DSA5101 Analysis Questions

Zhao Peiduo

AY2025/26 Sem1

Disclaimer: Answers could be based on ChatGPT results which may be inaccurate / incorrect.

Lecture 1: Frequent Itemsets and Association Rules

Question 1

You are given two approaches for counting item pairs in memory:

- **Approach 1 (Triangular Matrix):** Uses 4 bytes per pair.
- **Approach 2 (Table of Triples):** Uses 12 bytes per occurring pair (only for pairs count > 0).

Suppose there are n distinct items, leading to $\dfrac{n(n-1)}{2}$ possible pairs in total.

1. Determine the threshold value of f at which both approaches require the same amount of memory.
2. Interpret your result: for which values of f should we prefer Approach 1 vs. Approach 2?
3. For $n = 10^5$, estimate the total memory (in GB) used by both approaches when $f = 0.1$ and $f = 0.01$. Assume $1\,\text{GB} = 10^9$ bytes.

Relevant Concepts

Total Possible Pairs:

$$\text{Total Pairs} = \frac{n(n-1)}{2}$$

Triangular Matrix Storage:

$$M_1 = 4 \times \frac{n(n-1)}{2}$$

Triple Table Storage (only for pairs with count > 0):

$$M_2 = 12 \times f \times \frac{n(n-1)}{2}$$

Answer

(1) Threshold for Switching

$$M_1 = M_2 \quad \Rightarrow \quad 2n(n-1) = 6f\,n(n-1) \quad \Rightarrow f = \frac{1}{3}$$

(2) Interpretation

- If $f < \frac{1}{3}$, few pairs occur $\Rightarrow$ **Approach 2 (Triples)** uses less memory.
- If $f > \frac{1}{3}$, many pairs occur $\Rightarrow$ **Approach 1 (Matrix)** is more memory-efficient.

(3) Example with $n = 10^5$

$$\text{Total Pairs} = \frac{(10^5)(10^5 - 1)}{2} \approx 5 \times 10^9$$

$$M_1 = 4 \times 5 \times 10^9 = 2 \times 10^{10}\,\text{bytes} = 20\,\text{GB}$$

$$M_2 = 12f \times 5 \times 10^9 = 60f \times 10^9\,\text{bytes} = 60f\,\text{GB}$$

f	M_2(GB)	Better Approach
0.1	6	Triples (Approach 2)
0.01	0.6	Triples (Approach 2)

Question 2

We aim to analyze why many hash buckets in the PCY algorithm may end up being infrequent. Given:

- Main memory: $1\,\text{GB} = 10^9$ bytes, number of available buckets $\frac{10^9}{4} = 2.5 \times 10^8$.
- Each hash bucket is represented by a 4-byte integer.
- Data size: 10^9 baskets, each containing 10 items.

Answer the following questions:

1. Compute the total number of pairs in all baskets.
2. Compute the average count of each bucket if all pairs are evenly distributed among buckets.
3. Given a support threshold $s = 1000$, determine how many buckets can be frequent at most.
4. Interpret your findings: why are most buckets expected to be infrequent?

Relevant Concepts

Total Number of Pairs:

$$\text{Total Pairs} = (\text{Number of Baskets}) \times \binom{\text{Items per Basket}}{2}$$

Average Count per Bucket:

$$\text{Average Count} = \frac{\text{Total Pairs}}{\text{Number of Buckets}}$$

Maximum Number of Frequent Buckets:

$$\text{Max Frequent Buckets} = \frac{\text{Total Pairs}}{s}$$

Answer

(1) Total Number of Pairs

$$10^9 \times \binom{10}{2} = 10^9 \times 45 = 4.5 \times 10^{10}$$

(2) Average Count per Bucket

$$\frac{4.5 \times 10^{10}}{2.5 \times 10^8} = 180$$

(3) Maximum Number of Frequent Buckets

$$\frac{4.5 \times 10^{10}}{1000} = 4.5 \times 10^7$$

(4) Interpretation
Out of 2.5×10^8 buckets, at most 4.5×10^7 can be frequent. Since $4.5 \times 10^7 \ll 2.5 \times 10^8$, the majority of buckets are expected to be **infrequent**. This illustrates why the PCY algorithm often has a sparse bitmap, making it memory-efficient.

Question 3

Suppose we run the PCY algorithm, and the average bucket has count $\frac{s}{10}$.

1. If we use **Multihash** with two half-sized hash tables, determine the new average count per bucket.
2. Find the maximum fraction of buckets in either hash table that can be frequent.
3. Compute the probability that a random infrequent pair will be hashed to a frequent bucket in both hash tables.
4. Compare this probability with the case of using a single hash table in PCY, and explain the benefit of using Multihash.

Relevant Concepts

Average Count per Bucket:

$$\frac{\#\text{Pairs}}{\#\text{Buckets}}$$

Answer

(1) Average Count per Bucket (Multihash)
When using two half-sized hash tables, each receives half the total pairs. Therefore, the average count per bucket becomes:

$$\text{Average Count} = \frac{s}{5}$$

(2) Fraction of Frequent Buckets
A bucket is considered frequent if its count $\geq s$. Let:

$$n_b = \text{total number of buckets}, \quad c_i = \text{count in bucket } i, \quad \bar{c} = \frac{s}{5}$$

Then the total sum of counts across all buckets is:

$$\text{Total count} = n_b \times \bar{c} = n_b \times \frac{s}{5}$$

Suppose f fraction of buckets are frequent. Then the minimum total count that would make f fraction frequent is:

$$\text{Total count} \geq f \cdot n_b \cdot s$$

Since the actual total count is $n_b \times \frac{s}{5}$, we can compare:

$$n_b \times \frac{s}{5} \geq f \cdot n_b \cdot s$$

$$\Rightarrow f \leq \frac{1}{5}$$

Hence, at most $\frac{1}{5}$ of the buckets can be frequent.

(3) Probability of Infrequent Pair Mapping to Frequent Buckets
For a random infrequent pair, the probability that it hashes to a frequent bucket in **both** tables is:

$$\left(\frac{1}{5}\right)^2 = 0.04$$

(4) Comparison with PCY (Single Hash Table)

With one hash table and similar reasoning as part (2), at most $\frac{1}{10}$ of buckets can be frequent:

$$P(\text{infrequent pair} \rightarrow \text{frequent bucket}) = 0.1$$

Interpretation:
The probability 0.04 under Multihash is 2.5 times lower than 0.1 under PCY. Thus, using **Multihash** reduces false positives (infrequent pairs hashed to frequent buckets) and improves filtering efficiency.

Lecture 2: Finding Similar items: Locality Sensitivity Hashing

Question 1

We aim to prove the following fundamental property of Min-Hashing:

$$\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$$

where $\text{sim}(C_1, C_2)$ denotes the Jaccard similarity between two sets (or columns) C_1 and C_2.

1. Prove the claim $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$ using the three-type row argument.
2. Show that the expected similarity of two Min-Hash signatures equals their Jaccard similarity.

Relevant Concepts

Definition of Jaccard Similarity:

$$\text{sim}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

Min-Hash Function: Given a random permutation π of all rows,

$$h_\pi(C) = \text{index of the first row (under } \pi) \text{ where } C \text{ has a 1.}$$

- The fraction of matching hash values between two signatures is an unbiased estimator of their Jaccard similarity.
- Increasing K (the number of hash functions) reduces the variance of the estimate roughly by $O\left(\frac{1}{\sqrt{K}}\right)$.

Answer

(1) Let the total number of rows be n. Each row can belong to one of three types with respect to columns C_1 and C_2:

- **Type X:** Row has 1 in both columns. Let there be x such rows, i.e., $|C_1 \cap C_2| = x$.
- **Type Y:** Row has 1 in exactly one of the two columns. Let there be y such rows, so $|C_1 \cup C_2| = x+y$.
- **Type Z:** Row has 0 in both columns. These do not affect the outcome.

Under a random permutation π, the first row that has a 1 in either column must be either an type X or type Y row.
The event $h_\pi(C_1) = h_\pi(C_2)$ occurs if the first such row under permutation is a **Type X** row.

$$\Pr[h_\pi(C_1) = h_\pi(C_2)] = \frac{\text{number of Type X rows}}{\text{number of Type X + Type Y rows}} = \frac{x}{x+y}$$

$$\Rightarrow \Pr[h_\pi(C_1) = h_\pi(C_2)] = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} = \text{sim}(C_1, C_2) \quad \text{(proven)}$$

(2) Generate K independent Min-Hash functions, each corresponding to a random permutation $\pi_1, \pi_2, \ldots, \pi_K$. For each set (or column), construct a **signature vector**:

$$\text{Sig}(C_i) = [h_{\pi_1}(C_i), h_{\pi_2}(C_i), \ldots, h_{\pi_K}(C_i)]$$

Define the **similarity of two signatures** as:

$$\text{sim}_{\text{sig}}(C_1, C_2) = \frac{1}{K} \sum_{k=1}^{K} I[h_{\pi_k}(C_1) = h_{\pi_k}(C_2)]$$

where $I[\cdot]$ is an indicator function that equals 1 if the condition inside is true, and 0 otherwise.

Taking the expected value:

$$E[\text{sim}_{\text{sig}}(C_1, C_2)] = E\left[\frac{1}{K}\sum_{k=1}^{K} I[h_{\pi_k}(C_1) = h_{\pi_k}(C_2)]\right]$$

Since the hash functions are independent and identically distributed:

$$E[\text{sim}_{\text{sig}}(C_1, C_2)] = \frac{1}{K}\sum_{k=1}^{K} E[I[h_{\pi_k}(C_1) = h_{\pi_k}(C_2)]]$$

Each expectation $E[I[h_{\pi_k}(C_1) = h_{\pi_k}(C_2)]]$ equals the probability that the two sets have the same Min-Hash under permutation π_k:

$$E[I[h_{\pi_k}(C_1) = h_{\pi_k}(C_2)]] = \Pr[h_{\pi_k}(C_1) = h_{\pi_k}(C_2)]$$

Using the Min-Hash property proved earlier:

$$\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$$

Substituting this:

$$E[\text{sim}_{\text{sig}}(C_1, C_2)] = \frac{1}{K}\sum_{k=1}^{K} \text{sim}(C_1, C_2) = \text{sim}(C_1, C_2)$$

Question 2

In Locality-Sensitive Hashing (LSH), the probability that two items with similarity t become a *candidate pair* is given by:

$$P(\text{candidate}) = 1 - (1 - t^r)^b$$

where:

- r = number of rows per band
- b = number of bands
- t = similarity between two signatures

1. Find the threshold t corresponding to the steepest point of the S-curve, where $P(\text{candidate}) = \frac{1}{2}$.
2. Derive both the exact solution and the large-b approximation.

Relevant Concepts

The S-curve describes how the probability that two sets become candidates transitions from low to high as similarity t increases. The **threshold** t is the similarity value around which this transition is most rapid.

Answer

(1) Setting up the Equation
We find the threshold t by setting:

$$P(\text{candidate}) = 1 - (1 - t^r)^b = \frac{1}{2}$$

Rearranging:

$$1 - t^r = \left(\frac{1}{2}\right)^{1/b}$$

$$t^r = 1 - \left(\frac{1}{2}\right)^{1/b}$$

$$t = \left(1 - \left(\frac{1}{2}\right)^{1/b}\right)^{1/r}$$

(2) Approximation for Large b
For large b, note that:

$$\left(\frac{1}{2}\right)^{1/b} = e^{-\frac{\ln 2}{b}} \approx 1 - \frac{\ln 2}{b}$$

Substitute into the exact expression:

$$t^r \approx 1 - \left(1 - \frac{\ln 2}{b}\right) = \frac{\ln 2}{b} \quad \Rightarrow t \approx \left(\frac{\ln 2}{b}\right)^{1/r}$$

Question 3

In the **Random Hyperplane LSH** method, we hash high-dimensional vectors using random hyperplanes.

- Let $\mathbf{v}$ be the *normal vector* to a hyperplane.
- Each vector $\mathbf{v}$ defines a hash function $h_{\mathbf{v}}$ with two possible outputs:

$$h_{\mathbf{v}}(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{v} \cdot \mathbf{x} \geq 0, \\ -1 & \text{if } \mathbf{v} \cdot \mathbf{x} < 0. \end{cases}$$

Let H denote the family of all such hash functions determined by any random vector $\mathbf{v}$.
Prove that for any two points $\mathbf{x}$ and $\mathbf{y}$,

$$\Pr[h(\mathbf{x}) = h(\mathbf{y})] = 1 - d(\mathbf{x}, \mathbf{y}),$$

where $d(\mathbf{x}, \mathbf{y}) = \frac{\theta}{\pi}$ and θ is the angle between $\mathbf{x}$ and $\mathbf{y}$.

Relevant Concepts

Geometric Setup:

- Each random vector $\mathbf{v}$ defines a random hyperplane passing through the origin.
- The hyperplane divides space into two half-spaces (buckets): one where $\mathbf{v} \cdot \mathbf{x} \geq 0$ and one where $\mathbf{v} \cdot \mathbf{x} < 0$.
- The two points $\mathbf{x}$ and $\mathbf{y}$ will have the same hash value if they lie on the same side of the hyperplane.

Answer

Let θ be the angle between $\mathbf{x}$ and $\mathbf{y}$. A random hyperplane, determined by $\mathbf{v}$, will separate $\mathbf{x}$ and $\mathbf{y}$ if and only if it lies between them in the plane spanned by $\mathbf{x}$ and $\mathbf{y}$. The key observation is:

$$h_{\mathbf{v}}(\mathbf{x}) \neq h_{\mathbf{v}}(\mathbf{y}) \quad \text{if the hyperplane is between } \mathbf{x} \text{ and } \mathbf{y}.$$

Since $\mathbf{v}$ is chosen uniformly at random, the direction of the normal vector is equally likely to fall anywhere on the unit circle within that plane.
Thus, the probability that the hyperplane separates $\mathbf{x}$ and $\mathbf{y}$ is proportional to the angle θ between them:

$$\Pr[h_{\mathbf{v}}(\mathbf{x}) \neq h_{\mathbf{v}}(\mathbf{y})] = \frac{\theta}{\pi}.$$

Consequently, the probability that the two points hash to the same side of the hyperplane is:

$$\Pr[h_{\mathbf{v}}(\mathbf{x}) = h_{\mathbf{v}}(\mathbf{y})] = 1 - \frac{\theta}{\pi} = 1 - d(\mathbf{x}, \mathbf{y})$$

Lecture 3: Clustering

Question 1

Clustering evaluation can be performed using the **Purity** metric, which measures how homogeneous each cluster is with respect to the ground truth classes.Given the following clusters and class distributions, compute the purity of each cluster:

Cluster	Class 1	Class 2	Class 3
Cluster I	5	1	0
Cluster II	1	4	1
Cluster III	1	0	3

Relevant Concepts

The **Purity** of a cluster ω_i is the ratio between the number of points from the dominant (most frequent) class in that cluster and the total number of points in the cluster:

$$Purity(\omega_i) = \frac{1}{n_i} \max_j(n_{ij}),$$

where:

- n_{ij} is the number of points in cluster ω_i that belong to class j,
- n_i is the total number of points in cluster ω_i.

Answer

Cluster I:

$$Purity(\omega_1) = \frac{1}{6} \max(5, 1, 0) = \frac{5}{6}.$$

Cluster II:

$$Purity(\omega_2) = \frac{1}{6} \max(1, 4, 1) = \frac{4}{6} = \frac{2}{3}.$$

Cluster III:

$$Purity(\omega_3) = \frac{1}{4} \max(1, 0, 3) = \frac{3}{4}.$$

Lecture 5: Recommender System

Question 1

Given the following data, compute the TF-IDF score for term t_1 in document d_1.

$$f_{11} = 4, \quad \max_k f_{k1} = 8, \quad n_1 = 3, \quad N = 10$$

Interpretation:

- $f_{11} = 4$ means the term t_1 appears *four times* in document d_1.
- $\max_k f_{k1} = 8$ means the most frequent term in the same document d_1 appears *eight times*. This is used to normalize term frequencies inside the document.
- $n_1 = 3$ means the term t_1 appears in *three different documents* across the whole corpus.
- $N = 10$ means the corpus contains *ten documents* total.

Relevant Concepts

The TF-IDF metric helps identify the most "important" words in a document relative to the entire corpus.

- **Term Frequency (TF):** Measures how frequently a term appears in a document, normalized for document length.

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

- **Inverse Document Frequency (IDF):** Reduces the weight of common terms across many documents.

$$IDF_i = \log \frac{N}{n_i}$$

- **TF-IDF Score:**

$$TFIDF_{ij} = TF_{ij} \times IDF_i$$

Answer

For term t_1 in document d_1:

$$TF_{11} = \frac{4}{8} = 0.5, \quad IDF_1 = \log \frac{10}{3} = \log(3.333) \approx 0.523.$$

$$\Rightarrow TFIDF_{11} = 0.5 \times 0.523 = 0.262.$$

Lecture 6: Link Analysis

Question 1

Consider the problem of running the PageRank algorithm on a very large web graph that cannot fully fit into memory. The sparse matrix M is stored using only its nonzero entries as (source, degree, destination nodes). Assume we store r^{old} and M on disk, and r^{new} can only partially fit into memory.

1. Using the sparse matrix encoding below, perform one step of the power-iteration update for all nodes. Assume $\beta = 0.8$, $N = 6$, and the initial ranks $r^{old} = [1, 1, 1, 1, 1, 1]$.

source node	degree	destination nodes
0	4	$0, 1, 3, 5$
1	2	$0, 5$
2	2	$3, 4$

   Compute r^{new} after the first iteration.

2. Suppose r^{new} cannot fit in memory. Explain how the **block-based update algorithm** and the **block-stripe update algorithm** manage memory during PageRank computation. Illustrate how r^{new} and M are partitioned and processed in each method.

3. Derive the communication cost per iteration of the power method.

Relevant Concepts

- **Sparse Matrix Encoding:** Each row of M stores the source node i, its out-degree d_i, and the list of destination nodes $\{j_1, j_2, \ldots, j_{d_i}\}$.

$$M = \{(i, d_i, \text{dest}_1, \ldots, \text{dest}_{d_i})\}$$

  The space requirement is proportional to the number of edges plus the number of nodes:

$$|M| = \#\text{edges} + \#\text{nodes}.$$

- **Power-Iteration Update Step:**

$$r^{new}(dest_j) += \beta \frac{r^{old}(i)}{d_i}$$

Initialize all entries:

$$r^{new} = \frac{1-\beta}{N}.$$

- **Block-Based Update Algorithm:** Break r^{new} into k blocks that fit into memory, and for each block, scan M and r^{old} once.
- **Block-Stripe Update Algorithm:** Further divide M into *stripes* such that each stripe contains only destination nodes belonging to a corresponding block of r^{new}.

Answer

(1) Power Iteration Step
Initialize:

$$r^{new} = \frac{1-\beta}{N} = \frac{0.2}{6} \approx 0.0333$$
$$r^{old}(i) = 1, \ \forall i$$

For each source node:

$$\begin{aligned}
i = 0: &\quad \text{degree} = 4 \Rightarrow \text{dest} = \{0,1,3,5\} \\
&\quad r^{new}(0,1,3,5) += 0.8 \times \tfrac{1}{4} = 0.2 \\
i = 1: &\quad \text{degree} = 2 \Rightarrow \text{dest} = \{0,5\} \\
&\quad r^{new}(0,5) += 0.8 \times \tfrac{1}{2} = 0.4 \\
i = 2: &\quad \text{degree} = 2 \Rightarrow \text{dest} = \{3,4\} \\
&\quad r^{new}(3,4) += 0.8 \times \tfrac{1}{2} = 0.4
\end{aligned}$$

Therefore:

$$r^{new} = \begin{cases}
r^{new}(0) = 0.0333 + 0.2 + 0.4 = 0.6333 \\
r^{new}(1) = 0.0333 + 0.2 = 0.2333 \\
r^{new}(2) = 0.0333 \\
r^{new}(3) = 0.0333 + 0.2 + 0.4 = 0.6333 \\
r^{new}(4) = 0.0333 + 0.4 = 0.4333 \\
r^{new}(5) = 0.0333 + 0.2 + 0.4 = 0.6333
\end{cases}$$

(2) Block-Based and Block-Stripe Methods

- In the **block-based method**, r^{new} is divided into k parts that fit in memory, scanning M for each block. Communication cost increases linearly with k.
- In the **block-stripe method**, M is divided into stripes matching destination ranges. Each stripe is smaller but duplicated, trading off storage overhead (ε) for reduced scanning cost.

(3) Cost Analysis
Reading r^{old} and M, and writing r^{new} per iteration contributes to total cost:

$$2|r| + |M|$$

When M is striped and repeated across blocks, the adjusted cost becomes:

$$|M|(1+\varepsilon) + (k+1)|r|$$

where ε accounts for repeated storage of out-degrees.

Question 2

A spammer aims to artificially increase the PageRank of a target page t by constructing a **link farm**. The spammer owns M pages and tries to maximize t's PageRank using inbound links from these controlled pages.

1. Using the recursive relationship between y and the farm pages, derive the closed-form expression for y:
$$y = \frac{x}{1-\beta^2} + c\frac{M}{N}, \quad \text{where } c = \frac{\beta}{1+\beta}.$$
2. For $\beta = 0.85$, compute the numerical multiplier $\frac{1}{1-\beta^2}$ and discuss the effect of increasing M on the PageRank of t.

Relevant Concepts

- **Spammer's goal:** Maximize the PageRank of target page t.
- **Technique:** Acquire inbound links from as many accessible pages as possible, and construct a link farm of M owned pages pointing to t.
- **PageRank of farm pages:**
$$\text{Rank(farm)} = \frac{\beta y}{M} + \frac{1-\beta}{N}.$$
where:
  - x = PageRank contributed by accessible (external) pages
  - β = damping factor
  - M = number of spammer-owned pages
  - N = total number of pages on the web

Answer

(1) Recursive equation for target page t:

$$\text{Rank(farm)} = \frac{\beta y}{M} + \frac{1-\beta}{N}.$$

$$y = x + \beta M\left(\frac{\beta y}{M} + \frac{1-\beta}{N}\right) + \frac{1-\beta}{N}.$$

Simplifying:

$$y = x + \beta^2 y + \frac{\beta(1-\beta)M}{N} + \frac{1-\beta}{N}.$$

Ignoring the small term $\frac{1-\beta}{N}$, we solve for y:

$$y(1-\beta^2) = x + \frac{\beta(1-\beta)M}{N}.$$

Thus,

$$y = \frac{x}{1-\beta^2} + \frac{\beta}{1+\beta}\frac{M}{N} = \frac{x}{1-\beta^2} + c\frac{M}{N} \quad \text{(shown)}$$

(2) For $\beta = 0.85$:

$$\frac{1}{1-\beta^2} = \frac{1}{1-0.7225} = \frac{1}{0.2775} \approx 3.6.$$

Therefore, increasing M (the number of farm pages) increases y linearly, demonstrating the **PageRank multiplier effect**.

Lecture 7: Graphs

Question 1

Consider the graph below (a line graph $A - B - C$), where each node's degree d_u and neighbors are indicated:

Node	Degree (d_u)	Neighbors
A	1	$\{B\}$
B	2	$\{A, C\}$
C	1	$\{B\}$

We will run the **Approximate PPR algorithm** with parameters:

$$\beta = 0.5, \quad \varepsilon = 0.20$$

and seed node $S = A$.

Relevant Concepts

- **Lazy random walk interpretation:** Each push simulates a lazy random walk: stay put with probability $\frac{1}{2}$ and move to each neighbor with probability $\frac{1}{2d_u}$.
- **Residual update:** The residual q_u tracks unpropagated PageRank mass. A large ratio $\frac{q_u}{d_u}$ means u must push next.
- **Push operation** from node u:

$$\begin{aligned} r'_u &= r_u + (1-\beta)q_u, \\ q'_u &= \frac{1}{2}\beta q_u, \\ q'_v &= q_v + \frac{1}{2}\beta\frac{q_u}{d_u}, \quad v \in N(u). \end{aligned}$$

- **Termination criterion:**

$$\max_u \frac{q_u}{d_u} < \varepsilon.$$

Answer

Initialization:

$$r = [0, 0, 0], \qquad q = [1, 0, 0].$$

Step 1: Push from node A

$$\begin{aligned} r'_A &= 0 + (1 - 0.5)(1) = 0.5, \\ q'_A &= 0.5 \cdot 0.5(1) = 0.25, \\ q'_B &= 0 + 0.5 \cdot 0.5 \cdot \frac{1}{1} = 0.25, \\ q'_C &= 0 \quad \text{(Not a neighbor of A)}. \end{aligned}$$

$$r = [0.5, 0, 0], \qquad q = [0.25, 0.25, 0].$$

Update ratios:

$$\frac{q_A}{d_A} = \frac{0.25}{1} = 0.25, \qquad \frac{q_B}{d_B} = \frac{0.25}{2} = 0.125, \qquad \frac{q_C}{d_C} = 0.$$

Check termination:

$$\max_u \frac{q_u}{d_u} = 0.25.$$

Since $0.25 \geq \varepsilon = 0.20$, the algorithm continues. Node A has the highest ratio, so we push A again.

Step 2: Push from node A again

$$
\begin{aligned}
r'_A &= 0.5 + 0.5(0.25) = 0.625, \\
q'_A &= 0.5 \cdot 0.5(0.25) = 0.0625, \\
q'_B &= 0.25 + 0.5 \cdot 0.5 \cdot \frac{0.25}{1} = 0.25 + 0.0625 = 0.3125, \\
q'_C &= 0.
\end{aligned}
$$

$$
r = [0.625, 0, 0], \qquad q = [0.0625, 0.3125, 0].
$$

Update ratios:

$$
\frac{q_A}{d_A} = \frac{0.0625}{1} = 0.0625, \qquad \frac{q_B}{d_B} = \frac{0.3125}{2} = 0.15625, \qquad \frac{q_C}{d_C} = 0.
$$

Check termination:

$$
\max_u \frac{q_u}{d_u} = 0.15625.
$$

Since $0.15625 < \varepsilon = 0.20$, the algorithm terminates.

Final Approximate PageRank Vector

$$
r \approx [0.625,\ 0.00,\ 0.00].
$$

Interpretation

- **Why it stopped:** By Push 2, the residual mass at A was converted to PageRank r_A, and the mass pushed to B was divided by B's larger degree ($d_B = 2$), keeping B's ratio below the threshold.
- **Mass distribution:** Node A holds the majority of the computed PageRank. Node B has accumulated residual mass but has not yet converted it to permanent PageRank ($r_B = 0$).
- **Efficiency:** The high degree of neighbor B helped dampen the propagation, allowing early termination.

Lecture 8: Streams

Question 1

Exponentially decaying windows provide a smooth way to track the "current" importance of items in a data stream. Instead of keeping the last N elements, we maintain a weighted sum where recent items receive higher weight and older items decay smoothly over time.
Suppose a data stream is given by values

$$
a_1, a_2, a_3, \ldots
$$

and we maintain the exponentially decaying score at time t:

$$
S_t = \sum_{i=1}^{t} a_i (1-c)^{\,t-i},
$$

where c is a small constant such as 10^{-6} or 10^{-9}.
The score can be updated incrementally when a new element a_{t+1} arrives using:

$$
S_{t+1} = (1-c)S_t + a_{t+1}.
$$

Consider the following concrete stream of values:

$$
a = [5,\ 2,\ 4,\ 8,\ 10], \qquad c = 0.1.
$$

Relevant Concepts

- **Exponentially decaying weights:** Older elements are multiplied by $(1-c)^k$, where k is the number of time steps since they appeared.
- **Smooth aggregation:** Unlike a sliding window, the decaying window smoothly reduces the influence of the past rather than discarding it abruptly.
- **Implementation efficiency:** The incremental update

  $$S_{t+1} = (1-c)S_t + a_{t+1}$$

  requires only constant memory and constant time per update.
- **Effective window length:** Since

  $$\sum_{t=0}^{\infty}(1-c)^t = \frac{1}{c},$$

  the decaying window behaves like a sliding window of effective size approximately $1/c$.

Answer

Step-by-step update of the exponentially decaying sum:

$$S_1 = a_1 = 5,$$
$$S_2 = (1-c)S_1 + a_2 = 0.9(5) + 2 = 6.5,$$
$$S_3 = 0.9(6.5) + 4 = 9.85,$$
$$S_4 = 0.9(9.85) + 8 = 16.865,$$
$$S_5 = 0.9(16.865) + 10 = 25.1785.$$

Thus, the exponentially decaying score after processing the five items is:

$$S_5 \approx 25.18.$$

Lecture 9: Web Advertising

Question 1

Consider a bipartite graph $G = (B, G, E)$ with boys B on the left and girls G on the right. A simple greedy matching algorithm processes edges in arbitrary order and adds an edge to the matching whenever both endpoints are currently unmatched.
We wish to prove that the greedy matching is always at least **half as large** as the maximum matching:

$$|M_{\text{greedy}}| \;\geq\; \frac{1}{2}\,|M_{\text{opt}}|.$$

Relevant Concepts

- **Greedy matching:** Each time an edge is considered, it is added if neither endpoint is matched yet.
- **Optimal matching M_{opt}:** A maximum-cardinality matching in the bipartite graph.
- **Set G:** The set of girls who are matched in M_{opt} but not in M_{greedy}.
- **Set B:** The set of boys who are adjacent to the girls in G.
- **Key observations:**

  1. By definition of G:

     $$|M_{\text{opt}}| = |M_{\text{greedy}}| + |G|.$$

2. Each boy in B must already be matched in M_{greedy}, otherwise greedy would have matched him with the adjacent unmatched girl in G. Hence:

$$|M_{\text{greedy}}| \geq |B|.$$

3. In the optimal matching, every girl in G is matched to some boy in B, so:

$$|G| \leq |B|.$$

Answer

We prove that $|M_{\text{greedy}}| \geq \frac{1}{2}|M_{\text{opt}}|$.
Step 1: Relate M_{opt} and M_{greedy}. By definition, the girls in G are unmatched in the greedy matching but matched in the optimal matching. Thus:

$$|M_{\text{opt}}| = |M_{\text{greedy}}| + |G|. \tag{1}$$

Step 2: Define the adjacent boys. Let B be the set of boys adjacent to girls in G. If any such boy were unmatched by the greedy algorithm, then when the edge connecting him to a girl in G was considered, greedy would have matched them. Thus every boy in B is matched in M_{greedy}, giving:

$$|B| \leq |M_{\text{greedy}}|. \tag{2}$$

Step 3: Relate G and B. In the optimal matching, every girl in G must be matched to some boy in B. Therefore,

$$|G| \leq |B|. \tag{3}$$

Step 4: Combine inequalities. From (2) and (3):

$$|G| \leq |B| \leq |M_{\text{greedy}}|. \tag{4}$$

Substituting (4) into (1):

$$|M_{\text{opt}}| = |M_{\text{greedy}}| + |G| \leq |M_{\text{greedy}}| + |M_{\text{greedy}}| = 2|M_{\text{greedy}}|.$$

Thus,

$$|M_{\text{greedy}}| \ \geq \ \frac{1}{2}\,|M_{\text{opt}}|$$

This proves that the greedy matching algorithm always achieves a 1/2-approximation to the maximum matching.

Question 2

Consider two advertisers A_1 and A_2, each with equal budget B. Queries arrive one by one, each query being eligible for both advertisers. The BALANCE algorithm always assigns an arriving query to the advertiser with the *most remaining budget*.
Assume the offline optimal solution (OPT) assigns all queries to advertisers, exhausting both budgets:

$$\text{Revenue(OPT)} = 2B.$$

Your goal is to analyze BALANCE and prove the following guarantee:

$$\text{Revenue(BAL)} \ \geq \ \left\lfloor \frac{3B}{2} \right\rfloor.$$

Then, extend the reasoning to the general case with N advertisers and show that BALANCE has competitive ratio:

$$1 - \frac{1}{e}.$$

Relevant Concepts

- **Exhausting a budget:** BALANCE must exhaust at least one advertiser's budget. Otherwise, if a query becomes unassigned while both budgets are still positive, BALANCE would have assigned it earlier, contradicting termination.
- **Unused budget:** Let x be the amount of budget left unused by BALANCE for advertiser A_2. The remaining $B - x$ units must have been used on queries assigned to A_2.
- **Revenue of BALANCE:**

  $$\text{Revenue(BAL)} = (B - x) + B = B + y,$$

  where $y = B - x$ is the amount assigned to A_1 by BALANCE.
- **Key inequality:** To show Revenue(BAL) $\geq \frac{3B}{2}$, it suffices to prove:

  $$y \geq x.$$

  When $x = y = B/2$, BALANCE has its *minimum* possible revenue:

  $$\text{Revenue(BAL)} = \frac{3B}{2}.$$

- **General case (N advertisers):** BALANCE distributes queries across advertisers in N rounds. The sum of allocations to advertisers in rounds produces a harmonic sum:

  $$H_n = \sum_{i=1}^{n} \frac{1}{i} \approx \ln n.$$

  The critical number of rounds k after which no advertiser can accept more queries satisfies:

  $$H_{N-k} = \ln N - 1 = \ln\left(\frac{N}{e}\right).$$

  Thus:

  $$N - k = \frac{N}{e} \quad \Rightarrow \quad k = N\left(1 - \frac{1}{e}\right).$$

- **Competitive ratio:**

  $$\frac{\text{Revenue(BAL)}}{\text{Revenue(OPT)}} = 1 - \frac{1}{e}.$$

  No online algorithm can obtain a better ratio.

Answer

Step 1: BALANCE must exhaust at least one budget. Suppose BALANCE leaves both advertisers with some unused budget. Then there exists a query q that remains unassigned even though both A_1 and A_2 have available budget. At the moment q arrived, BALANCE would have assigned it to the advertiser with larger remaining budget. This is a contradiction. Therefore, BALANCE exhausts at least one advertiser's budget. Without loss of generality, assume BAL exhausts A_2.
Step 2: Define unused budget x. Let x be the unused budget of advertiser A_2. Then A_2 received $B - x$ units of queries. Let:

$$y = B - x.$$

Step 3: Revenue expression. Total revenue collected by BALANCE is:

$$\text{Revenue(BAL)} = \underbrace{y}_{A_1} + \underbrace{B}_{A_2} = B + y.$$

Step 4: Prove $y \geq x$. Consider the B queries assigned by OPT to advertiser A_1. BALANCE must also assign all these B queries, because each has at least advertiser A_1 as a bidder.

- **Case 1:** At least $B/2$ of these are assigned by BAL to A_1. Then:

$$y \geq B/2 \geq x,$$

since $x \leq B/2$.

- **Case 2:** More than $B/2$ of these are assigned by BAL to A_2. Let q be the last such query assigned to A_2. At that moment, the remaining budget of A_1 must be *no larger* than that of A_2 (because BAL assigns to the advertiser with *more* remaining budget). But A_2 has at most $B/2$ remaining budget at this point. Thus:

$$x \leq B/2.$$

And since $y = B - x \geq B/2$, again:

$$y \geq x.$$

Thus in both cases,

$$y \geq x.$$

Step 5: Minimum revenue. The revenue is:

$$\text{Revenue(BAL)} = B + y.$$

Since $y \geq x$ and $x + y = B$, the minimum occurs at $x = y = B/2$, giving:

$$\text{Revenue(BAL)} = B + \frac{B}{2} = \frac{3B}{2}.$$

Step 6: General case (N advertisers). In the N-advertiser example:

$$k = N\left(1 - \frac{1}{e}\right)$$

rounds of queries are successfully allocated, giving revenue:

$$\text{Revenue(BAL)} = BN\left(1 - \frac{1}{e}\right).$$

The optimal revenue is BN, so:

$$\frac{\text{Revenue(BAL)}}{\text{Revenue(OPT)}} = 1 - \frac{1}{e}.$$

Final Result:

BALANCE is $\left(1 - \frac{1}{e}\right)$-competitive, and achieves at least $\frac{3B}{2}$ in the two-advertiser case.

Question 3

Hoeffding's inequality provides a high–probability bound on how far the empirical mean of i.i.d. random variables can deviate from its true expectation.
Let $X_1, \ldots, X_m$ be i.i.d. random variables taking values in $[0, 1]$, and let

$$\mu = \mathbb{E}[X], \qquad \widehat{\mu}_m = \frac{1}{m}\sum_{i=1}^{m} X_i.$$

Suppose we want to determine a confidence interval width b such that

$$\Pr\left(|\mu - \widehat{\mu}_m| \geq b\right) \leq \delta,$$

and evaluate the bound when b is chosen as

$$b(a, T) = \sqrt{\frac{2\log(T)}{m_a}},$$

where m_a is the number of times an action a has been played in a multi–armed bandit algorithm.

Relevant Concepts

- **Hoeffding's Inequality:** For i.i.d. variables in $[0, 1]$,

$$\Pr\left(|\mu - \widehat{\mu}_m| \geq b\right) \leq 2\exp(-2b^2 m).$$

- **Confidence parameter:** Setting the RHS equal to δ gives:

$$b \geq \sqrt{\frac{\ln(2/\delta)}{2m}}.$$

- **Application to bandits:** If the confidence radius is chosen as

$$b(a, T) = \sqrt{\frac{2\log(T)}{m_a}},$$

  then Hoeffding's bound implies:

$$\Pr\left(|\mu - \widehat{\mu}_{m_a}| \geq b(a, T)\right) \leq 2T^{-4},$$

  which decays extremely quickly as T grows.

- **Exploration guarantee:** If an action is not played often, then m_a is small and $b(a, T)$ becomes large, ensuring the algorithm never prematurely eliminates an arm.

Answer

Starting from Hoeffding's inequality,

$$\Pr\left(|\mu - \widehat{\mu}_m| \geq b\right) \leq 2\exp(-2b^2 m).$$

To obtain a confidence interval of width b with error probability δ, set:

$$2\exp(-2b^2 m) = \delta.$$

Taking logarithms:

$$-2b^2 m = \ln(\delta/2) \quad \Rightarrow \quad b^2 = \frac{\ln(2/\delta)}{2m},$$

and thus

$$b \geq \sqrt{\frac{\ln(2/\delta)}{2m}}.$$

Applying the bandit–style confidence radius. Let

$$b(a, T) = \sqrt{\frac{2\log(T)}{m_a}}.$$

Plugging this into Hoeffding:

$$\Pr\left(|\mu - \widehat{\mu}_{m_a}| \geq b(a, T)\right) \leq 2\exp\left(-2 \cdot \frac{2\log(T)}{m_a} \cdot m_a\right) = 2\exp(-4\log(T)) = 2T^{-4}.$$

Since $2T^{-4}$ decays extremely fast:

- The probability that the confidence bound is incorrect becomes negligible for large T.
- Actions with small m_a retain large confidence radii, forcing continued exploration.
- The upper confidence bound (UCB) strategy remains optimistic and prevents discarding potentially good actions.

Thus, Hoeffding's inequality ensures strong control over uncertainty and underlies the theoretical guarantees of UCB algorithms.