Homework 4

CSCI 4030 / CSCI 6220: Randomized Algorithms — Fall 2026

Rensselaer Polytechnic Institute — 100 points

> Lectures 7 and 8 developed several ideas and techniques that we will reuse here in new settings. Problem 1 is a warm-up in reasoning about a randomized sketch whose selected samples are dependent. Problem 2 asks you to identify a small structural certificate for failure and then use it to analyze a random storage construction. Problem 3 starts from the fully independent CountSketch model and asks, after the analysis is complete, which weaker independence assumptions are sufficient for the proof.

Justify every claim. Acknowledge any collaborators and any external sources you consult, including AI tools, and describe clearly what role they played in your submission.

Throughout the homework, log denotes the natural logarithm. Ignore integer rounding when it does not affect the substance of an argument.

[25 points] Problem 1 (Bottom-k MinHash for near-duplicate detection).

A web crawler may need to decide whether two pages are near duplicates without repeatedly comparing their full text. We will represent each page by the set of its consecutive two-word phrases, called *shingles*. For example,

$$\text{“the cat sat on the mat”}$$

has shingle set

$$\{\text{the cat}, \text{cat sat}, \text{sat on}, \text{on the}, \text{the mat}\}.$$

For two shingle sets A and B, their Jaccard similarity is

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Our goal is to estimate $J(A, B)$ while storing only a small summary of each set.

Algorithm 1: one-feature MinHash. Before processing any documents, assign every possible shingle u a random priority $R(u)$, independently and uniformly from $(0, 1)$. Once assigned, the same priority $R(u)$ is used whenever that shingle appears in any document.

Given a shingle set S:

1. Look at the priority $R(u)$ of every shingle $u \in S$.
2. Return the shingle with the smallest priority:

$$H(S) = \arg\min_{u \in S} R(u).$$

A *tie* would mean that two different shingles receive exactly the same priority. Because the priorities are drawn independently from a continuous distribution, a tie occurs with probability 0. Thus the minimum-priority shingle is unique with probability 1.[1]

How does this relate to the Jaccard similarity? Look at all shingles that occur in either document, $A \cup B$. Exactly one of them has the smallest priority. If that shingle occurs in *both* documents, then Algorithm 1 returns the same shingle for A and B. If it occurs in only one document, the two outputs cannot agree. Thus the event "the two MinHash outputs agree" is directly connected to how much of $A \cup B$ lies in $A \cap B$.

Here is a concrete run. Let

$$D_A = \text{“the cat sat on the mat”},$$
$$D_B = \text{“the cat slept on the mat”},$$

so that

$$A = \{\text{the cat}, \text{cat sat}, \text{sat on}, \text{on the}, \text{the mat}\},$$
$$B = \{\text{the cat}, \text{cat slept}, \text{slept on}, \text{on the}, \text{the mat}\}.$$

Their exact Jaccard similarity is $J(A, B) = 3/7$. Suppose the random priorities happen to be

shingle	the cat	cat sat	sat on	on the	the mat	cat slept	slept on
$R(\cdot)$	.12	.71	.44	.08	.31	.56	.22

Algorithm 1 returns "on the" for both documents, because .08 is the smallest priority appearing in either set and that shingle occurs in both.

(a) (5 points) Prove that Algorithm 1 satisfies

$$\mathbb{P}[H(A) = H(B)] = \frac{|A \cap B|}{|A \cup B|} = J(A, B).$$

Your argument should identify which shingle of $A \cup B$ determines whether the two outputs agree.

One run is not enough for an accurate numerical estimate. Algorithm 1 gives only the yes/no random variable

$$Y = \begin{cases} 1, & H(A) = H(B), \\ 0, & H(A) \neq H(B). \end{cases}$$

Part (a) says that $\mathbb{E}Y = J(A, B)$, so Y is an unbiased estimator of the similarity. But a single run can return only 0 or 1. For example, if the true similarity is 0.6, one run still reports either 0 or 1, never a number close to 0.6.

One straightforward fix would be to perform Algorithm 1 independently k times. Formally, generate k independent priority assignments $R_1, \ldots, R_k$. Under assignment R_t, compute

$$H_t(S) = \arg\min_{u \in S} R_t(u).$$

[1] In practice, MinHash uses a seeded hash function with a large output range and treats the hash value as the priority. This lets priorities be generated as needed from a short seed rather than stored for every possible shingle. We analyze the ideal independent-priority model to avoid implementation details such as hash collisions.

Then estimate $J(A, B)$ by the fraction of trials t for which $H_t(A) = H_t(B)$. This gives k independent yes/no observations, but it also requires processing every shingle under k separate priority assignments.

Bottom-k MinHash gets k pieces of information from just *one* priority assignment. For the remainder of the problem, assume

$$k \leq |A \cup B|.$$

If a set contains fewer than k shingles, its sketch simply stores all of them.

Algorithm 2: Bottom-k MinHash. Fix one random priority assignment R and an integer k.
Sketch a set S:

1. Examine $R(u)$ for every $u \in S$.
2. Keep the $\min\{k, |S|\}$ shingles of S with the smallest priorities, together with their priorities. Call this stored set $B_k(S)$.

Compare sketches $B_k(A)$ and $B_k(B)$:

1. Merge the two stored sets and remove duplicate shingles.
2. Among the merged shingles, keep the k with smallest priorities. Call this set T.
3. Count how many shingles in T occur in both input sketches. Call this number X.
4. Output
$$\widehat{J} = \frac{X}{k}.$$

Intuition. Imagine sorting every shingle in $A \cup B$ from smallest to largest random priority. Algorithm 2 tries to recover just the first k shingles in this random ordering. The fraction of those k shingles that belong to both documents should reflect the fraction of the whole union that belongs to both documents. The analysis below makes this precise.

For the same documents and priorities above, take $k = 3$. The sketches are

$$B_3(A) = \{\text{on the}, \text{the cat}, \text{the mat}\}, \qquad B_3(B) = \{\text{on the}, \text{the cat}, \text{slept on}\}.$$

Merging them gives four distinct shingles with priorities $.08, .12, .22, .31$. The three smallest are

$$T = \{\text{on the}, \text{the cat}, \text{slept on}\}.$$

Among these, "on the" and "the cat" occur in both input sketches, so Algorithm 2 outputs

$$\widehat{J} = \frac{2}{3}.$$

The true similarity is $3/7$, so this particular randomized run is not exact. The questions below ask whether the estimator is correct on average and how much it varies from run to run.

(b) (4 points) First justify the comparison step in Algorithm 2. Prove that merging $B_k(A)$ and $B_k(B)$ and retaining the k smallest priorities gives exactly the k smallest-priority shingles in $A \cup B$.

In particular, explain why a shingle discarded from both input sketches could not possibly belong among the k smallest-priority shingles of the union. Then prove that if one of these k union shingles lies in $A \cap B$, it must appear in both input sketches. Conclude that X is exactly the number of shared shingles among the k smallest-priority shingles of $A \cup B$.

(c) (3 points) Argue that the k smallest-priority shingles form a uniformly random k-element subset of $A \cup B$. Use this fact to prove

$$\mathbb{E}[\widehat{J}] = J(A, B).$$

(d) (9 points) Now quantify the dependence among those k selected shingles. Write

$$W = A \cup B, \qquad C = A \cap B, \qquad M = |W|.$$

For each shared shingle $u \in C$, let

$$I_u = \mathbf{1}\{u \text{ is among the } k \text{ smallest-priority shingles of } W\}.$$

Show that, for distinct $u, v \in C$,

$$\mathbb{P}[I_u = 1] = \frac{k}{M}, \qquad \mathbb{P}[I_u = I_v = 1] = \frac{k(k-1)}{M(M-1)}.$$

Use these identities to compute $\mathrm{Cov}(I_u, I_v)$ and show that it is negative. Then derive

$$\mathrm{Var}(\widehat{J}) = \frac{J(A, B)(1 - J(A, B))}{k} \, \frac{M - k}{M - 1}.$$

Derive this variance formula directly from the indicators above.

(e) (4 points) Return to the alternative described above: run Algorithm 1 using k independent priority assignments $R_1, \ldots, R_k$, and estimate the similarity by the fraction of runs in which the two outputs agree.

Show that this estimator has variance

$$\frac{J(A, B)(1 - J(A, B))}{k}.$$

Compare it with the bottom-k variance from part (d). Why does using one priority assignment not hurt here, even though the k selected shingles are not independent? Interpret the factor $(M-k)/(M-1)$: when does the variance improvement become substantial, and what happens when $k = M$?

[40 points] Problem 2 (Random XOR redundancy for repairing failed storage shards).

A large file is partitioned into n non-overlapping, equal-length blocks

$$x_1, \ldots, x_n \in \{0, 1\}^b,$$

called *data shards*. Concatenating $x_1, \ldots, x_n$ reconstructs the original file. We would like to tolerate the loss of a small number of data shards without storing a complete second copy of the file.

XOR can repair one missing shard. The shards are bit vectors, and $\oplus$ denotes bitwise XOR. For example,

$$x_1 = 1011, \qquad x_2 = 0110, \qquad x_3 = 1100.$$

If we additionally store

$$y = x_1 \oplus x_2 \oplus x_3 = 0001,$$

then losing x_2 is harmless as long as x_1, x_3, and y remain available:

$$x_2 = y \oplus x_1 \oplus x_3 = 0110.$$

We call y a *parity shard*. More generally, a parity shard is the XOR of some selected data shards.

Generating several parity shards at random. Fix a failure budget $t \geq 2$: later we will ask whether the stored parity information can recover as many as t missing data shards. Choose a parameter $q \in (0, 1)$. To create parity shard y_j, independently include each data shard x_i with probability q, and XOR together the included shards. If $P_{ji} = 1$ means that x_i was included in y_j, then

$$P_{ji} \sim \text{Bernoulli}(q), \qquad y_j = \bigoplus_{i:P_{ji}=1} x_i.$$

Doing this independently for $j = 1, \ldots, r$ produces a random binary matrix $P \in \{0, 1\}^{r \times n}$. Its columns correspond to data shards and its rows correspond to parity shards. For example, one possible draw is

	x_1	x_2	x_3	x_4	x_5	x_6
y_1	1	0	1	0	0	1
y_2	0	1	1	0	1	0
y_3	1	0	0	1	1	0
y_4	0	1	0	1	0	1

so, for instance,

$$y_1 = x_1 \oplus x_3 \oplus x_6, \qquad y_2 = x_2 \oplus x_3 \oplus x_5.$$

The n data shards together with the r parity shards form an *XOR erasure code*; the random matrix P specifies which code we generated.

Throughout this problem, assume the parity shards themselves remain available and only data shards fail. This lets us isolate the recovery question and is realistic when r is much smaller than n: the smaller collection of parity shards can be separately replicated or placed on more reliable storage. There is no universal best value of r; more parity shards cost more storage but can protect against more failure patterns. Our goal is to understand what a random construction can guarantee for a given r.

One repair can enable another. Suppose four data shards are protected by two available parity shards

$$y_1 = x_1 \oplus x_2, \qquad y_2 = x_2 \oplus x_3 \oplus x_4,$$

and suppose x_1 and x_2 are missing while x_3, x_4, y_1, and y_2 are available. The relation defining y_1 initially has two unknowns, so it cannot recover either one. The relation defining y_2 has exactly one unknown, x_2, so it recovers x_2. Once x_2 is known, the first relation has only one unknown left and recovers x_1.

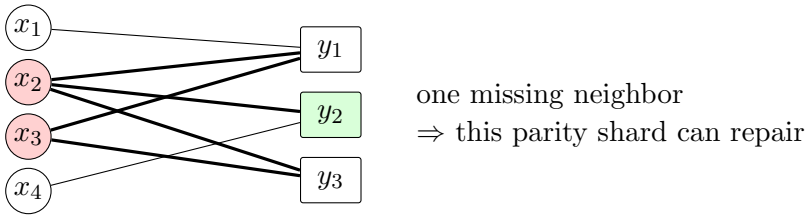
This motivates the following recovery procedure.

Algorithm: Iterative XOR repair. Let E be the set of data shards that are currently missing.

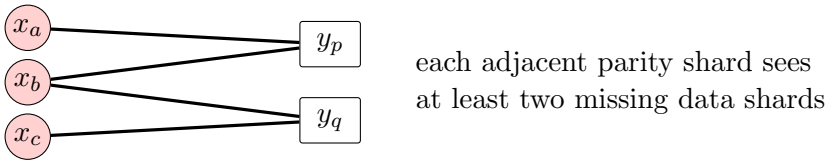
1. If $E = \varnothing$, stop: every failed shard has been repaired.
2. Find an available parity shard whose defining XOR contains exactly one shard from E. If no such parity shard exists, stop: the repair procedure is stuck.
3. Recover that missing shard by XORing the parity value with all known data shards appearing in that parity relation.
4. Remove the recovered shard from E and return to Step 1.

We will analyze when this algorithm gets stuck, and how likely that is for the randomly generated matrix P.

P as a bipartite graph. Place a vertex for each data shard on the left and a vertex for each parity shard on the right. Join x_i to y_j exactly when $P_{ji} = 1$. Thus the graph and the matrix P contain exactly the same information. If the highlighted data shards are currently missing, a parity vertex with exactly one highlighted neighbor can perform the next repair.



When does iterative repair get stuck? A nonempty set S of data vertices is called a *stopping set* if every parity vertex has either zero or at least two neighbors in S. If exactly the shards in S remain missing, then no parity shard isolates a single unknown, so the algorithm has no legal next step.



(a) (7 points) Prove that iterative XOR repair succeeds on an initially failed set E if and only if E contains no nonempty stopping set.

(b) (10 points) We now choose the inclusion probability q. Fix a set S of s missing data shards. For one randomly generated parity shard, show that the probability that its defining XOR contains *exactly one* shard from S is

$$\alpha_s(q) = sq(1-q)^{s-1}.$$

Suppose all t shards in our failure budget are currently missing. Find the value of q that maximizes $\alpha_t(q)$, and show that the maximizer is

$$q = \frac{1}{t}.$$

Use this value of q for the rest of the problem. Also compute the expected number of data shards included in one parity shard when $q = 1/t$. Finally, show that for every $1 \le s \le t$,

$$\alpha_s(1/t) \ge \frac{s}{et}.$$

(c) (10 points) Fix a particular failed set $E \subseteq [n]$ with $|E| = t$, chosen *before* the random parity matrix P is generated. Use part (a), part (b), and a union bound to prove that

$$r \ge 3et \log t$$

is sufficient to make the probability that iterative repair fails on this fixed E at most $2t^{-2}$. You may use $\binom{t}{s} \le t^s$.

Interpret the result: how much parity storage is sufficient to protect one fixed pattern of t failed data shards when $t \ll n$?

(d) (13 points) Now ask for one randomly generated parity matrix that repairs *every* failure pattern $E \subseteq [n]$ with $|E| \le t$. Prove that

$$r \ge 2et \log n$$

is sufficient to make this happen with probability at least $1 - 2/n$. Do not union-bound over failure sets E; instead, use part (a) to identify the smaller collection of bad objects that must be ruled out and union-bound over those. You may use $\binom{n}{s} \le n^s$.

Explain where the $\log n$ factor enters this argument, and why the fixed-E argument in part (c) did not need to pay for it.

[35 points] Problem 3 (CountSketch as a randomized linear map).

Suppose $x, y \in \mathbb{R}^d$ are very high-dimensional vectors and we want to estimate $\langle x, y \rangle$ after compressing each vector to only $m \ll d$ numbers. CountSketch is a random linear map

$$S : \mathbb{R}^d \to \mathbb{R}^m$$

that first flips coordinate signs and then adds coordinates into randomly chosen buckets.

The computation. Choose a sign $s(i) \in \{-1, +1\}$ and a bucket $h(i) \in [m]$ for every coordinate $i \in [d]$. Define

$$D = \mathrm{diag}(s(1), \ldots, s(d)) \in \mathbb{R}^{d \times d}$$

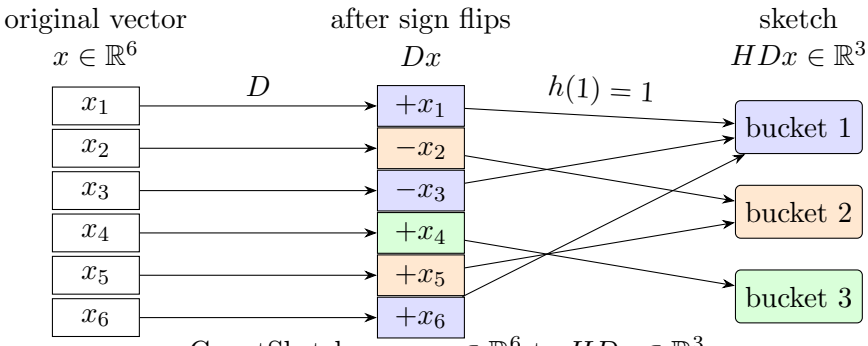
and the hashing matrix $H \in \{0, 1\}^{m \times d}$ by

$$H_{bi} = \mathbf{1}\{h(i) = b\}.$$

Every column of H therefore contains exactly one 1. The sketching matrix is

$$S = HD, \qquad Sx = HDx.$$

So D changes signs, while H sums the signed coordinates assigned to each bucket.



CountSketch maps $x \in \mathbb{R}^6$ to $HDx \in \mathbb{R}^3$;
coordinates with the same shading land in the same bucket.

For this example,

$$D = \mathrm{diag}(+1, -1, -1, +1, +1, +1), \qquad H = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix},$$

so

$$Sx = HDx = \begin{pmatrix} x_1 - x_3 + x_6 \\ -x_2 + x_5 \\ x_4 \end{pmatrix}.$$

Given only Sx and Sy, we estimate the original inner product by

$$Z = \langle Sx, Sy \rangle = x^\top S^\top S y.$$

Define

$$G := S^\top S = DH^\top HD.$$

If G were exactly I, then $Z = \langle x, y \rangle$. Verify from the definitions,

$$G_{ii} = 1, \qquad G_{ij} = s(i)s(j)\mathbf{1}\{h(i) = h(j)\} \quad (i \neq j).$$

Thus the diagonal entries are already correct; collisions create the random off-diagonal entries.

Analysis of the error $Z - \langle x, y \rangle$. For now, assume full independence: the bucket choices $h(1), \ldots, h(d)$ are independent and uniform on $[m]$, the signs $s(1), \ldots, s(d)$ are independent and uniform on $\{-1, +1\}$, and the bucket choices are independent of the signs. After proving the guarantee, you will determine how much independence the proof actually needs.

(a) (7 points) Prove

$$\mathbb{E}[G] = I.$$

Do this entry-by-entry. Deduce that for every fixed $x, y \in \mathbb{R}^d$,

$$\mathbb{E}\langle Sx, Sy \rangle = x^\top \mathbb{E}[G] y = \langle x, y \rangle.$$

(b) (10 points) Let $A := G - I$. Then

$$\langle Sx, Sy\rangle - \langle x, y\rangle = x^\top Ay.$$

Using $A_{ii} = 0$ and $A_{ij} = A_{ji}$, show that

$$x^\top Ay = \sum_{1\le i<j\le d} (x_i y_j + x_j y_i) A_{ij}.$$

To prepare for the variance calculation, prove these two identities:

(i) For three distinct indices i, j, k, prove

$$\mathbb{E}[A_{ij}A_{ik}] = 0.$$

(ii) For four distinct indices i, j, k, ℓ, prove

$$\mathbb{E}[A_{ij}A_{k\ell}] = 0.$$

Explain briefly why, after relabeling indices, these two cases cover every product of two distinct off-diagonal entries that appears when the square of the sum above is expanded.

(c) (10 points) Use part (b) to show

$$\mathrm{Var}(\langle Sx, Sy\rangle) \le \frac{1}{m}\sum_{i<j}(x_i y_j + x_j y_i)^2 \le \frac{2}{m}\|x\|_2^2\|y\|_2^2.$$

Then use Chebyshev's inequality to prove

$$\mathbb{P}[|\langle Sx, Sy\rangle - \langle x, y\rangle| > \varepsilon\|x\|_2\|y\|_2] \le \frac{2}{m\varepsilon^2}.$$

Give a sufficient value of m that makes this probability at most δ.

(d) (8 points) The proof above assumed full independence. Now determine weaker assumptions that are already sufficient for the same proof. In each case, give a sufficient value of ℓ for ℓ-wise independence and justify your answer from the calculation above.

(i) How much independence among the signs is sufficient for the unbiasedness calculation $\mathbb{E}[G] = I$?

(ii) How much independence among the signs is sufficient for the shared-index calculation $\mathbb{E}[A_{ij}A_{ik}] = 0$ with distinct i, j, k?

(iii) How much independence among the signs is sufficient for the four-distinct-indices calculation $\mathbb{E}[A_{ij}A_{k\ell}] = 0$?

(iv) What property of the hash family is sufficient to guarantee $\mathbb{E}[A_{ij}^2] \le 1/m$? Does the proof require collision events for different pairs to be independent?

(v) Where is independence between the sign family and the hash family used?

Conclude by stating one concise set of sufficient randomness assumptions under which the CountSketch guarantee proved above still holds.