Massachusetts Institute of Technology
6.856J/18.416J: Randomized Algorithms
David Karger

April 17, 2021

Problem Set 7 Solutions

Problem 1. **(a)** Suppose we make k estimations $p_1 \dots p_k$ of the true value p. For ease of exposition, suppose k is odd: $k = 2l - 1$. Note that the lth element among the sorted $p_1, \dots p_k$ is the median.

Let's compute the probability that the median is $< (1-\epsilon)p$. This happens only if at least l estimations are less than $(1-\epsilon)p$, which happens with probability at most $\sum_{i=l}^{k} \binom{k}{i} \frac{1}{4^i} (\frac{3}{4})^{k-i} = \sum_{i=l}^{k} \binom{k}{i} \frac{3^{k-i}}{4^k} \leq \sum_{i=l}^{k} \binom{k}{i} \frac{3^{k-l}}{4^k} \leq 2^k \cdot \frac{3^{k-l}}{4^k} \leq 3 \frac{3^{k/2}}{4^{k/2}} = 3(3/4)^{k/2}$.

The probability that the median is $> (1-\epsilon)p$ is similarly $\leq 3(3/4)^{k/2}$. Therefore, the probability that the median is not in $(1-\epsilon)p \dots (1+\epsilon)p$ is at most $6(\sqrt{\frac{3}{4}})^k$.

To get a probability of failure of at most δ, we require that $6(\sqrt{\frac{3}{4}})^k < \delta$, which implies that $k \geq \log(\delta/6)/\log(3/4) = \Theta(\log(1/\delta))$.

(b) Consider a distribution as follows: with probability 3/4, it is uniform on the interval $[-\varepsilon, \varepsilon]$, and with probability 1/4, it is 400. The mean of this distribution is 100. Thus the average of the samples will be heavily influenced by these outliers and hence is not reflective of the true value.

(c) Suppose the variable has mean μ and standard deviation $\sigma \leq \mu$ (so variance σ^2). It follows that a sum S of n samples has mean $n\mu$ and variance $n\sigma^2$. Now we apply the Chebyshev bound which tells us

$$\begin{aligned}\Pr([|S - n\mu| > \epsilon n\mu]) &\leq (n\sigma^2)/(\epsilon n\mu)^2 \\ &= (\sigma^2/\mu^2)/\epsilon^2 n \\ &\leq 1/\epsilon^2 n\end{aligned}$$

It follows that setting $n = 4/\epsilon^2$ we can take the probability of this deviation below 1/4. This means S fits the conditions of the previous part, so we can conclude that $O(\ln 1/\delta)$ samples of S (each involving $O(1/\epsilon^2)$ samples of the underlying variable) suffice for an (ϵ, δ)-approximation.

(d) We start from the Chebyshev bound in (c), however, we need to consider that the assumption $\sigma \leq \mu$, i.e., $r \leq 1$, does not hold anymore. In particular,

$$\begin{aligned}\Pr([|S - n\mu| > \epsilon n\mu]) &\leq (n\sigma^2)/(\epsilon n\mu)^2 \\ &= (\sigma^2/\mu^2)/\epsilon^2 n \\ &\leq r/\epsilon^2 n.\end{aligned}$$

Setting $n = 4r/\epsilon^2$ we can take the probability below 1/4, and use the median approach from (a) to conclude that $O(r\epsilon^{-2} \ln 1/\delta)$ samples suffice to obtain an (ϵ, δ)-approximation.

Problem 2. As in the transitive closure estimation in the class, we will be sampling. In other words, for each v we want to sample "destinations" until we get $M = O((\log n/\epsilon^2) \cdot \log(1/\delta))$ samples that are at distance at most d from v. As in the class, from this, we can estimate the total number of nodes at distance d from v with high probability.

We parallelize this for all vertices v in total. That is, take around $O(nM)$ samples of "destinations", and do backwards BFS to find all the vertices v such that the distance from v to the sample is at most d. This, so far, is pretty much as in the transitive closure algorithm.

Again, we would want to delete an edge when we've delivered (through backwards BFS) M samples through it. We need to be more attentive here, however. Instead of having a unique counter per edge saying how many samples back-BFS'ed through it, we keep d counters per edge. For edge e, the counter $c_i^{(e)}$, $1 \le i \le d$, is increased when the e is at level i in the backwards BFS tree (which has at most d levels of edges going down). Once we've reached M for a counter $c_i^{(e)}$, we do not back-propagate on that edge anymore whenever e is at the ith level. (Technically, imagine we have d copies of each edge, for each value of i. Then we delete ith copy of e whenever $c_i^{(e)}$ becomes M.)

Now, note that for each "source" v, if the current sample is at distance at most d, then this sample will be counted towards v's counter (unless, v already has M samples). Consider current sample, which is at some distance $k \le d$ from v. Take the path from v to s; look at the respective counters of the edges: all of them have to be less than M if the counter of v is less than M.

Thus, we do expected nM BFS backpropagations. However, we do only $O(dM)$ backpropagations per edge. Therefore, in total, we can make only $O(mdM)$ backprogations on edges. Running time is $O((m+n)M) = \tilde{O}(((m+n)/\epsilon^2) \cdot \log 1/\delta)$.

Problem 3. **(a)** For any $\mathbf{x} = (x_1, \ldots, x_m) \in \{+1, -1\}^m$, we have that,

$$\langle h, \mathbf{x} \rangle^2 \quad = \quad \left(\sum_{i=1}^{m} h_i x_i \right)^2 \quad = \quad \sum_{i=1}^{m} h_i^2 + 2 \sum_{i<j} h_i h_j x_i x_j \ .$$

Thus, we get that,

$$\mathbb{E}_x[\langle h, \mathbf{x} \rangle^2] \quad = \quad ||h||_2^2 + 2 \sum_{i<j} h_i h_j \mathbb{E}_x[x_i x_j] \quad = \quad ||h||_2^2$$

where, the last equality follows because $\mathbb{E}_x[x_i x_j] = 0$. Similarly we have that,

$$\begin{aligned}
\mathrm{Var}_{\mathbf{x}}\left[\langle h, \mathbf{x}\rangle^2\right] &= \mathbb{E}_{\mathbf{x}}\left[\left(\langle h, \mathbf{x}\rangle^2 - \mathbb{E}_{\mathbf{x}}[\langle h, \mathbf{x}\rangle^2]\right)^2\right] \\
&= \mathbb{E}_{\mathbf{x}}\left[\left(2\sum_{i<j} h_i h_j x_i x_j\right)^2\right] \\
&= 4\sum_{\substack{i_1<j_1 \\ i_2<j_2}} h_{i_1} h_{j_1} h_{i_2} h_{j_2} \mathbb{E}_{\mathbf{x}}\left[x_{i_1} x_{j_1} x_{i_2} x_{j_2}\right] \\
&= 4\sum_{i<j} h_i^2 h_j^2 \\
&\le 2\left(\sum_i h_i^2\right)^2 \quad = \quad 2||h||_2^4
\end{aligned}$$

where, we use that $\mathbb{E}_{\mathbf{x}}\left[x_{i_1} x_{j_1} x_{i_2} x_{j_2}\right] = 1$ if $i_1 = i_2$ and $j_1 = j_2$, and 0 otherwise.

(b) We use the randomness to obtain several vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(q)} \in \{+1, -1\}^m$. We use the space of our algorithm to store a vector $v \in \mathbb{Z}^q$ (where all entries of v lie in $[-N, N]$, and hence the space needed to store v is only $O(q \log N)$.

On receiving element i, we add the vector $(x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(q)})$ to v.

At the end of the stream, we simply output $\frac{1}{q} \cdot \sum_{j=1}^{q} v_j^2$.

Let $h \in \mathrm{Z}_{\ge 0}^m$ be the underlying histogram vector. We note that $v_j = \left\langle h, \mathbf{x}^{(j)}\right\rangle$. And hence,

$$\mathbb{E}_{(\mathbf{x}_i^{(1)}, \ldots, \mathbf{x}_i^{(q)})}\left[\frac{1}{q}\sum_{j=1}^{q} v_j^2\right] = ||h||_2^2 \qquad \text{and} \qquad \mathrm{Var}_{(\mathbf{x}_i^{(1)}, \ldots, \mathbf{x}_i^{(q)})}\left(\frac{1}{q}\sum_{j=1}^{q} v_j^2\right) = \frac{2||h||_2^4}{q}$$

Using Chebyshev's inequality, we get that,

$$\Pr\left[\left|\frac{1}{q}\sum_{j=1}^{q} v_j^2 - ||h||_2^2\right| > \varepsilon ||h||_2^2\right] \quad \le \quad \frac{2}{\varepsilon^2 q}$$

Thus, if we choose $q = \frac{8}{\varepsilon^2}$, we get an estimate of $||h||_2^2$ up to multiplicative $(1 \pm \varepsilon)$ error with probability at least $3/4$. Taking the median of $O(\log 1/\delta)$ such estimators reduces the failure probability to δ. The space used is $O_{\varepsilon,\delta}(\log N)$.

(c) We simply observe that the analysis in part(a) works as long as $\mathbf{x}$ is 4-wise independent. We can sample such an $\mathbf{x}$ using $O(\log m)$ bits of randomness. Since we are running $O_{\varepsilon,\delta}(1)$ such experiments in parallel, we need $O_{\varepsilon,\delta}(\log m)$ additional memory to store the randomness. Thus, overall the algorithms runs in $O_{\varepsilon,\delta}(\log N + \log m)$ space.

Problem 4. Consider the set of all (a, i), where a is an assignment and i is an index from 1 to m (number of clauses). Denote by O the set of all (a, i), where a satisfies clause i. Denote by X the set of all (a, i) where a is satisfiable and i is the smallest index of a satisfied clause. Consider a measure space Σ over all (a, i) with the following measure: (a, i) has the probability of $\frac{1}{m}p_a$, where p_a is the probability of choosing assignment a (when each variable has probability p of being set to 1.).

If A is the probability of getting a satisfying assignment (the number we want to estimate), then note that $\Pr_{\sigma\in\Sigma}[\sigma \in X] = A/m$. Further on, I will denote $\Pr[X] = \Pr_{\sigma\in\Sigma}[\sigma \in X]$. Also, $\Pr[O] = \Pr_{\sigma\in\Sigma}[\sigma \in O]$. Therefore,

$$\begin{aligned} A &= m \cdot \Pr[X] \\ &= m \cdot (\Pr[X \mid O] \cdot \Pr[O] \ + \ \underbrace{\Pr[X \mid \neg O]}_{=\,0} \cdot \Pr[\neg O]) \\ &= m \cdot \Pr[X \mid O] \cdot \Pr[O] \end{aligned}$$

We need to estimate $\Pr[X \mid O]$ and $\Pr[O]$.

Computing $\Pr[O]$ is easy: for each clause C_i, compute $q_i = \Pr_a[a \text{ satisfies } C_i]$ (this is the probability of choosing the variables in C_i right). Then $\Pr[O] = \sum_{i=1}^m q_i/m$.

Computing $\Pr[X \mid O]$ can be done in an approximate way. Specifically, we will generate many samples from O (according to measure Σ; that is, $\sigma \in O$ have original probabilities scaled by $1/\Pr[O]$). This can be done by first picking a clause C_i with probability $q_i/\sum_{j=1}^m q_i$. We then sample the unfixed variables in C randomly, setting them to 1 with probability p and 0 with probability $1 - p$.

Thus, we need to estimate the 0/1 function $\mathbf{1}_{[\sigma\in X]}$ where σ is generated from O. We can estimate the mean of this function using again $O(\frac{m}{\epsilon^2}\log 1/\delta)$ samples (note that $\Pr[X \mid O]$ is at least $\frac{1}{m}$ since to each $\sigma = (a, i) \in X$ there are at most m pairs $(a, j) \in O$, for $j \in \{1 \ldots m\}$, and all pairs (a, j) have the same probability as (a, i)).

This gives $1 \pm \epsilon$ approximation with probability $\geq 1 - \delta$.

Problem 5. **(a)** Let U be the disjoint union (multiset) of satisfying assignments for each clause (i.e., $|U| = N$); and let S be the satisfying assignments (i.e., $|S| =$DNF-count). Then, $N \cdot \mathbb{E}[X_t] = N\sum_{a\in U}\frac{1}{N}\frac{1}{c_a} = \sum_{a\in U}\frac{1}{c_a} = \sum_{a\in S}\frac{c_a}{c_a} = |S|$ (since an assignment a that satisfies c_a clauses, contributes to the sum $1/c_a$ per each satisfied clause).

(b) Let $q = O(m/\epsilon^2 \log 1/\delta)$ be the number of samples of X_t. Our estimate is $T = N\frac{\sum X_t}{q}$. Note $\mathbb{E}[T] = S$. We need to estimate $\Pr[|N\sum X_t/q - S| \geq \epsilon S] \leq \Pr[|\sum X_t - qS/N| \geq \epsilon\frac{qS}{N}]$. Since $\mathbb{E}[\sum X_t] = \frac{qS}{N}$ and X_t are independent variables in $[0, 1]$, we can apply Chernoff. Thus, $\Pr[|N\sum X_t/q - S| \geq \epsilon S] \leq \exp[-\epsilon^2\frac{qS}{N}/3] \leq \exp[-O(\log 1/\delta)] < \delta$ (since $S/N \geq 1/m$ and with the appropiate choice of constants).

Our estimate is $1 \pm \epsilon$ of S with probability $\geq 1 - \delta$.

(c) Just sample clauses from assignment a until we get $\mu_{\epsilon\delta'} = O(\log 1/\delta'\epsilon^{-2})$ hits (into clauses that are satisfied by a). With probability $1-\delta'$, we have $m\frac{\mu_{\epsilon\delta'}}{f} \in (1\pm\epsilon)c_a$, where f is the total number of samples we did (as stated by the lecture on sampling).

Expected running time is $O(\frac{m}{c_a}\log 1/\delta'\epsilon^{-2})$. With high probability, we will make $O(\frac{m}{c_a}\log^2 1/\delta'\epsilon^{-2})$ samples in total.

Moreover, the overall estimator is as follows. Sample a's. For the sampled a's, estimate c_a, then let $X_t = 1/\hat{c}_a$ ($\hat{c}_a$ is the estimation of c_a). Sum X_t, compute mean, multiply by N to get estimate of S. Note that the estimator for c_a is right with high probability every time for an appropriate choice of δ'. We get our estimation $\hat{S}$ within $(1\pm\epsilon)\cdot(1\pm\epsilon)^{-1} \in (1\pm O(\epsilon))$ of S (first $1\pm\epsilon$ is from estimator (b), and second from estimators (c)).

(d) Let's see how to make it fast. In the following, by "time" I will usually mean # of sampled/estimated clauses (unless specified otherwise).

Note that in (b), we don't need $O(m/\epsilon^2 \log 1/\delta)$ samples of a, but only $O(\frac{N}{S}/\epsilon^2\log 1/\delta)$. For every time we sample a, we need to estimate c_a. Thus we need $O(\frac{N}{S}/\epsilon^2\log 1/\delta)$ calls to our estimator from (c), which fails with probability δ'. By a union bound argument, we can set $\delta' = O(\delta\frac{S}{N}\epsilon^2\log^{-1} 1/\delta)$ such that with probability δ none of the estimators from (c) fail.

Note that expected running time is then $O(\frac{N}{S}/\epsilon^2\log 1/\delta)\cdot\mathbb{E}[O(\frac{m}{c_a}/\epsilon^2\log 1/\delta')] = O\left(\frac{mN}{S}/\epsilon^4\log 1/\delta\log(1/\delta\frac{N}{S}\epsilon^{-2}\log 1/\delta)\mathbb{E}[1/c_a]\right) = O_{\epsilon,\delta}(\frac{mN}{S}\log\frac{N}{S}\mathbb{E}[1/c_a]) = O_{\epsilon,\delta}(m\log m)$ (where $O_{\epsilon,\delta}$ hides $\text{poly}(1/\epsilon)$ and $\text{poly}\log(1/\delta)$ factors).

To compute the actual runtime (whp), we need to consider that our estimator in (c) needs to see at least $\mu_{\epsilon\delta'}$ hits, and thus the runtime might be much larger than the expected runtime computed above. However, when we run each one of the estimators from (c) at most $O(\log 1/\delta')$ times their expected time, they all find a good estimate with overall probability δ by the union bound argument above and our choice for δ'. Therefore, again whp, all estimators (c) run in time at most $O(\log 1/\delta')$ of their expected time. Thus, whp, the entire algorithm should run in $O_{\epsilon,\delta}(m\log m\log 1/\delta') = O_{\epsilon,\delta}(m\log^2 m)$ time. Therefore, the probability that we fail the algorithm by not running it sufficiently long is only 2δ. To see this note that there is a δ failure probability that any of the estimators for c_a fail and a δ failure probability that the estimator $\sum X_t$ fails. Then apply union bound.

Considering that a clause has z variables, we have a total (real) running time equal to $O_{\epsilon,\delta}(mz\log^2 m)$.

This is compared to $O_{\epsilon,\delta}(m^2 z)$ for the algorithm in class, since our algorithm from class runs in $O_{\epsilon,\delta}(m)$ times the formula size and the overall formula size is $O(mz)$.

Note: A very common mistake on this problem was to analyze the running time of this algorithm by directly multiply the answers from parts (b) and (c)—i.e. (expected number of clauses)*(expected time per clause). Remember that this is not a valid argument because the time per clause and the number of clauses are *dependent* random variables, and in this case the expectation of their product is not necessarily equal to the product of their expectations.