Midterm Solution Sketches

- Don't Panic.
- The midterm contains six problems (and one just for fun). You have 100 minutes to earn 100 points.
- The midterm contains 18 pages, including this one and 4 pages of scratch paper.
- The midterm is closed book. You may bring one double-sided sheet of A4 paper to the midterm. (You may not bring any magnification equipment!) You may not use a calculator, your mobile phone, or any other electronic device.
- Write your solutions in the space provided. If you need more space, please use the scratch paper at the end of the midterm. Do not put part of the answer to one problem on a page for another problem.
- Read through the problems before starting. Do not spend too much time on any one problem.
- Show your work. Partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- You may use any algorithm given in class without restating it—simply give the name of the algorithm and the running time. If you change the algorithm in any way, however, you must provide complete details.
- Good luck!

Problem #	**Name**	**Possible Points**	**Achieved Points**
1	A Few Random Variables	12	
2	Independence Day	14	
3	Frenemy Fred	14	
4	Right in the Middle	20	
5	Building Strong Bonds	20	
6	Network Robustness	20	
Total:		100	

Student Number: _____

Problem 1. A Few Random Variables

Assume you have a collection of indicator random variables $x_1, x_2, \ldots, x_n$. For each x_i, the $\Pr[x_i = 1] = p_i$ (and otherwise $x_i = 0$). The variables are *not independent*, and you do not know anything about how they are correlated. Which of the following statements are *always* true? For each part, explain why in less than ten words.

$$\Pr\left[\sum_{i=1}^{n} x_i \geq 1\right] \leq \sum_{j=1}^{n} p_j$$

TRUE **FALSE**

Solution: True. This is the union bound, i.e., the inequality holds if any one of the random variables is one.

$$\Pr\left[\sum_{i=1}^{n} x_i \leq 0\right] \leq \prod_{j=1}^{n} (1 - p_j)$$

TRUE **FALSE**

Solution: False. This would be true if the variables were independent, but they are not. For example, assume that with probability 1/2, all the x_i are 1, and otherwise all are 0.

$$\Pr\left[\sum_{i=1}^{n} x_i \geq \sum_{j=1}^{n} \frac{p_j}{2}\right] \leq \frac{1}{2}$$

TRUE **FALSE**

Solution: False. This is almost Markov's Inequality, but not quite.

$$\Pr\left[\sum_{i=1}^{n} x_i \geq (1+\epsilon)\sum_{j=1}^{n} p_j\right] \leq e^{-(\epsilon^2/3)\sum_{j=1}^{n} p_j}$$

Solution: False. This is a Chernoff bound, but it is not true because the variables are not independent. False. This would be true if the variables were independent, but they are not. Again, consider the example where with probability 1/2, all the x_i are 1, and otherwise all are 0.

TRUE **FALSE**

Problem 2. Independence Day

Recall that the L0-sampler is constructed from a collection of $\log n$ distinct s-sparse samplers. To decide which indices are mapped to which samplers, we use $\log n$ hash functions $h_0, h_1, \ldots, h_{\log n - 1}$ where $h_i(x) = 1$ with probability $1/2^i$ (and 0 otherwise). What happens if these hash functions are not independent?

Consider, instead, using a hash function $h : [1, n] \to [1, n]$ that maps to a random integer from 1 to n. I.e., for every index k and every integer $\ell \in [1, n]$ (independently for each k):

$$\Pr[h(k) = \ell] = 1/n \ .$$

Then we define: $h_i(k) = 1$ if $h(k) \leq n/2^i$. In this case, each hash function h_i has the proper probability distribution, but they are not independent.

If we use the h_i defined in this manner, does the L0-sampler still work as expected? If so, why? If not, why not? (Circle your answer and then explain.)

WORKS **FAILS**

Explanation:

Solution: Works! In this case, everything continues to work exactly as described in class. Notice that correlations among the s-sparse do not matter. In the analysis, we proved that there exists a good s-sparse sampler (that will return a non-zero index). The behavior of the other s-sparse samplers is irrelevant.

Problem 3. Frenemy Fred.

To pass the time, you create an internet service[1] that generates a stream of integers. You have hidden a secret pattern in the integers, and you are waiting for someone to find it! You notice that your frenemy Fred is monitoring the stream of integers, trying to figure out your code. Bored, you hack into his computer and discover that Fred is about to start running a Flajolet-Martin sketch to count the number of distinct items in your stream for the month of November. And you find the hash function that he will be using! Here is an excerpt from Fred's hash function:

1	2	3	4	5	6	7	8	9	10
0.221	0.467	0.619	0.689	0.413	0.263	0.716	0.292	0.890	0.831
11	12	13	14	15	16	17	18	19	20
0.026	0.110	0.179	0.761	0.474	0.926	0.289	0.333	0.037	0.550

(For example, $h(13) = 0.179$.)

You decide to mess with Fred a bit, and update your sequence generation routine to ensure that Fred's algorithm fails. And you want it to fail *as badly as possible*.

Give a stream of integers of length 10 in which Fred's implementation of FM will return an answer that is at least 5 off:

Solution: One possible sequence: $3, 4, 7, 9, 10, 16, 20, 4, 7, 9$

What will Fred's implementation of FM return for your sequence? (Your answer here can be approximate, e.g., round to the nearest integer.)

Solution: Fred's algorithm returns, approximately, 1. The correct answer was 7.

[1] You were lucky to get the URL `www.CodedStreamIsSequenceForUnderstandingNumbers.com`!

Problem 4. Right in the Middle

Your job is to monitor a stream of numbers: $x_1, x_2, x_3, \ldots$. (Perhaps these numbers are stock prices for Acme Corporation.) You do not know, in advance, how many numbers will be in the stream. Your task is to design an algorithm that uses as little space as possible and returns an approximate median for the values in the stream. To be more specific, given some error parameter ϵ, your algorithm should return a value v that, with probability at least $1 - \epsilon$, is:

- bigger than at least 1/3 of the numbers in the stream, and
- smaller than at least 1/3 of the numbers in the stream.

(That is, it is between the 33rd and 67th percentile of the numbers.) For example, if the stream consists of the numbers: $1, 1, 2, 3, 5, 5, 6, 7, 9$ (in any order), then your algorithm may return either the number 3 or the number 5.

Problem 4.a. Explain (succinctly) how your algorithm works.

Solution: Using reservoir sampling, select $s = 72 \ln(2/\epsilon)$ values from the stream uniformly at random. Return the median value from your sample.

Problem 4.b. How much space does your algorithm use? You may assume that each number in the stream can be stored in 1 unit of space.

Solution: It uses $s = 72 \ln(2/\epsilon)$ space.

Problem 4.c. Prove that your algorithm is correct.

Solution: First, we will argue that the answer returned is bigger than at least 1/3 of the values in the stream.

Assume that the samples are numbered from 1 to s. Let $X_i = 1$ if sample i is bigger than at least 1/3 of the values in the stream and $X_i = 0$ otherwise. Define $X = \sum(X_i)$.

Notice that $\Pr[X_i = 1] = 2/3$ and hence $\mathrm{E}[X] = 2s/3$. Hence by a Chernoff Bound:

$$\Pr[X < (1 - 1/4)(2s/3)] \leq e^{-(2s/3)(1/4)^2/3} \leq \epsilon/2 \ .$$

That is, with probability at least $1 - \epsilon/2$, we know that at least 1/2 the elements in the sample—including the median in the sample—will be larger than at least 1/3 of the values in the stream.

Notice that the same holds, symmetrically, for values that are smaller than at least 1/3 of the values in the stream. I.e., with probability at least $1 - \epsilon/2$, at least half the elements in the sample—including the meidan in the sample—will be smaller than at least 1/3 of the values in the stream. Thus we conclude that the median of the value is correct with probability at least $1 - \epsilon$.

Problem 5. Building Strong Bonds

We are building a new social network that provides more precisely quantified relationships than Facebook. Instead of simply deciding whether someone is your friend or not, you can carefully adjust your relationship with each person. When you "up" a person, you strengthen your relationship, and when you "down" a person, you weaken your relationship. In this way, for each user, we can classify their relationships;

- If Alice gives Bob more "ups" than "downs," then Bob is a friend of Alice.
- If Alice gives Bob more "downs" than "ups," then Bob is an enemy of Alice.
- If Alice gives Bob the same number of "ups" and "downs," then Alice and Bob have no relationship.

Notice that relationships are not symmetric. Alice may think Bob a friend, while Bob may think Alice an enemy!

The network produces a constant stream of "up" and "down" actions, e.g., (up, u, v) to indicate that user u has given an "up" to user v, or $(down, v, w)$ to indicate that user u has given a "down" to user v.

Continued on the next page.

Problem 5.a. Give an algorithm that will monitor the stream and decide whether or not there are any relationships in the graph. That is, if for every pair (u, v) the number of "up"s equals the number of "down"s, then your algorithm should return EMPTY. Otherwise, it should return FULL. Describe your algorithm and explain why it works.

Solution: This can be solved directly with an L0-sampler. In fact, this is exactly what an L0-sampler does. We need an L0-sampler for a vector with one position for every pair (u, v) (where (u, v) and (v, u) are different positions). Thus it will take polylogarithmic space in n (where the exact space depends on the error that you have chosen). That is, for a fixed probability of error ϵ, it will take $O(\log^4(n/\epsilon))$ space.

Notice that there still is a probability of error. Even for testing whether the graph is empty, the L0-sampler has a possible error.

You might instead use a 1-sparse sampler, which has a fingerprint mechanism to check whether a stream is not 1-sparse. In that case, the space is a bit smaller. There is still some error that the fingerprint mechanism fails, but if the graph is empty, then it will always return that the vector is empty.

What is the probability of error? 

How much space does your algorithm take? 

Continued on the next page.

Problem 5.b. Now describe an algorithm that will monitor the stream and decide whether there are more than 10 relationships in the graph. Show that your algorithm is correct.

What is the probability of error?

How much space does your algorithm take?

Solution: Again, you might use a collection of $O(1)$ L0-samplers. Since each returns a random index, if you have enough of them (e.g., 100) you will see either all 10 relationships, or you will see more than 10 relationships. (You should work out the probability a bit more precisely.)

To be more precise, let's assume we use 100 L0-samplers. Assume that there are ≤ 10 relationships in the graph. Since each L0-sampler can be used to find a random relationship, we can calculate the probability that a specific relationship is *not* discovered: $(1 - 1/10)^100 \leq 1/2^10 \leq 1/1000$. So with probability at least 99/100, we will discover all 10 or fewer relationships.

Similarly, if there are more than 10 relationships in the graph, we can calculate the probability that we will find a specific 11 of the relationships. The same calculation again yields that with probability at least 99/100, we will find 11 relationships.

So at the end of the stream, we query all the L0-samplers. If we see 10 or fewer relationships we return FEW and if we see more than 10 relationships we return MANY.
For a fixed probability of error ϵ, it will take $O(\log^4(n/\epsilon))$ space.

Alternatively, you might try to use an s-sparse sampler, where $s = 11$ (or $s = 20$, or something). However, you do have to be careful here because the s-sparse sampler does not guarantee what it returns if the stream is *not* s-sparse. It may return no relationships, it may return less than 10, or it may return more than 10. Hence if you want to just use an s-sparse sampler, you will need to modify it in some way to detect when the stream is not s-sparse.

Problem 6. Network Robustness

Dr. Bessy Myst is an engineer tasked with keeping the company network up and running. And yet she is always worried about failures—links in the network just seem to keep failing. She has measured that each link in the network fails (over a given period of time) with probability about p, and she wants to know how likely the network is to remain connected throughout. Your job is to help Dr. Myst to estimate this probability.

More formally, Dr. Myst has a graph $G = (V, E)$ which represents the topology of the network which contains n nodes and m edges. Your algorithm should take as input an error parameter ϵ and a probability p. Assuming that each edge fails (i.e., is deleted) independently with probability p, define $R(p)$ to be the *reliability* of the graph, i.e., the probability that the graph remains connected.

Your algorithm should return the probability $R(p) \pm \epsilon$, i.e., a value v such that:

$$R(p) - \epsilon \leq v \leq R(p) + \epsilon \ .$$

It should do this with probability at least $2/3$.

Problem 6.a. Describe an algorithm that outputs $R(p) \pm \epsilon$ in polynomial time. (Your algorithm does not have to be sublinear.)

Solution: We will use a simple sampling algorithm that repeatedly runs experiments deleting edges (with probability p). That is, we will repeat the following procedure s times for $j = 1tos$:

- Make a copy $G_j = G$ of the graph.
- Delete each edge in G_j with probability p.
- Check if G_j is connected. If it is connected, set $X_j = 1$. Otherwise, set $X_j = 0$.

Let $X = \sum_{i=1}^{s} X_j$. Return X/s. We will choose $s = 1/\epsilon$.

Problem 6.b. Prove that your algorithm is correct.

Solution: We can show using a Hoeffding Bound that X/s is a good estimation of $R(p)$. Specifically:

$$\begin{aligned}
\Pr\left[|X/s - R(p)| > \epsilon\right] &= \Pr\left[|X - sR(p)| > \epsilon \cdot s\right] \\
&= \Pr\left[|X - \mathrm{E}\left[X\right]| > \epsilon \cdot s\right] \\
&\leq 2e^{-2\epsilon^2 s^2/s} \\
&\leq 2e^{-2\epsilon s}
\end{aligned}$$

By choosing $s = 1/\epsilon$, we observe that the probability of error is $< 1/3$.

Problem 6.c. Dr. Myst decides to use her algorithm to determine the reliability of a streaming graph, i.e., a graph which is presented as a stream of edge additions and deletions. Explain how to modify your existing algorithm so that it can be used in the streaming context using $o(n^2)$ space (i.e., less than n^2 space, asymptotically). Assume p and ϵ are fixed in advance, and you know the number of nodes n in the graph. Describe your algorithm and explain why it works.

Give the space needed by your algorithm as a function of n and ϵ: [ ]

Solution: We will use a collection of $1/\epsilon$ connectivity sketches. For each connectivity sketch, we will process each of the edges in the graph with probability p. Each sketch represents one of the sampled X_i in the algorithm above.

In more detail: Let h_j be a hash function that maps from edges to $\{0, 1\}$. Assume that $\Pr[h_j(e) = 1] = p$. As we monitor the stream, we will check for each edge whether $h_j(e) = 1$. If so, we will ignore it. Otherwise, we will process it with sketch j.

(Beware, it causes trouble for the sketches to delete edges that were never added, so you do not want to wait until the end to delete edges, unless you first check that the edge exists.)

When the stream is complete, use each sketch to decide whether each graph G_j is connected. Use this to define the random variables $X_1, \ldots, X_s$ where $s = 1/\epsilon$ that you can use as in the previous part to estimate $R(p)$.

Using the graph sketch where the error $= 1/(\epsilon n^{2+c})$ for the L0-samplers in the graph sketch (for an arbitrary constant c), we get a proability of error for each graph sketch of $1/(\epsilon n^c)$, and so the probability that any of the connectivity sketches fail is $1/n^c$. The space usage, then, is $O((n/\epsilon)\log^5(n))$.

Problem 6.d. The CEO of Dr. Myst's company is excited to see her results. But she is unhappy that the approximation is additive. She asks Dr. Myst instead to devise an algorithm to calculate $R(p)(1 \pm \epsilon)$, i.e., a value v such that:

$$R(p)(1-\epsilon) \leq v \leq R(p)(1)+\epsilon) \ .$$

Dr. Myst explains that this is much more difficult. Explain why. (Perhaps give an example where $R(p)$ is very hard to approximate in this way. Perhaps explain where the analysis of your algorithm would fail. There are several different ways you might explain the problem.)

Note that there do exist polynomial-time approximation algorithms for $(1 - R(p))$, so it is not impossible. But there are reasons why it is harder.

Solution: Imagine you have a graph where $R(p)$ is very close to 1, e.g., something like $1 - 1/n^2$. In that case, Dr. Myst will have to repeat her simulation algorithm approximate n^2 times before seeing even one instance in which the graph is disconnected. Thus it will be very hard to find the right answer!

More technically, if you try to use the algorithm given here, the Chernoff/Hoeffding bound will not work, as you will end up with the expected value of X in the exponent, and this expected value will be (potentially) too small.

Scratch Paper

Scratch Paper

Scratch Paper

Scratch Paper