \frac{\hat{N}_1}{\hat{r}}.$$

Here, $\hat{r}$ is a thing we can estimate by sampling (take sample and count the number of samples in left subtree), and note that the left subtree is a smaller instance of a SAT problem, so we can calculate the estimate $\hat{N}_1$ by repeating similar steps down the tree.

For each level i subtree, we estimate r using sampling, where we assume again without loss of generality that $r \geq \frac{1}{2}$ all the time. From Chebyshev inequality we can get accurate estimates of r by using $t = O(m^3/\epsilon^2)$ samples for each level, we can get approximate estimate of N with total $O(m^4/\epsilon^2)$ samples, which is polynomial in tree depth m and ϵ^{-1}.

Therefore, we have an fpras.

(fpras $\Rightarrow$ sampling) For this part, assume that $\delta = 0$ in fpras; that is, we get an approximately close solution with probability 1.

Consider a tree with left and right subtrees. The total number of leaves are N, the number of leaves in left and right subtrees are N_l and N_r, respectively. To get a uniformly random sample, we want go down the tree until we reach a leaf. So what we would want to do is to move to left subtree with probability $\frac{N_l}{N_l+N_r}$ and right with probability $\frac{N_r}{N_l+N_r}$. As we repeat this recursively we will get to a leaf at the end of the day.

Unfortunately we can't get exact numbers N_l and N_r. Instead we have fpras to give us estimates $\hat{N}_l$ and $\hat{N}_r$. Then, we move to the left with probability $\frac{\hat{N}_l}{\hat{N}_l+\hat{N}_r}$ or to the right otherwise, and repeat this until we hit a leaf. If we control the relative error of each estimates to $(1 \pm \frac{\epsilon}{4m})$, at the end of the day the error of probability of sampling a leaf will be

$$\left(1 \pm \frac{\epsilon}{4m}\right)^{2m} \in (1 \pm \epsilon),$$

where $2m$ in the exponent comes from the fact that we estimate two things at each level.

In conclusion, we have an approximate uniform sampling algorithm. $\square$

5 "All or nothing" theorem

Theorem 3. *For a self-reducible problem, if there exists a polynomial-time randomized algorithm with approximation ratio* $[\text{poly}(|x|)]^{\pm 1}$*, then there exists an fpras.*

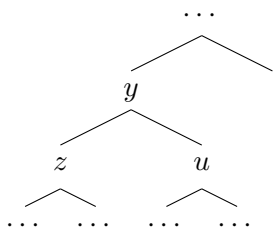
The ratio $[\text{poly}(|x|)]^{\pm 1}$ refers to any giant polynomial, for example $|x|^{10^{10}}$. Surprisingly, even if we start with this terrible algorithm, we can approximate as well as we want.

Corollary 4 ("All or nothing")**.** *For a self-reducible problem, either one of the following holds:*

1. *There exists an fpras.*
2. *There is no polynomial-time approximation algorithm to within any polynomial factor.*

It is indeed a very interesting dichotomy which we can't usually see in other types of problems.

Proof. The proof is by "boosting" the given algorithm to produce a random sample. Then by the theorem in the previous section this gives an fpras.



What we are going to do is to look at our self-reducibility tree and assign a weight to each edge by the estimate of the number of leaves in a subtree rooted at the down node of the edge. For example, the weight given to the edge between y and z is $w(y, z) = \hat{N}_z$, which is an approximate number of leaves in a subtree rooted at z. Similarly, $w(y, u) = \hat{N}_u$.

We now have an edge-weighted tree, and recall that the estimates may be terribly wrong. Instead of only moving downwards, here we do a weighted random walk on this tree, where the probability of choosing an edge is proportional to the weight. The intuition behind this is that when we encounter an edge with a giantic weight, we are likely to travel down the edge, but are also very likely to

move back up through the same edge. Turns out that this cancels the huge errors in the estimates.

The algorithm runs like this:

1. Run weighted random walk on the tree.
2. Take random node v from the stationary distribution π.
3. If the node v is a leaf, return v and terminate. If not, run again.

Let N_z be the number of leaves in the subtree rooted at z, and assume that $\alpha^{-1}N_z \leq \hat{N}_z \leq \alpha N_z$ where $\alpha = \text{poly}(|x|)$. Also assume that the weights on leaf edges are 1. Then, we finish the proof by proving the following three claims.

Claim 3. *The following are true:*

1. *The stationary distribution π is uniform over the leaves.*
2. *The total sum of probability assigned to leaves in the tree is large enough: it satisfies*
$$\pi(\{\text{leaves}\}) \geq \frac{c_l}{\alpha m}$$
where c_l is a constant.
3. *The random walk on the tree mixes fast: $t_{\text{mix}} = O(m^2\alpha^2 \log(\alpha m))$.*

Before proving the claims, let us see why these finish the proof of our theorem. If we run this random walk for t_{mix}, we get close to the stationary distribution π, which will give us distribution over nodes, and sample a node from that distribution. Part 2 tells us that we get a leaf with probability bigger than $1/\text{poly}(|x|)$, so we can get a leaf in polynomial time. Once we get a leaf, Part 1 says we are uniform over the leaves. So we get a (approximately) uniform sample over leaves in polynomial time. □

Proof of Claims. We prove each part of the Claim.

1. The stationary probability π of weighted random walk is given by
$$\pi(x) = \frac{d(x)}{d} \text{ where } d(x) = \sum_{y \sim x} w(y, x), \text{ and } d = \sum_x d(x).$$
Note that for all leaf nodes, $d(x) = 1$. Then $\pi(x)$ is constant for all leaf nodes x.

2. We have
$$d = \sum_x d(x) = \sum_{x \sim y} w(x, y) \leq 2\alpha m N,$$
where the factor 2 comes from the fact that we count each edge twice. Using this inequality,
$$\sum_{x:\text{leaves}} \pi(x) = \sum_{x:\text{leaves}} \frac{d(x)}{d} = \frac{N}{d} \geq \frac{1}{2\alpha m}.$$

3. We provide upper bound on the mixing time using multi-commodity flow. Consider two nodes x and y. Since the graph is a tree, there exists a unique path between x and y. The flow between x and y is equal to the demand: $\pi(x)\pi(y)$. Let l be the maximum length of path between two nodes, so $l \leq 2m$ in this case. For an edge $e = (u, v)$,
$$\begin{aligned} f(e) &\triangleq \text{total flw on edge } e, \\ c(e) &\triangleq \pi(u)P(u, v), \\ \rho &\triangleq \max_e \frac{f(e)}{c(e)}. \end{aligned}$$
Then, we have an upper bound on the mixing time:
$$t_{\text{mix}} = O\left(\rho l \log \pi_{\min}^{-1}\right).$$
Now, we actually compute the quantities $f(e)$ and $c(e)$ to get an upper bound on ρ. Consider $e = (z, y)$, where z is lower the vertex, and let T_z denote the subtree rooted at z. Recall that the edge e is assigned weight $\hat{N}_z$.
$$\begin{aligned} f(e) &= \sum_{u \in T_z, v \notin T_z} \pi(u)\pi(v) = \pi(T_z)\pi(T_z^c) \leq \pi(T_z) \\ &= \sum_{u \in T_z} \frac{d(u)}{d} \leq \frac{2\alpha m N_z}{d}. \\ c(e) &= \frac{d(u)}{d} \cdot \frac{\hat{N}_z}{d(u)} = \frac{\hat{N}_z}{d} \geq \frac{N_z}{\alpha d}. \end{aligned}$$
Thus we can bound above $\frac{f(e)}{c(e)}$. This holds for any e, so
$$\rho \leq 2\alpha^2 m.$$

Now, $\pi(\text{root}) = \frac{d(\text{root})}{d}$ is essentially the smallest among all nodes in the tree, and we have

$$d(\text{root}) \geq \alpha^{-1}N, d \leq 2\alpha mN \Rightarrow \frac{1}{2\alpha^2 m}.$$

Plugging these inequalities on the multi-commodity flow bound, we get

$$t_{\text{mix}} = O\left(\alpha^2 m^2 \log \alpha m\right).$$

□