# COS 521: Advanced Algorithms
# Homework 1

Zachary Hervieux-Moore

Friday 30th September, 2016

**Exercise 1:** The simplest model for a *random graph* consists of $n$ vertices, and tossing a fair coin for each pair $\{i, j\}$ to decide whether this edge should be present in the graph. Call this $G(n, 1/2)$. A $k$-clique is a set of $k$ vertices with an edge between every pair. What is the expected number of $k$-cliques? What is the variance? Try to use Chebyshev inequality to show that the number is concentrated around the expectation and give an expression for the exact decay in probability. Can you guess the size of the largest clique in a random graph from the above estimates?

**Answer:** Pick any $k$ vertices in the graph, the probability that it is a $k$-clique is exactly:

$$\left(\frac{1}{2}\right)^{\binom{k}{2}}$$

Since each pair of vertices must be joined together. The new problem becomes binomial with $\binom{n}{k}$ trials each with success probabilty above. Thus, the mean is:

$$\mu = \mathbb{E}[X_k] = \binom{n}{k}\left(\frac{1}{2}\right)^{\binom{k}{2}}$$

Where $X_k$ is the number of cliques of size $k$. It's variance is bounded by the independent case $np(1-p)$,

$$Var[X_k] \leq \binom{n}{k}\left(\frac{1}{2}\right)^{\binom{k}{2}}\left(1-\left(\frac{1}{2}\right)^{\binom{k}{2}}\right)$$

Chebyshev's Inequality in this case becomes:

$$P(|X_k - \mu| \geq c) \leq \frac{\binom{n}{k}\left(\frac{1}{2}\right)^{\binom{k}{2}}\left(1-\left(\frac{1}{2}\right)^{\binom{k}{2}}\right)}{c^2}$$

Now working with this inequality,

$$\frac{\binom{n}{k}\left(\frac{1}{2}\right)^{\binom{k}{2}}\left(1-\left(\frac{1}{2}\right)^{\binom{k}{2}}\right)}{c^2} \leq \frac{\binom{n}{k}\left(\frac{1}{2}\right)^{\binom{k}{2}}}{c^2} \leq \frac{\left(\frac{en}{k}\right)^k\left(\frac{1}{2}\right)^{\left(\frac{k^2}{4}\right)}}{c^2}$$

Note that,

$$\left(\frac{en}{k}\right)^k \left(\frac{1}{2}\right)^{\left(\frac{k^2}{4}\right)} \sim \left(\frac{n}{k}\right)^k \left(\frac{1}{2}\right)^{k^2} = 2^{k\log(n)-k^2/4-k\log(k)}$$

Thus, picking,

$$c = \sqrt{2^{k\log(n+1)-k^2/4-k\log(k)}}$$

We have, $\frac{1}{n}$ decay. Picking $k = \log(n)$ we see that,

$$c = \sqrt{2^{\log(n)\log(n+1)-\log(n)^2/4-\log(n)\log(\log(n))}}$$

Which $c \to 0$ as $n \to \infty$. Thus, the largest clique size with high probability is $\log(n)$.

**Exercise 2:** Note that the second question on the homework has been postponed to the next homework. So we continue from 3 on the homework.

Show that given $n$ numbers in $[0,1]$ it is impossible to estimate the *value* of the median within say 1.1 factor with $o(n)$ samples. (Hint: to show an impossibility result you show that two different sets of $n$ numbers that have very different medians but which generate - whp - identical samples of size $o(n)$.)

Now calculate the sample size needed (as a function of $t$) so that the following is true: with high probability, the median of the sample has at least $n/2-t$ numbers less than it and at least $n/2+t$ numbers more than it.

**Answer:** Define the two sets as follows:

$$N_1 = \{0, \frac{1}{3\lfloor\frac{n}{2}\rfloor}, \frac{2}{3\lfloor\frac{n}{2}\rfloor}, \dots, \frac{n-1}{3\lfloor\frac{n}{2}\rfloor}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, 1-\frac{n-1}{3\lfloor\frac{n}{2}\rfloor}, \dots, 1-\frac{1}{3\lfloor\frac{n}{2}\rfloor}, 1\}$$

$$N_2 = \{0, \frac{1}{3\lfloor\frac{n}{2}\rfloor}, \frac{2}{3\lfloor\frac{n}{2}\rfloor}, \dots, \frac{n-1}{3\lfloor\frac{n}{2}\rfloor}, \frac{1}{3}, \frac{3}{5}, \frac{2}{3}, 1-\frac{n-1}{3\lfloor\frac{n}{2}\rfloor}, \dots, 1-\frac{1}{3\lfloor\frac{n}{2}\rfloor}, 1\}$$

Heuristically, pick many evenly spaced point from [0,1/3] and [2/3,1] and include a middle. The only difference between $N_1$ and $N_2$ is their middle point. Thus, as $n \to \infty$, the probability of the middle point vanishes and so the samples will be identical whp. However, the true median of $N_1$ is $\frac{1}{2}$ and the median of $N_2$ is $\frac{3}{5}$ which differ by a factor greater than 1.1.

Now, to prove the other bound. Order the elements in bins labelled from $\{1,\dots,n\}$ and place a ball in bin $i$ if it is sampled. We calculate that a certain bin is the median that satisfies the requirements:

$$\begin{aligned}
P(\text{bin}_i \text{ is median}) &= P(n/2-t \text{ balls in bins } < i) \cap P(i \text{ is picked}) \\
&\quad \cap P(n/2-t \text{ balls in bins } > i) \\
&= \frac{(i-\frac{n}{2}-t)!}{i!} \cdot \frac{1}{n} \cdot \frac{(n-i-\frac{n}{2}-t)!}{(n-i)!} \\
&= \frac{1}{n} \cdot \frac{\binom{n}{i}}{\binom{n}{i-\frac{n}{2}-t}} \geq \frac{1}{n} \cdot \frac{(\frac{n}{i})^i}{(\frac{n}{i-\frac{n}{2}-t})^{(i-\frac{n}{2}-t)}}
\end{aligned}$$

$$\geq \frac{1}{n} \cdot \frac{(\frac{n}{i})^i}{(\frac{n}{i-\frac{n}{2}-t})^i} = \frac{1}{n} \cdot \left(\frac{i-\frac{n}{2}-t}{i}\right)^i = \frac{1}{n} \cdot \left(1 - \frac{\frac{n}{2}-t}{i}\right)^i$$

Note that $\frac{n}{2}-t < i < \frac{n}{2}+t$ We see that as $n \to \infty$ the above $\sim \frac{1}{n}$ Summing over all $i$'s yields that the probability of the sample of the median is $> \frac{t}{n}$ and so we should pick $f(t) = \frac{n}{t}$ to ensure the condition whp.

**Exercise 3:** Consider the following process for matching $n$ jobs to $n$ processors. In each step, every job picks a processor at random. The jobs that have no contention on the processors they picked get executed, and all other jobs *back off* and then they try again. Jobs only take one round of time to execute, so in every round all the processors are available. Show that all jobs finish executing whp after $O(\log\log n)$ steps.

**Answer:** Let $X_i$ represent if job $i$ collides with a previous job. $X_i = 1$ if it collides, 0 otherwise. Then, $X = \sum_i X_i$ is the number of unprocessed jobs. We now come up with upper bounds for all the $X_i$,

$$P(X_1) = 0$$
$$P(X_i) = \frac{i-1}{n} \text{ the worst case is that all the other jobs didn't collied}$$

Now define a new set of RV's to achieve independence. $Y_i$ is Bernoulli with probability $\frac{i-1}{n}$. Thus, $Y = \sum_i Y_i$ dominates the RV $X$. Now we apply Markov's inequality to $Y$,

$$E[Y] = \sum_{i=1}^{\alpha n} E[Y_i] = \sum_{i=1}^{\alpha n} \frac{i-1}{n}$$
$$= \frac{(\alpha n)(\alpha n - 1)}{2n} \leq \frac{\alpha^2 n}{2}$$

Note that this implies $P(Y \geq \frac{\alpha^2 n}{2}) \leq P(Y \geq 5E[Y])$ Now we can apply Chernoff,

$$P(Y \geq 5E[Y]) \leq \left(\frac{e^{-4}}{5^5}\right)^{\frac{\alpha^2 n}{2}} \leq e^{-(\alpha^2 n)}$$

Thus $P(X \geq 5E[X]) \leq e^{-(\alpha^2 n)}$ which converges when $\alpha^2 n \geq \log n$. Thus, $\alpha n$ shrinks at an exponential rate as long as $\alpha^2 n \geq \log n$ whp. This means the runtime is $\log\log n$.

When $\alpha^2 n < \log n$, the probability of collisions are very small and the probability of no collions is very high. Note: I received guidance from http://people.csail.mit.edu/moitra/docs/6854hw1.pdf

**Exercise 4:** In class, we saw that the Karger-Stein algorithm to find a min-cut in a graph which can be implemented in $O(n^2 \log^2(n))$ time (you can assume this bound on the running time in your solutions). Modify the algorithm to find *all* min-cuts in a graph. (Hint: Run Karger-Stein procedure a few times and keep track of all the min-cuts produced).

**Answer:** We showed in class that the probability that Karger-Stein finds a min cut is on the order of $P(n) = O(\frac{1}{\log n})$. Thus, if we run this algorith for $O(\log^2 n)$ times, we have that the probability that it misses a min-cut is:

$$\begin{aligned} P(\text{misses a min-cut}) &= (1 - P(n))^{c\log^2 n} \\ &= \left(\left(1 - \frac{1}{\log n}\right)^{\log n}\right)^{c\log n} \leq e^{-c\log n} \end{aligned}$$

Picking our coefficients carefully, we set $c = 3$. Thus,

$$e^{-c\log n} = e^{-3\log n} = \frac{1}{n^3}$$

In the worst case, the graph is a cycle and there are $\binom{n}{2}$ min-cuts. Thus,

$$P(\text{misses any min-cut}) \leq \binom{n}{2}\frac{1}{n^3} = \frac{1}{n}$$

Hence, with high probability, we find all the min-cuts by running the Karger-Stein procedure an additional $O(\log^2 n)$ times. Up until this point, the procedure only takes $O(n^2 \log n)$. So the total run time is $O(n^2 \log^3 n)$. One just needs to stote all the min-cuts as you perform the algorithm.

**Exercise 5:** A cut is said to be a $B$-*approximate min cut* if the number of edges in it is at most $B$ times that of the minimum cut. Show that a graph has at most $(2n)^{2B}$ cuts that are $B$-approximate. (Hint: Run Karger's algorithm until it has $2B+1$ supernodes. What is the chance that a particular $B$-approximate cut is still available? How many possible cuts does this collapsed graph have?)

**Answer:** If we run Karger's algorithm until $2B+1$ supernodes are left, the chances that a $B$-approximate min cut survives is:

$$\begin{aligned}
&P(C \text{ survives until } 2B+1 \text{ nodes left})\\
&\qquad = \left(1-\frac{2}{n}\right)\cdot\left(1-\frac{2}{n-1}\right)\cdot\ldots\cdot\left(1-\frac{2}{2B+1}\right)\\
&\qquad \geq \frac{\binom{2B+1}{2}}{\binom{n}{2}} = \frac{\frac{2B+1}{(2B-1)!2!}}{\frac{n!}{(n-2)!2!}} = \frac{(2B+1)!(n-2)!}{n!(2B-1)!}\\
&\qquad \geq 1/\binom{n}{2B+1}
\end{aligned}$$

Now we pick a random cut in the remaining graph,

$$P(C \text{ survives a final random cut}) = \frac{1}{2^{2B+1}}$$

Since there are $2B+1$ nodes and we randomly pick from the power set to create the partition. So,

$$\begin{aligned}
P(C \text{ survives}) &\geq 1/\binom{n}{2B+1}\cdot P(C \text{ survives a final random cut})\\
&= 1/\binom{n}{2B+1}\cdot\frac{1}{2^{2B+1}} \geq \frac{1}{\frac{n^{2B+1}}{(2B+1)!}}\cdot\frac{1}{2^{2B+1}}\\
&= \frac{1}{n^{2B+1}}\cdot\frac{(2B+1)!}{2^{2B+1}} \geq \frac{1}{n^{2B+1}}
\end{aligned}$$

Where the last inequality comes from the fact that $\frac{x!}{2^x} > 1$ for $x$ sufficiently large ($x > 5$). We conclude from this that the number of $B$-approximate min cuts is no more than $n^{2B+1}$.

**Exercise 6:** In MATLAB or another suitable programming environment implement a pairwise independent hash function and use it to map

$\{100, 200, 300, \ldots, 100n\}$ to a set of size around $n$. (Use $n = 10^5$ for starters.) Report the largest bucket size you noticed. Then make up a hash function of your own design (could involve crazy stuff like taking XOR of bits, etc.) and repeat the experiment with it and report the largest bucket size. Include your code with you answer and brief description of any design decisions.

**Answer:** The MATLAB code is posted below. Using a pairwise independent hash function, the largest bucket size was 5 which was quite astonishing since the size of the hash was very close to 1/5 the size of the elements hashed. This ran in roughly 0.85 seconds. I wanted to improve the speed so I generated a list of random numbers and XORed these numbers bitwise with the incoming elements. The hope was that the bitwise XOR is a faster operation than the modulus operator which must run Euclidean's algorithm to computer the modulus. Note, that in my custom hash, the modulus must be computed only if the hash maps to something greater than $p$, the size of the hash table. This produced a maximum bucket size of 22 and ran in twice the time as the other hash.

```
clear;
clc;

p = 206483;
n = 1000000;
x = linspace(100, 100*n, n);

array_hash = zeros(1, p);

start = tic;

for i = x
    % Hash functions
    h = mod(100003*i + 739967, 206483);

    % Keep track of frequency
    array_hash(h+1) = array_hash(h+1) + 1;
end

time = toc(start);

sprintf('Largest bucket size of independent hashes is: %0.0f', max(
    array_hash))
sprintf('Time elapsed: %0.5f', time)

rng(0,'twister');
r = randi([0 p],1,n);
```

```
array_hash = zeros(1, p);

start = tic;

for i = x
    % Hash functions
    h = bitxor(r(i/100)+1,i);
    if h > p
        h = mod(h,p);
    end

    % Keep track of frequency
    array_hash(h+1) = array_hash(h+1) + 1;
end

time = toc(start);

sprintf('Largest bucket size of custom hashes is: %0.0f', max(array_hash
    ))
sprintf('Time elapsed: %0.5f', time)
```