CS 466/666 Algorithm Design and Analysis

Keven Qiu
Instructor: Rafael Oliveira
Spring 2023

Contents

Part I

Amortized Analysis

Chapter 1

Amortized Analysis

1.1 Introduction

Definition: Amortized Analysis

Average of the total time required to perform a sequence of data-structure operations over all operations performed.
Amortized analysis is a **worst-case** analysis.

Types of Amortized Analysis

1. **Aggregate Analysis**: Upper bound $T(n)$ on total cost of sequence of n operations. Amortized complexity is $T(n)/n$.
2. **Accounting Method**: Assign certain charges to each operation. If the operation is cheaper than the charge, then build up credit to use later.
3. **Potential Method**: A potential energy of a data structure, which maps each state of the entire data structure to $\mathbb{R}$ (potential). Assign credit to the whole data structure instead of each operation in accounting method.

Problem

Input: A binary counter C initially set to 0.
Output: Increment this counter up to n.
Question: How many bit operations will it take to increment C from 0 to n?

Aggregate Analysis: The worst-case time per operation is $\log n$. So an upper bound is $O(n \log n)$.

The most significant bits get updated very infrequently. Flip the kth bit after 2^{k-1} opera-

tions/increments.

$$\#\text{ bit flips} = \sum_{k=0}^{\lceil \log n \rceil} \left\lfloor \frac{n}{2^k} \right\rfloor < \sum_{k \geq 0} \frac{n}{2^k} = 2n$$

The total time is $\Theta(n)$. This implies that the amortized cost per operation is $\Theta(1)$.

Accounting Method: Suppose the actual cost of each operation of an algorithm is c_i. In each step of the algorithm, we assign charges γ_i to each operation such that

$$\sum_{i=1}^{l} \gamma_i \geq \sum_{i=1}^{l} c_i$$

for any $l \geq 1$. That is, the total charged up to step l is greater than or equal to the actual cost of all operations up to that point.

Suppose we charge the cost of clearing a bit ($1 \to 0$) to the operation that sets the bit to 1 in the first place. If we flip k bits during an increment, we have already charged $k-1$ of those bit flips to earlier bit flips. Note that if we flip k bits, we must set $k-1$ of these bits to 0, so that it carries over.

So instead of paying for k bit flips, we charge at most 2: one for setting a bit to 1 and the other is to clear this bit.

So then total cost $\leq$ total charged $= 2n$.

Potential Method: Suppose the actual cost of each operation of our algorithm is c_i. Assign potential Φ_i to data structure at time i. Amortized cost of ith operation is

$$\gamma_i = c_i + \Phi_i - \Phi_{i-1}$$

That is, total amortized cost is the actual cost of the operation plus the change in potential. We have

$$\sum_{i=1}^{n} \gamma_i = \sum_{i=1}^{n} (c_i + \Phi_i - \Phi_{i-1}) = \Phi_n - \Phi_0 + \sum_{i=1}^{n} c_i$$

So if $\Phi_k - \Phi_0 \geq 0$ for all $k \geq 0$, the total amortized cost is an upper bound on total cost.

Potential: Φ_i = number of bits with value 1 at step i for $i \geq 0$. Now for every $k \geq 0$, $\Phi_k \geq \Phi_0 = 0$.

$c_i = (\#\text{ bits } 0 \to 1) + (\#\text{ bits } 1 \to 0)$.
$\Phi_i - \Phi_{i-1} = (\#\text{ bits } 0 \to 1) - (\#\text{ bits } 1 \to 0)$.

Amortized cost:

$$\gamma_i = c_i + \Phi_i - \Phi_{i-1} = 2 \times (\#\text{ bits } 0 \to 1) = 2$$

since each increment only changes 1 bit from 0 to 1, each amortized cost is 2. So,

$$\sum_{i=1}^{n} c_i + \Phi_n - \Phi_0 = \sum_{i=1}^{n} \gamma_i = \sum_{i=1}^{n} 2 = 2n$$

1.2 Splay Trees

Data structures with great amortized running time are great for internal processes, such as internal graph algorithms (e.g. MST). It is bad when you have client-server model, as in this setting, one wants to minimize worst-case per query.

Definition: Splay Trees

Self-adjusting binary search trees.

Theorem (Sleator & Tarjan 1985)

Splay trees have $\Theta(\log n)$ amortized cost per operation., $\Theta(n)$ worst-case time.

Does not keep any balancing information. Adjust the tree whenever a node is accessed.

Definition: Splaying

Move the node that was searched to the root.

Notation

$n \leftarrow$ number of elements, $m \leftarrow$ number of operations = searches + insertions + deletions.
Operations: $SEARCH(k)$, $INSERT(k)$, $DELETE(k)$

1.2.1 Splay Operations

Definition: Splay Operation

$SPLAY(k)$
Input: element k
Output: rebalancing of binary search tree

Definition: Zig-Zag Condition

$parent(k)$ has k as left/right-child and $parent(parent(k))$ has $parent(k)$ as right/left-child.

Definition: Zig-Zig Condition

$parent(k)$ has k as left/right-child and $parent(parent(k))$ has $parent(k)$ as left/right-child.

Only apply zig rotation when there is no grandparent of the node we are rotating.

Algorithm 1 $SPLAY(k)$

```
1: while k is not root do
2:     if k satisfies zig-zag condition then
3:         zig-zag rotation
4:     if k satisfies zig-zig condition then
5:         zig-zig rotation
6:     if k is a child of root then
7:         zig rotation (normal)
```

1.2.2 Splay Tree Algorithm

$SEARCH(k)$: after searching for k, if k in the tree, do $SPLAY(k)$. If k is not in the tree, then perform $SPLAY(k')$ where k' is the last key that was compared to k when searching.

$INSERT(k)$: standard insert operation, then do $SPLAY(k)$

$DELETE(k)$: standard delete operation, then $SPLAY(parent(k))$

- delete first (moves k to the bottom of tree) by finding successor
- then delete k as in the cases where k has at most one child
- then we splay the parent of k (after placing k at the bottom)

Intuition: zig-zag and zig-zig make a lot of progress in unbalanced trees. If the tree is balanced, then splaying is quite fast.

1.2.3 Analysis

Potential Method: The charge $\hat{c}_i$ of the ith operation with respect to the potential function Φ is:

$$\hat{c}_i := c_i + \Phi(D_i) - \Phi(D_{i-1})$$

The amortized cost of all operations is

$$\begin{aligned}\sum_{i=1}^{m} \hat{c}_i &= \sum_{i=1}^{m} c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= \Phi(D_m) - \Phi(D_0) + \sum_{i=1}^{m} c_i \geq \sum_{i=1}^{m} c_i\end{aligned}$$

So long as $\Phi(D_m) \geq \Phi(D_0)$, then amortized charge is an upper bound on amortized cost.

Definition: Potential Function of Splay Tree

- $\delta(k) :=$ number of descendants of k (including k)
- $\text{rank}(k) := \log(\delta(k))$
- $$\Phi(T) = \sum_{k \in T} \text{rank}(k)$$

The minimum potential is with a perfectly balanced tree. The root has rank $\log n$, any node in second level has rank $\log \frac{n}{2}$, and so forth. So

$$\Phi(T) = \sum_{h=1}^{\log n} h \cdot \frac{n}{2^h} = \Theta(n)$$

Analysis - Splay Operation: Let $\text{rank}(k)$ be the current rank of k and $\text{rank}'(k)$ be the new rank of k after we perform a rotation on k.

Lemma (Amortized Cost from *SPLAY* Subroutines)

The charge γ of an operation (zig, zig-zig, zig-zag) is bounded by:

$$\gamma \leq \begin{cases} 3(\text{rank}'(k) - \text{rank}(k)) & \text{for zig-zig, zig-zag} \\ 3(\text{rank}'(k) - \text{rank}(k)) + 1 & \text{for zig} \end{cases}$$

Proof. Let T' be the tree after rotation.

We begin by analyzing the zig rotation. Let k be the node we rotating, $b = \textit{parent}(k)$. Then, $\text{rank}'(k) = \text{rank}(b)$. The charge in this case is given by

$$\begin{aligned} \gamma &= \text{cost} + \Phi(T') - \Phi(T) \\ &= 1 + \text{rank}'(k) + \text{rank}(b) - \text{rank}(k) - \text{rank}(b) \\ &= 1 + \text{rank}'(b) - \text{rank}(k) \\ &\leq 1 + \text{rank}'(k) - \text{rank}(k) \\ &\leq 1 + 3(\text{rank}'(k) - \text{rank}(k)) \end{aligned}$$

where $\text{rank}'(b) \leq \text{rank}'(k)$ since b is a child of k in T'.

Analyzing the zig-zag rotation. Let k be the node we are rotating, $b = \textit{parent}(k)$, and $a = \textit{parent}(b)$. Then, $\text{rank}'(k) = \text{rank}(a)$, $\text{rank}'(b) \leq \text{rank}'(k)$, and $\text{rank}(k) \leq \text{rank}(b) \leq \text{rank}(a)$. Moreover, $\delta'(k) \geq \delta'(a) + \delta(k)$. The charge in this case is given by

$$\begin{aligned} \gamma &= \text{cost} + \Phi(T') - \Phi(T) \\ &= 2 + \text{rank}'(a) + \text{rank}'(b) + \text{rank}'(k) - \text{rank}(a) - \text{rank}(b) - \text{rank}(k) \\ &= 2 + \text{rank}'(a) + \text{rank}'(b) - \text{rank}(b) - \text{rank}(k) \\ &\leq 2 + \text{rank}'(a) + \text{rank}'(k) - 2 \cdot \text{rank}(k) \end{aligned}$$

Now since $\delta'(k) \geq \delta'(a) + \delta(k)$, by concavity of log, we have

$$\log\left(\frac{\delta'(a)}{\delta'(k)}\right) + \log\left(\frac{\delta(k)}{\delta'(k)}\right) \leq -2 \implies \log\delta'(a) + \log\delta(k) \leq 2\log\delta'(k) - 2$$

Equivalently, we have $\text{rank}'(a) \leq 2 \cdot \text{rank}'(k) - \text{rank}(k) - 2$. Thus, we have

$$\gamma \leq 2 + \text{rank}'(a) + \text{rank}'(k) - 2 \cdot \text{rank}(k) \leq 3(\text{rank}'(k) - \text{rank}(k))$$

The proof for zig-zig is similar to zig-zag.

Lemma (Total Amortized Cost of $SPLAY(k)$)

Let T be our current tree, with root t and k be a node in this tree. The charge Γ of the $SPLAY(k)$ operation is bounded by:

$$\Gamma \leq 3(\text{rank}(t) - \text{rank}(k)) + 1 \leq 3\,\text{rank}(t) + 1 = O(\log n)$$

Proof. Note that Γ is the sum of the charges of the basic rotations performed during the $SPLAY(k)$ operation. Let γ_i be the charge of the ith rotation, and $\text{rank}_i(k)$ be the rank of k after the ith rotation. Then, we have $\text{rank}_0(k) = \text{rank}(k)$, $\text{rank}_l(k) = \text{rank}(t)$, where l is the number of basic rotations performed during the $SPLAY(k)$ operation.

There is only one zig operation so we add 1. Thus,

$$\Gamma = \sum_{i=1}^{l} \gamma_i \leq 1 + \sum_{i=1}^{l} 3(\text{rank}_i(k) - \text{rank}_{i-1}(k)) = 1 + 3(\text{rank}(t) - \text{rank}(k))$$

where the inequality comes from the previous lemma.

Analysis - Amortized Cost: For each of the 3 operations we have:

$$\text{charge per operation} = (\text{charge of } SPLAY) + (\text{potential change not from } SPLAY)$$

The charge of $SPLAY$ is $O(\log n)$ from second lemma. Charge of $SPLAY$ already includes the cost of the operation.

Tracking potential outside splay:

- $SEARCH$: only splay changes the potential
- $DELETE$: removing a node decreases potential
- $INSERT$: adding new element k increases ranks of all ancestors of k post insertion (might be $O(n)$ of them)

So we need to handle case 3: Let $k := k_0 \mapsto k_1 \mapsto \cdots \mapsto k_l$ where $k - I$ is the ith ancestor of k and k_l is the root of the tree. If we denote $\delta'(a)$ as the number of descendants of a

post insertion, then we have $\delta'(k) = 1$ (since before splaying, k is a leaf of the tree) and $\delta'(k_i) = \delta(k_i) + 1$ for $1 \leq i \leq l$. Hence, the change in potential is:

$$\sum_{i=1}^{l} \log\left(\frac{\delta'(k_i)}{\delta(k_i)}\right) = \sum_{i=1}^{l} \log\left(\frac{\delta(k_i)+1}{\delta(k_i)}\right) \leq \sum_{i=1}^{n} \log\left(\frac{i+1}{i}\right) = \log(n+1) = O(\log n)$$

Thus, the amortized cost of each operation is $O(\log n)$, since we upper bounded each of the 3 quantities (charge of splay, cost of operation, potential change not from splay) by $O(\log n)$.

To show our potential function is valid, the initial potential is 0 for the empty tree and the potential is always nonnegative (sum of logarithms).

Dynamic Optimality Conjecture (Sleator & Tarjan 1985)

Splay trees are optimal within a constant in a very strong sense:
Given a sequence of items to search for $a_1, \ldots, a_m$, let OPT be the minimum cost of doing these searches + any rotations you like on the binary search tree.
You can charge 1 for following tree pointer (parent $\rightarrow$ child or child $\rightarrow$ parent), charge 1 per rotation.
Conjecture: Cost of splay tree is $O(OPT)$.

Part II

Randomized Algorithms

Chapter 2

Concentration Inequalities

When evaluating the performance of randomized algorithms, we want to not only analyze the expected runtimes, but also if the algorithm runs in time close to its expected runtime *most of the time*.

A small runtime with high probability is *better than* small expected runtime.

2.1 Markov's Inequality

Theorem (Markov's Inequality)

Let X be a nonnegative discrete random variable. Then

$$P[X \geq t] \leq \frac{\mathbb{E}[X]}{t}, \ \forall t > 0$$

Proof.

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{n \geq 0} P[X = n] \cdot n \\
&= \sum_{n=0}^{t-1} P[X = n] \cdot n + \sum_{n \geq t} P[X = n] \cdot n \\
&\geq 0 + t \cdot \sum_{n \geq t} P[X = n] \\
&= t \cdot P[X \geq t]
\end{aligned}$$

Quicksort: The expected runtime of quicksort is $2n \log n$. Markov's inequality tells us that the runtime is at least $2cn \log n$ with probability $\leq 1/c$, for any $c \geq 1$.

Coin Flipping: If we flip n fair coins, the expected number of heads is $n/2$. Markov's inequality tells that $P[\#\text{ heads} \geq 3n/4] \leq 2/3$.

Remark: Useful when there is no information beyond expected value.

2.2 Chebyshev's Inequality

Definition: Variance

$$Var[X] := \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$$

Definition: Standard Deviation

$$\sigma(X) := \sqrt{Var[X]}$$

Theorem (Chebyshev's Inequality)

Let X be a discrete random variable. Then

$$P[|X - \mathbb{E}[X]| \geq t] \leq \frac{Var[X]}{t^2}, \ \forall t > 0$$

Proof. We only know Markov's inequality. Let $Y = (X - \mathbb{E}[X])^2$. Then,

$$Y \begin{cases} \text{discrete} & \text{if } X \text{ discrete} \\ \geq 0 \end{cases}$$

so we can use Markov's inequality.

$$P[Y \geq t^2] = P[|X - \mathbb{E}[X]| \geq t] \leq \frac{\mathbb{E}[Y]}{t^2} = \frac{Var[X]}{t^2}$$

Definition: Covariance

The covariance of two random variables X, Y is defined as

$$Cov[X, Y] := \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$$

Note that $Cov[X, X] = Var[X]$.
We say that X, Y are positively correlated if $Cov[X, Y] > 0$ and negatively correlated if $Cov[X, Y] < 0$.

Independent random variables are uncorrelated, but uncorrelated random variables are not necessarily independent.

Proposition

- $Var[X + Y] = Var[X] + Var[Y] + 2Cov[X, Y]$
- If X, Y are independent, then $Var[X + Y] = Var[X] + Var[Y]$

Coin Flipping: If X be # heads in n independent unbiased flips, let us bound $P[X \geq 3n/4]$.

$$X_i = \begin{cases} 1 & \text{if heads} \\ 0 & \text{otherwise} \end{cases}$$

Then, $X = \sum_{i=1}^{n} X_i$ and each X_i, X_j are independent.

$$\mathbb{E}[X_i] = 1 \cdot \frac{1}{2} + 0 \cdot \frac{1}{2} = \frac{1}{2}$$

and

$$Var[X_i] = \mathbb{E}[(X_i - \mathbb{E}[X_i])^2] = \frac{1}{2}\left(1 - \frac{1}{2}\right)^2 + \frac{1}{2}\left(0 - \frac{1}{2}\right)^2 = \frac{1}{4}$$

So by proposition,

$$Var[X] = \sum_{i=1}^{n} Var[X_i] = \frac{n}{4}$$

By Chebyshev's inequality,

$$P\left[X \geq \frac{3n}{4}\right] \leq P\left[\left|X - \frac{n}{2}\right| \geq \frac{n}{4}\right] \leq \frac{n/4}{(n/4)^2} = \frac{4}{n}$$

Chebyshev's inequality is most useful when we only have information about the second moment of X.

Definition: kth Moment

The kth moment of a random variable X is $\mathbb{E}[X^k]$.

Definition: kth Central Moment

The kth central moment of a random variable X is

$$\mu_X^{(k)} := \mathbb{E}[(X - \mathbb{E}[X])^k]$$

if it exists.

Definition: i.i.d.

Independent and identically distributed.

Definition: Law of Large Numbers

Average of i.i.d. variables is approximately the expectation of the random variables.

$$\frac{1}{n} \cdot \sum_{i=1}^{n} X_i \approx \mathbb{E}[X]$$

2.3 Chernoff Bounds

Definition: Chernoff Bounds

Give quantitative estimates of the probability that X is far from $\mathbb{E}[X]$ for any value of n, when $X = \sum_{i=1}^{n} X_i$.

Definition: Moment Generating Function

$$M_X(t) := \mathbb{E}[e^{tX}] = \mathbb{E}\left[\sum_{k\geq 0} \frac{t^k}{k!} \cdot X^k\right] = \sum_{k\geq 0} \frac{t^k \mathbb{E}[X^k]}{k!}$$

Theorem (Chernoff Inequality)

Let $X_1, \ldots, X_n$ be independent variables such that $X_i \in 0, 1$ for all $i \in [n]$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X]$. Then, for $\delta > 0$,

$$P[X \geq (1+\delta)\mu] \leq \left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu}$$

and

$$P[X \leq (1-\delta)\mu] \leq \left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^{\mu}$$

Theorem (Chernoff Inequality - $0 < \delta < 1$)

Let $X_1, \ldots, X_n$ be independent variables such that $X_i \in 0, 1$ for all $i \in [n]$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X]$. Then, for $0 < \delta < 1$,

$$P[X \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/3}$$

and

$$P[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}$$

Proof. We will prove the first inequality. Let $p_i := P[X_i = 1]$ and thus, $P[X_i = 0] = 1 - p_i$ and $\mu = \sum_{i=1}^{n} p_i$.

Idea is to use Markov's inequality to the random variable e^{tX}. Since the exponential function is increasing, we have

$$P[X \geq a] = P[e^{tX} \geq e^{ta}] \leq \frac{\mathbb{E}[e^{tX}]}{e^{ta}}, \ \forall t > 0$$

When we use the exponential function, we are using information about all moments of X. The Taylor series of e^{tX} is

$$e^{tX} = \sum_{k\geq 0} \frac{(tX)^k}{k!} = 1 + tX + \frac{t^2X^2}{2!} + \frac{t^3X^3}{3!} + \cdots$$

If $X = X_1 + X_2$ where X_1, X_2 are independent, then $M_X(t) = M_{X_1}(t)M_{X_2}(t)$.

Apply Markov's inequality to e^{tX}.

$$P[X \geq (1+\delta)\mu] = P[e^{tX} \geq e^{t(1+\delta)\mu}] \leq \frac{\mathbb{E}[e^{tX}]}{e^{t(1+\delta)\mu}}$$

By the above and independence of X_i's, we have

$$\mathbb{E}[e^{tX}] = \prod_{i=1}^{n} \mathbb{E}[e^{tX_i}] = \prod_{i=1}^{n} (p_i \cdot e^t + (1-p_i)\cdot 1)$$

Since $p_i \cdot e^t + (1-p_i)\cdot 1 = 1 + p_i \cdot (e^t - 1) \leq e^{p_i(e^t-1)}$ as $e^x \geq 1 + x$ for all $x \geq 0$, we have

$$\frac{\mathbb{E}[e^{tX}]}{e^{t(1+\delta)\mu}} \leq \frac{\prod_{i=1}^{n} e^{p_i(e^t-1)}}{e^{t(1+\delta)\mu}} = \left(\frac{e^{e^t-1}}{e^{t(1+\delta)}}\right)^{\mu} \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

where $t = \ln(1+\delta)$. The main inequality follows and the fact that $\dfrac{e^{\delta}}{(1+\delta)^{1+\delta}} \leq e^{-\delta^2/3}$ for all $0 < \delta < 1$.

2.4 Hoeffding's Inequality

Lemma (Hoeffding)

If Z is a random variable such that $Z \in [a, b]$, then

$$\mathbb{E}[e^{t(Z-\mathbb{E}[Z])}] \leq e^{t^2(b-a)^2/8}$$

Theorem (Hoeffding's Inequality)

Let X_i be independent random variables, taking values in $[a_i, b_i]$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X]$. Then, for any $l > 0$

$$P[|X - \mu| \geq l] \leq 2 \cdot \exp\left(-\frac{2l^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

Proof. Similar to Chernoff's inequality, but use Hoeffding's lemma.

Chapter 3

Balls and Bins

3.1 Introduction

Theorem (Union Bound)

Can be generalized to any level of intersections.

$$P[A \cup B \cup C] \leq P[A] + P[B] + P[C]$$

$$P[A \cup B \cup C] \geq P[A] + P[B] + P[C] - P[A \cap B] - P[A \cap C] - P[B \cap C]$$

Definition: Conditional Probability

The conditional probability of E_1 given E_2 is

$$P[E_1|E_2] := \frac{P[E_1 \cap E_2]}{P[E_2]}$$

Proposition

If $E_1, \ldots, E_k$ partition out sample space, then for any event E

$$P[E] = \sum_{i=1}^{k} \underbrace{P[E|E_i] \cdot P[E_i]}_{P[E \cap E_i]}$$

Theorem (Simple Bayes' Rule)

$$P[E_1|E_2] = \frac{P[E_2|E_1] \cdot P[E_1]}{P[E_2]}$$

Theorem (Bayes' Rule)

If $E_1, \ldots, E_k$ partition our sample space, then for event E

$$P[E_i|E] = \frac{P[E \cap E_i]}{P[E]} = \frac{P[E|E_i] \cdot P[E_i]}{\sum_{j=1}^{k} P[E|E_j] \cdot P[E_j]}$$

Balls and Bins

Given m balls and n bins to throw each ball into a uniformly random bin independently.

- What is the expected number of balls in a bin?
- What is the expected number of empty bins?
- What is typically the maximum number of balls in any bin (maximum load)?
- What is the expected number of bins with k balls in them?
- For what values of m do we expect to have no empty bins? (coupon collector)

Strategy of Randomized Algorithms: Devise the randomized algorithm with good expected runtime and prove concentration of measure around expectation.

3.1.1 Expected Number of Balls in a Bin

Label the m balls $1, \ldots, m$ and the n bins $1, \ldots, n$. Let B_{ij} be the indicator variables that ball i was thrown into bin j.

$$\begin{aligned}
\mathbb{E}[\#\text{ balls in bin } j] &= \mathbb{E}\left[\sum_{i=1}^{m} B_{ij}\right] \\
&= \sum_{i=1}^{m} \mathbb{E}[B_{ij}] \\
&= \sum_{i=1}^{m} 1 \cdot P[\text{ball } i \text{ in bin } j] + 0 \cdot (1 - P[\text{ball } i \text{ in bin } j]) \\
&= \sum_{i=1}^{m} P[\text{ball } i \text{ in bin } j] \\
&= \sum_{i=1}^{m} \frac{1}{n} = \frac{m}{n}
\end{aligned}$$

When $m = n$, expect one ball per bin.

3.1.2 Expected Number of Empty Bins

Let N_i be the indicator variable that bin i is empty after m throws.

$$\begin{aligned}
\mathbb{E}[\#\text{ empty bins}] &= \mathbb{E}\left[\sum_{i=1}^{n} N_i\right] \\
&= \sum_{i=1}^{n} \mathbb{E}[N_i] \\
&= \sum_{i=1}^{n} P[\text{bin } i \text{ is empty}] \\
&= \sum_{i=1}^{n} \left(1 - \frac{1}{n}\right)^m \\
&= n\left(1 - \frac{1}{n}\right)^m \\
&\approx ne^{-m/n} && \left(1 - \frac{1}{n}\right)^n \approx e^{-1}
\end{aligned}$$

When $m = n$, expected fraction of empty bins is $\frac{1}{e}$.

When $m = n$, the first expectation was one ball per bin while the second expectation was $\frac{1}{e}$ fraction of empty bins. This is where concentration of probability measure tries to address. The second expectation is actually concentrated around the mean so more typical to happen.

3.1.3 Maximum Load in a Bin

Birthday Paradox

For what value of m do we expect to see two balls in one bin?

The probability that there are no collisions after we have thrown m balls is:

$$1\left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right)\cdots\left(1 - \frac{m-1}{n}\right) \leq e^{-\frac{1}{n}} \cdots e^{-\frac{m-1}{n}} \approx e^{-m^2 2n}$$

This is $\leq 1/2$ when $m = \sqrt{2n \ln 2}$. For $n = 365$, this is $m \approx 22.4$ for the the probability that two people (balls) have birthday on the same date (bins) to become $\geq 1/2$.

Thus, we expect to see collision when $m = \Theta(\sqrt{n})$.

3.1.4 Maximum Load in a Bin when $m = n$

What is the probability that a particular bin has $\geq k$ balls in it?

$$\begin{aligned} P[\text{bin } x \text{ has} \geq k \text{ balls}] &\leq \sum_{\substack{S \subseteq [n] \\ |S|=k}} \prod_{i \in S} P[\text{ball } i \text{ in bin } x] \\ &= \sum_{\substack{S \subseteq [n] \\ |S|=k}} \prod_{i \in S} \frac{1}{n} \\ &= \binom{n}{k} \cdot \frac{1}{n^k} \\ &\leq \left(\frac{ne}{k}\right)^k \cdot \frac{1}{n^k} \\ &= \frac{e^k}{k^k} \end{aligned}$$

By union bound,

$$P[\text{some bin has} \geq k \text{ balls}] \leq \sum_{i=1}^{n} P[\text{bin } i \text{ has} \geq k \text{ balls}] \leq \frac{ne^k}{k^k} = e^{\ln n + k - k \ln k}$$

Therefore, the probability of maximum load at most k is

$$P[\text{max load} \leq k] = 1 - P[\text{some bin has} > k \text{ balls}] \geq 1 - e^{\ln n + k - k \ln k}$$

The above probability will be large ($>> \frac{1}{2}$) when $k \ln k > \ln n$ such as setting $k = \frac{3 \ln n}{\ln \ln n}$.

With high probability, the maximum load is $O\left(\frac{\ln n}{\ln \ln n}\right)$.

3.2 Coupon Collector and Power of Two Choices

3.2.1 Coupon Collector

Coupon Collector: For what value of m do we expect to have no empty bins?
Suppose each bin is a different coupon. We can buy one coupon at random. What is the number of coupons that we need to buy to collect all of them?

Let X_i be the number of balls thrown to get from i empty bins to $i-1$ empty bins. Let X be the number of balls thrown until we have no empty bins. Thus, $X = \sum_{i=1}^{n} X_i$.

X_i is a random variable that follows a geometric distribution with parameter $p = \frac{i}{n}$ (Geometric distribution is the number of trials until first success, where success probability is p).

$$P[X_i = k] = p(1-p)^{k-1}$$

So the expected value of X_i where X_i takes values over $\mathbb{N}$ is

$$\mathbb{E}[X_i] = \sum_{k \geq 1} k \cdot P[X_i = k] = p_i \sum_{k \geq 1} (1-p)^{k-1} = p_i \cdot \frac{1}{(1-(1-p))^2} = \frac{1}{p} = \frac{n}{i}$$

by the derivative of the closed form of the series. Thus,

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \frac{n}{i} = n \sum_{i=1}^{n} \frac{1}{i} \approx n \ln n$$

The $n \ln n$ bound shows up in cover time of random walks in complete graphs and the number of edges needed in graph sparsification.

3.2.2 Power of Two Choices

We now know that when n balls are thrown into n bins, the maximum load is $\Theta\left(\frac{\ln n}{\ln \ln n}\right)$ with constant probability.

Consider if when throwing a ball in a bin, before we thrown the ball we choose two bins uniformly at random and put the ball in the bin with few balls.

This simplification reduces maximum load to $O(\ln \ln n)$.

Idea: Let the height of a bin be the number of balls in it. The process above tells us that to get one bin height $h+1$, we must have at least two bins with height h. We can bound the number of bins with height $\geq h$, as this will tell us how likely it is to get a bin with height $h+1$.

If N_h is the number of bins with height $\geq h$, then we have

$$P[\text{at least one bin of height } h+1] \leq \binom{N_h}{2} \leq \left(\frac{N_h}{n}\right)^2$$

To bound N_h, say we only have $\frac{n}{4}$ bins with 4 items, i.e. height 4. Then, the probability of selecting 2 such bins is $\leq \frac{1}{16}$. So, we should expect only $\frac{n}{16}$ bins with height 5. Analogously, we should expect only $\frac{n}{16^2} = \frac{n}{256} = \frac{n}{2^{2^3}}$ bins with height 6.

Repeating this, we should expect only $n/2^{2^{h-3}}$ bins with height h. So we expect $\log \log n$ maximum height after throwing n balls.

Chapter 4

Hashing

4.1 Hash Functions

4.1.1 Computational Model

Definition: Word RAM Model

In the word RAM model:

- All elements are integers that fit in a machine word of w bits.
- Basic operations (comparison, arithmetic, bitwise) on such words take $\Theta(1)$ time.
- Access any position in the array in $\Theta(1)$ time.

This model is relevant for problems of good enough size (so asymptotic analysis can work), but not super huge that words do not fit in a machine word.

4.1.2 Hash Functions

Problem

Store n elements (keys) from the set $U = \{0, 1, \ldots, m-1\}$ where $2^w > m >> n$, in a data structure that supports insertion, deletions, search as efficiently in runtime and memory.

Definition: Hash Function

A function $h : U \to [0, n-1]$, where $|U| = m >> n$.

Definition: Hash Table

A data structure that consists of

- a table T with n cells $[0, n-1]$, each cell storing a word
- a hash function $h : U \to [0, n-1]$

Definition: Collision

We say that a collision happens for hash function h with inputs $x, y \in U$ if $x \neq y$ and $h(x) = h(y)$.

By pigeonhole principle, it is impossible to achieve no collisions without knowing keys in advance. We want the number of collisions to be small *with high probability*.

Solution: Construct a family of hash function $\mathcal{H}$ such that the number of collisions is small with high probability, when we pick hash function uniformly at random from $\mathcal{H}$.

$$P_{h \in_R \mathcal{H}}[h(x) = h(y)] \leq \frac{1}{\text{poly}(n)}, \ \forall x \neq y \in U$$

Assumptions: keys are independent from hash function that are chosen and we do not know the keys in advance. This can still have collisions.

4.1.3 Random Hash Functions

From all functions $h : U \to [0, n-1]$, pick one uniformly at random. This is the same as balls and bins.

If we have to store n keys:

- Expected number of keys in a location is 1.
- Maximum number of collisions (maximum load) in one location is $O\left(\frac{\log n}{\log \log n}\right)$ keys.

To handle collisions, we can store all keys hashed into location i by a linked list, known as chain hashing.

We can also pick two random hash functions and use power of two choices. The collision bound becomes $O(\log \log n)$.

Question: How much time and space does it take to compute random hash functions?

Storing entire functions $h : U \to [0, n-1]$ requires $O(m \log n)$ bits and if we only stored the elements we saw, we would require $O(n)$ time to evaluate $h(x)$, since we need to decide if we had already computed it. Thus, for random function operations (search, insert, delete) take $O(n)$ time at best.

4.2 k-Wise Independence

We want something is random-like, but easy to compute/represent. Ideally, $O(1)$ time to compute. h is described in $c \log m$ bits. This is at most the number of bit strings of length $c \log m$: $2^{c \log m} = m^c$ so poly(m) functions, as each function takes at most $O(\log m)$ bits to describe. These are succinct functions which have random-like properties.

Definition: Full Independence

A set of random variables $X_1, \ldots, X_n$ are said to be fully independent if for any subset $J \subseteq [n]$, they satisfy

$$P\left[\bigcap_{i \in J} X_i = a_i\right] = \prod_{i \in J} P[X_i = a_i]$$

Definition: k-Wise Independence

A set of random variables $X_1, \ldots, X_n$ are said to be k-wise independent if for any set $J \subseteq [n]$ such that $|J| \leq k$, they satisfy

$$P\left[\bigcap_{i \in J} X_i = a_i\right] = \prod_{i \in J} P[X_i = a_i]$$

When $k = 2$, k-wise independence is called pairwise independence.

Example (XOR Pairwise Independence)

Given t uniformly random bits $Y_1, \ldots, Y_t$, we can generate $2^t - 1$ pairwise independent random variables as follows:

$$X_S := \bigoplus_{i \in S} Y_i, \ S \subseteq [t] \setminus \emptyset$$

Example (Pairwise Independence in $\mathbb{F}_p$)

Let p be a prime number. Given two uniformly random variables $Y_1, Y_2 \sim [0, \ldots, p-1]$, generate p pairwise independent random variables as follows:

$$X_i := Y_1 + i \cdot Y_2 \mod p, \ i \in [0, p-1]$$

4.3 Universal Hash Functions

Definition: Universal Hash Functions

Let U be a universe with $|U| \geq n$. A family of hash functions $\mathcal{H} = \{h : U \to [0, n-1]\}$ is k-universal if, for any distinct elements $u_1, \ldots, u_k \in U$, we have

$$P_{h \in_R \mathcal{H}}[h(u_1) = h(u_2) = \cdots = h(u_k)] \leq \frac{1}{n^{k-1}}$$

Definition: Strongly Universal Hash Functions

$\mathcal{H} = \{h : U \to [0, n-1]\}$ is strongly k-universal if, for any distinct elements $u_1, \ldots, u_k \in U$ and for any values $y_1, \ldots, y_k \in [0, n-1]$, we have

$$P_{h \in_R \mathcal{H}}[h(u_1) = y_1, \ldots, h(u_k) = y_k] \leq \frac{1}{n^k}$$

Theorem

Family $\mathcal{H}$ is strongly k-universal if the random variables $h(0), \ldots, h(|U|-1)$ are k-wise independent.

Proposition

Let p be a prime number and $U = [0, n-1]$.

$$\mathcal{H} = \{h_{a,b}(x) := ax + b \mod p | a, b \in [0, p-1]\}$$

is strongly 2-universal.

Proposition

Let $U = [0, p^k - 1] \equiv [0, p-1]^k \setminus \{(0, \ldots, 0)\}$ and $a = (a_0, \ldots, a_{k-1})$, then

$$\mathcal{H} = \{h_{a,b}(x) := a \cdot x + b \mod p | a \in U, b \in [0, p-1]\}$$

is strongly 2-universal.

Proposition

If hash table size is not prime, then

$$\mathcal{H} = \{h_{a,b}(x) := (a \cdot x + b \mod p) \mod n | a, b \in [0, p-1]$$

is 2-universal (not strongly 2-universal).

We can construct k-universal families of hash functions by instead of constructing degree 1 polynomials, we can construct polynomials of degree $k-1$. Random $k-1$ degree polynomial

is k-wise independent.

Lemma (Maximum Number of Collisions)

The expected number of collisions when inserting k elements in a table of size n using a 2-universal hash family is

$$\leq \frac{k^2}{2n}$$

Proof. Let $X_{ij} = 1$ if $h(i) = h(j)$ and $X_{ij} = 0$ otherwise. The number of collisions is $X = \sum_{i<j} X_{ij}$.

$$\begin{aligned}
\mathbb{E}[X] &= \mathbb{E}[\sum_{i<j} X_{ij}] \\
&= \sum_{i<j} \mathbb{E}[X_{ij}] \\
&\leq \frac{\binom{k}{2}}{n} \\
&\leq \frac{k^2}{2n}
\end{aligned}$$

Lemma (Maximum Load of Entry of Hash Table)

With probability $\geq \frac{1}{2}$, the maximum load when inserting k elements in a table of size n using a 2-universal hash family is

$$\leq \sqrt{\frac{2k^2}{n}}$$

When $k \approx n$, we expect $\sqrt{2n}$.

Proof. Let C be the maximum load entry of the hash table. The number of collisions $X \geq \binom{C}{2} \sim \frac{C^2}{2}$. By Markov's inequality,

$$P\left[X \geq \frac{k^2}{n}\right] \leq \frac{1}{2} \implies P\left(X < \frac{k^2}{n}\right) \geq \frac{1}{2}$$

Then,

$$P\left[\frac{C^2}{2} < \frac{k^2}{n}\right] \geq P\left[X < \frac{k^2}{n}\right] \geq \frac{1}{2}$$

Corollary

If $h \in \mathcal{H}$ is a random hash function from a 2-universal family of hash functions, then for any set $S \subseteq U$ of size $k \leq \sqrt{n}$, the probability of h being perfect for S is at least $\frac{1}{2}$.

4.4 Perfect Hashing

Suppose we are given a set of keys S of size n in advance. Our goal is to construct a hash table with no collisions and $O(n)$ memory.

We can do this with a 2-universal hash family, but we would need two layers of hash tables.

For the first layer, use a hash table with n entries. Let $\mathcal{H}$ be a 2-universal hash family of maps from $U \to [0, n-1]$. If we pick a random hash function $h \in \mathcal{H}$, with probability $\geq \frac{1}{2}$, the maximum load of any entry is $\sqrt{2n}$.

Since we have the keys, we can check if the maximum load is $\leq \sqrt{2n}$, otherwise we can just pick another random hash function and try again. In constantly many tries, with high probability, we will find a hash function h such that maximum load is $\leq \sqrt{2n}$.

Now that we have a first hash table with maximum load $\sqrt{2n}$, we can construct a second layer of hash tables. For each entry $i \in [0, n-1]$, we will construct a hash table with n_i entries, where n_i is the number of keys that hash to i.

Since n_i is the load of entry i, we have $\frac{n_i(n_i-1)}{2}$ collisions in entry i. By corollary, we also have that the total number of collisions will be $\leq n$, and thus we have

$$\sum_{i=0}^{n-1} n_i^2 = O(n)$$

Let $\mathcal{H}_i$ be a 2-universal hash family of maps from $U \to [0, n_i^2 - 1]$. For each entry $i \in [0, n-1]$, we can pick a random $h_i \in \mathcal{H}_i$ with probability $\geq \frac{1}{2}$, the maximum load of any entry is $\leq \sqrt{2}$ (no collisions). Since we have the keys, we can check the maximum load, and if it is not $\leq \sqrt{2}$, then try again. In constantly many tries, with high probability, we will find a h_i which has no collisions.

Thus, we have constructed a hash scheme with no collisions and total memory

$$n + \sum_{i=1}^{n} n_i^2 = O(n)$$

Chapter 5

Graph Sparsification

Often times, graph algorithms for graphs $G = (V, E)$ have runtimes that depend on the number of edges $|E|$.

For example, the runtime of Dijkstra's algorithm is $O(|E| + |V| \log |V|)$ and the runtime of the Ford-Fulkerson algorithm is $O(|E| \cdot f^*)$, where f^* is the value of the maximum flow.

Definition: Dense

A graph is dense if

$$|E| = \omega(|V|^{1+\gamma})$$

for $\gamma \in (0, 1)$.

Definition: Sparse

A graph is sparse if

$$|E| = O(|V| \cdot \operatorname{poly} \log |V|) = \tilde{O}(|V|)$$

We would like to sparsify a graph, while preserving properties of the graph to reduce to the runtime in practical purposes.

When sparsifying a graph, we may lose some information about the graph, so we will settle with approximately preserving properties.

5.1 Minimum Cut

Let $G = (V, E, w)$ be an undirected graph with nonnegative edge weights $w_e \geq 0$. When working with an unweighted graph, then $w_e = 1$ for all $e \in E$. Denote $n = |V|$ and $m = |E|$.

Definition: Graph Cut

A cut $(S, \overline{S})$ is a partition of the vertices V into two sets S and $\overline{S}$. That is $V = S \cup \overline{S}$ and $S \cap \overline{S} = \emptyset$.
The value of a cut is the sum of the weights of the edges that cross the cut, i.e. edges with one end in S and the other end in $\overline{S}$. Denote $E(S, \overline{S})$ as the edges that cross the cut, then the value of the cut is

$$w(S, \overline{S}) = \sum_{e \in E(S, \overline{S})} w_e$$

Definition: Minimum Cut

A cut of minimum value.

Definition: Edge Contraction

Let $e = uv$ be an edge in $G = (V, E, w)$. The contraction of e is a new graph $H = (V \cup \{z\} \setminus \{u, v\}, F, w')$, where u, v is replaced by z, any edge $ux \neq e$ is replaced by zx, and any edge $vx \neq e$ is replaced by zx. Also, $w'(z, x) = w(u, x)$.

The contraction of an edge e is a graph with one less vertex and it may not necessarily be a simple graph. In the case of weighted graphs, we can combine parallel edges into one edge with the sum of the weights.

Lemma

Let $e = uv$ be an edge in $G = (V, E, w)$ and let H be obtained by contracting an edge $e \in E$.
The value of the minimum cut in H is at least the value of the minimum cut in G.

Algorithm 2 Randomized Minimum Cut

1: **Input**: Undirected, unweighted graph $G = (V, E)$
2: **Output**: A minimum cut $(S, \overline{S})$ of G
3: **while** $n > 2$ **do**
4: Pick $e = uv$ uniformly at random from E
5: Contract e
6: **if** $n = 2$ **then return** $(S, \overline{S})$ induced by the two vertices in V

Theorem (Karger)

The randomized minimum cut algorithm outputs a minimum cut with probability at least $\frac{2}{n(n-1)}$.

Proof. Let $(S, \overline{S})$ be a minimum cut of G and let $c = w(S, \overline{S})$. If we never contract an edge from $E(S, \overline{S})$, then the algorithm succeeds, as we will output $w(S, \overline{S})$.

Computing the probability that an edge from $E(S, \overline{S})$ is contracted in one iteration of the algorithm. Each vertex is a cut, so each vertex has degree at least c. Hence, we know that at least $(n-i+1) \cdot \frac{c}{2}$ edges remain.

The probability that we contract an edge from $E(S, \overline{S})$ is

$$\frac{w(S, \overline{S})}{\#\text{ edges}} \leq \frac{c}{(n-i+1) \cdot \frac{c}{2}} = \frac{2}{n-i+1}$$

Hence, the probability that we never contract an edge from $E(S, \overline{S})$ is at least

$$\prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) = \prod_{i=3}^{n} \left(1 - \frac{2}{i}\right) = \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}}$$

To improve the above probability, we can run the algorithm multiple times and output the minimum cut over all iterations. If we repeat t times, then the failure probability is at most

$$\left(1 - \frac{2}{n(n-1)}\right)^t$$

If we set $t = 2n(n-1)$, then we get a failure probability of at most

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

Runtime: Each iteration of the algorithm takes $O(m)$ time, and we run the algorithm $O(n^2)$ times, so the total runtime is $O(n^2 m)$.

Corollary (Karger)

There are at most $\binom{n}{2}$ minimum cuts in a graph.

Proof. Each minimum cut is preserved with probability at least $\frac{1}{\binom{n}{2}}$. Since the events that each minimum cut is preserved are disjoint (and sum of probability is ≤ 1), there can be at most $\binom{n}{2}$ minimum cuts.

Lemma

If c is the minimum cut in G, then there are at most $n^{2\alpha}$ cuts of value $k \leq \alpha c$ in G.

5.2 Graph Sparsification Algorithm

We need to set p to be the corrected expected value for both the number of edges in H and the value of each cut in H. After that, we need to prove concentration bounds for values of all cuts in H, simultaneously.

Algorithm 3 Randomized Sparsification

1: **Input**: Undirected, unweighted graph $G = (V, E)$ and parameter $\varepsilon > 0$
2: **Output**: A sparse weighted graph $H = (V, F, w)$ such that for every cut $(S, \overline{S})$, we have

$$(1 - \varepsilon) \cdot w(S, \overline{S}) \leq w_H(S, \overline{S}) \leq (1 + \varepsilon) \cdot w(S, \overline{S})$$

3: $p \in (0, 1)$ be a parameter
4: For each edge $e \in E$, include e in F with probability p, and if included, set $w_H(e) = \frac{1}{p}$

We can do this using Chernoff-Hoeffding bounds, then show that there are not too many small cuts in G, and thus, the probability that we have a bad cut in H is small. We can then use union bound to prove all cuts are concentrated.

Theorem (Karger)

Let c be the value of the minimum cut in G. Set

$$p = \frac{15 \log n}{\varepsilon^2 c}$$

With probability $\geq 1 - \frac{4}{n}$, the above algorithm outputs a graph $H = (V, F, w_H)$ with $|F| = O(p \cdot |E|)$ such that for every cut $(S, \overline{S})$ in G:

$$(1 - \varepsilon) \cdot w(S, \overline{S}) \leq w_H(S, \overline{S}) \leq (1 + \varepsilon) \cdot w(S, \overline{S})$$

Proof. Let $H = (V, F, w_H)$ be the graph output by the algorithm. Take a cut $(S, \overline{S})$. Denote $k = w(S, \overline{S})$. Let X_e be the indicator random variable for the event that $e \in F$.

Then, $w_H(S, \overline{S}) = \sum_{e \in E(S, \overline{S})} \frac{X_e}{p}$ and $|F| = \sum_{e \in E} X_e$.

Hence, the expected values are

- $\mathbb{E}[|F|] = \sum_{e \in E} \mathbb{E}[X_e] = p \cdot m$
- $\mathbb{E}[w_H(S, \overline{S})] = \sum_{e \in E(S, \overline{S})} \mathbb{E}[w_H(e)] = \sum_{e \in E(S, \overline{S})} \mathbb{E}[X_e / p] = k$

Now compute concentration bounds for $|F|$ and $w_H(S, \overline{S})$:

- For $|F|$,
$$P[|F| \geq (1 + \varepsilon) \cdot p \cdot m] \leq e^{-\varepsilon^2 \cdot \mathbb{E}[|F|]/3} = e^{-\varepsilon^2 \cdot p \cdot m/3} \leq \frac{1}{n^2}$$
- For $w_H(S, \overline{S})$, note that $p \cdot w_H(S, \overline{S})$ is a sum of independent random variables with values in $0, 1$. We can use Chernoff bounds to get
$$P[\left| w_H(S, \overline{S}) - k \right| \geq \varepsilon \cdot k] \leq 2 \cdot e^{-\varepsilon^2 \cdot kp/3} = 2n^{-5k/c}$$

Note that $k \geq c$, as c is the value of the minimum cut of G. The above is the probability that a single cut deviates from its expectation. We want all cuts. Using union bound,

$$\begin{aligned}
P[\text{a cut deviates from expectation}] &\leq \sum_{S \subseteq V} P\left[\left|w_H(S, \overline{S}) - k\right| \geq \varepsilon \cdot k\right] \\
&\leq \sum_{\alpha=1,2,4,8,\ldots} \sum_{\substack{S \subseteq V \\ \alpha c \leq \left|w_G(S,\overline{S})\right| \leq 2 \cdot \alpha c}} P\left[\left|w_H(S, \overline{S}) - k\right| \geq \varepsilon \cdot k\right] \\
&\leq \sum_{\alpha=1,2,4,8,\ldots} n^{4\alpha} \cdot P\left[\left|w_H(S, \overline{S}) - k\right| \geq \varepsilon \cdot k | \alpha c \leq k \leq 2\alpha c\right] \\
&\leq \sum_{\alpha=1,2,4,8,\ldots} n^{4\alpha} \cdot 2 \cdot n^{-5\alpha} \\
&= \sum_{\alpha=1,2,4,8,\ldots} n^{-\alpha} \leq \frac{4}{n}
\end{aligned}$$

We have assumed that the graph has a large min-cut value: $c = \Omega(\log n)$. This assumption is so that $p < 1$. To remove the assumption, we can use non-uniform sampling of edges. If we choose this non-uniform sampling carefully, we can get a sparse graph which approximates all cuts with high probability.

Definition: Strong Connectivity

A k-strong component is a maximal induced subgraph that is k-edge-connected.

Chapter 6

Algebraic Techniques: Fingerprinting, Polynomial Identity Testing, and Parallel Matching Algorithms

6.1 Verifying String Equality

Suppose Alice and Bob each maintain the same large database of information. They want to check if their databases are consistent.

However, transmission of all data is expensive and sending the entire database is not feasible. Say Alice's database is given by bits $(a_1, \ldots, a_n)$ and Bob's database is given by bits $(b_1, \ldots, b_n)$. A deterministic consistency check requires both of them to communicate n bits.

Problem

Given strings $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$, check if they are equal.

Fingerprinting: Let $a = \sum_{i=1}^{n} a_i 2^{i-1}$ and $b = \sum_{i=1}^{n} b_i 2^{i-1}$. Let $F_p(x) = x \mod p$ be a fingerprinting function for a prime p.

Protocol:

1. Alice picks a random prime p and sends $(p, F_p(a))$ to Bob.
2. Bob checks whether $F_p(a) \equiv F_p(b) \mod p$ and sends

$$\begin{cases} 1 & \text{if values are equal} \\ 0 & \text{otherwise} \end{cases}$$

The total bits communicated is $O(\log p)$ bits, dominated by Alice's message. If the two strings are equal, then the protocol is always right.

If $(a_1, \ldots, a_n) \neq (b_1, \ldots, b_n)$, then $a \neq b$. If a number M is in $\{-2^n, \ldots, 2^n\}$, then the number of distinct primes $p|M$ is $< n$.

- Each prime divisor of M is ≥ 2, so if M has t distinct prime divisors, then $|M| > 2^t$.
- $|M| \leq 2^n \implies t \leq n$

Thus, $F_p(a) \equiv F_p(b)$ if and only if $p|a-b$. The protocol fails for at most n choices of p.

Theorem (Prime Number Theorem)

There are $\frac{m}{\log m}$ primes among the first m positive integers.

Choosing p among the first $tn\log(tn)$ integers, we have

$$P[F_p(a) \equiv F_p(b)] \leq \frac{n}{tn\log(tn)/\log(tn\log(tn))} = \tilde{O}\left(\frac{1}{t}\right)$$

The number of bits sent is $\tilde{O}(\log t + \log n)$. Choosing $t = n$ solves it.

6.2 Polynomial Identity Testing

Problem

Input: two polynomials $P(x), Q(x)$.
Output: are they equal?

Two polynomials are equal if and only if all their coefficients are equal.

We cannot just compare coefficients since polynomials may sometimes be given implicitly. For example, we may want to test if $P_1(x) \cdot P_2(x) = P_3(x)$. If P_1, P_2 have degree $\leq n$, then $\deg(P_3) \leq 2n$, otherwise the problem is trivial.

Multiplication of two polynomials of degree n take $O(n \log n)$ by arithmetic operation of fast Fourier transform. Polynomial evaluation also takes $O(n)$ arithmetic operations, if we are given the coefficients.

Lemma (Roots of Univariate Polynomials)

Let $\mathbb{F}$ be a field and $P(x) \in \mathbb{F}[x]$ be a nonzero univariate polynomial of degree d. Then, $P(x)$ has at most d roots in $\mathbb{F}$.

$P_1, P_2 = P_3 \iff P_1P_2 - P_3 = 0$, so there are at most $2n$ roots by lemma. Take $S \subseteq \mathbb{F}$ of size $4n$. Let $a \in S$ be random. Compute $Q(a) = P_3(a) - P_1(a)P_2(a)$.

$$P_{a \in S}[Q(a) = 0] \leq \frac{\deg(Q)}{|S|} \leq \frac{2n}{4n} = \frac{1}{2}$$

Lemma (Ore-Schwartz-Zippel-de Millo-Lipton)

Let $\mathbb{F}$ be a field and $P(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ be a nonzero polynomial of degree $\leq d$. Then, for any set $S \subseteq \mathbb{F}$, we have

$$Pr[P(a_1, \ldots, a_n) = 0 | a_i \in S] \leq \frac{d}{|S|}$$

6.3 Bipartite Matching

Problem

Input: bipartite graph $G = (L, R, E)$ with $|L| = |R| = n$.
Output: does G have a perfect matching?

A perfect matching corresponds to a permutation S_n.

Let $X \in \mathbb{F}^{n \times n}$ be such that

$$X_{i,j} = \begin{cases} y_{i,j} & \text{if there is an edge between } (i,j) \in L \times R \\ 0 & \text{otherwise} \end{cases}$$

The determinant is

$$\det(X) = \sum_{\sigma \in S_n} (-1)^\sigma \prod_{i=1}^{n} X_{i,\sigma(i)}$$

G has a perfect matching if and only if $\det(X)$ is a nonzero polynomial. Therefore, testing if G has a perfect matching is a special case of Polynomial Identity Testing.

Algorithm: Evaluate $\det(X)$ at a random value for the variables $y_{i,j}$.

6.4 General Matching

Problem

Input: undirected graph $G = (V, E)$ where $|V| = 2n$.
Output: does G have a perfect matching?

Definition: Tutte Matrix

T_G is defined by the $2n \times 2n$ matrix: let F be an arbitrary orientation of the edges in E. Then,

$$[T_G]_{i,j} = \begin{cases} x_{i,j} & \text{if } (i,j) \in F \\ -x_{i,j} & \text{if } (j,i) \in F \\ 0 & \text{otherwise} \end{cases}$$

Theorem (Tutte 1947)

G has a perfect matching if and only if $\det(T_G) \neq 0$.

6.5 Parallel Algorithms and Isolation Lemma

Often times in parallel computation, when solving a problem with many possible solutions, we want all processors working towards the same solution. We need to isolate a specific solution without knowing any element of the solution space.

The solution is to implicitly choose a random order on the feasible solutions and require processors to find solution of lowest rank in this order. We can use this to compute the minimum weight perfect matching.

Lemma (Isolation Lemma)

Given a set system over $[n] := \{1, \ldots, n\}$, if we assign a random weight function $w : [n] \to [2n]$, then the probability that there is a unique solution minimum weight set is at least $\frac{1}{2}$.

This could be quite counter-intuitive. A set system can have $\Omega(2^n)$ sets. On average, there are $\Omega\left(\frac{2^n}{2n^2}\right)$ sets of of a given weight, as max weight is $\leq 2n^2$. Isolation lemma tells us that with high probability, there is only one set of minimum weight.

Proof. Let $\mathcal{S}$ be our set system and $v \in [n]$. Let $\mathcal{S}_v$ be a family of sets from $\mathcal{S}$ which contain v, and $\mathcal{N}_v$ be the family of sets from $\mathcal{S}$ which do not contain v. Let

$$\alpha_v := \min_{A \in \mathcal{N}_v} w(A) - \min_{B \in \mathcal{S}_v} w(B \setminus \{v\})$$

- $\alpha_v < w(v)$ implies v does not belong to any minimum weight set.
- $\alpha_v > w(v)$ implies v belongs to every minimum weight set.
- $\alpha_v = w(v)$ implies v is ambiguous.

α_v is independent of $w(v)$ and $w(v)$ is chosen uniformly at random from $[2n]$.

$$P[v \text{ ambiguous}] \leq \frac{1}{2n} \implies_{\text{union bound}} P[\exists \text{ ambiguous element}] \leq \frac{1}{2}$$

If two different sets A, B have minimum weight, then any element in $A \Delta B$ must be ambiguous. The probability this happens $\leq \frac{1}{2}$.

Chapter 7

Sublinear Time Algorithms

We cannot answer for all, there exists, or exactly type statements. We can answer for most, averages, or approximate type statements.

If N is the input size, then a sublinear time algorithm may not read in all of N, so we want an algorithm in $o(N)$.

7.1 Approximate Diameter

Definition: α-Multiplicative Approximation

$$\frac{1}{\alpha} X_{\max} \leq X \leq \alpha X_{\max}$$

Approximate Diameter of a Point Set

Input: m points and a distance matrix D such that

- D_{ij} is the distance from i to j.
- D is symmetric and satisfies triangle inequality; $D_{ij} \leq D_{ik} + D_{kj}$.

The input is given in adjacency matrix representation. Input size is $N = m^2$.
The diameter D_{ab} is the maximal distance between indices a and b.
Output: Indices k, l such that

$$D_{kl} \geq \frac{D_{ab}}{2}$$

This will be a 2-approximation algorithm.

Algorithm: Pick k arbitrary, pick l to maximize D_{kl}, output indices k, l.

Correctness:

$$D_{ab} \leq D_{ak} + D_{kb} \leq D_{kl} + D_{kl} = 2D_{kl}$$

Runtime: $O(m) = O(N^{1/2})$

7.1.1 Lower Bound

Let D be the distance matrix where $D_{i,i} = 0$ for all $i \in [m]$ and $D_{i,j} = 1$ otherwise. Let D' be the same matrix as D except for one pair (a, b), we make

$$D'_{ab} = D'_{ba} = 2 - \delta$$

Check that D' satisfies properties of a distance matrix. We can prove that it would take $\Omega(N)$ time (number of queries) to decide if diameter is 1 or $2 - \delta$.

7.2 Connected Components

Approximate Number of Connected Components

Input: graph $G = (V, E)$ in adjacency list, precision parameter $\varepsilon > 0$.
Output: if C is number of connected components of G, output with probability $\geq \frac{3}{4}$, C' such that

$$|C' - C| \leq \varepsilon n$$

Lemma (Number of Connected Components)

Let $G = (V, E)$ be a graph. For vertex $v \in V$, let n_v be the number of vertices in connected component of v. Let C be the number of connected components of G. Then,

$$C = \sum_{v \in V} \frac{1}{n_v}$$

Take a sample of vertices from G, compute n_v and output the normalization. The problem is that computing n_v make take linear time. If n_v is large, then $\frac{1}{n_v}$ is small, so we can drop it.

Lemma (Estimating Number of Connected Components)

Let $n'_v = \min\{n_v, 2/\varepsilon\}$, then

$$\left| \sum_{v \in V} \frac{1}{n_v} - \sum_{v \in V} \frac{1}{n'_v} \right| \leq \frac{\varepsilon n}{2}$$

Proof. Let $U = \{v \in V | n_v > 2/\varepsilon\}$.

$$\begin{aligned}\left|\sum_{v\in U}\left(\frac{\varepsilon}{2} - \frac{1}{n_v}\right)\right| &= \sum_{v\in U}\left(\frac{\varepsilon}{2} - \frac{1}{n_v}\right) \\ &\leq \frac{|U|\,\varepsilon}{2} \\ &\leq \frac{n\varepsilon}{2}\end{aligned}$$

Sample vertex v, run BFS, and stop if we see $2/\varepsilon$ vertices.

Algorithm: Choose $s = \Theta(1/\varepsilon^2)$ vertices $v_1, \ldots, v_s$ uniformly at random. Compute n'_{v_i} using BFS. Return

$$C' = \frac{n}{s}\sum_{i=1}^{s}\frac{1}{n'_{v_i}}$$

Runtime: $\Theta(1/\varepsilon^2)$ vertices sampled, each run takes $O(1/\varepsilon^2)$ time to compute. Adding results takes $O(s) = O(1/\varepsilon^2)$ time. The total runtime is $O(1/\varepsilon^4)$.

Correctness: To prove correctness, we show that with probability $\geq \frac{3}{4}$, we have

$$\left|\frac{n}{s}\sum_{i=1}^{s}\frac{1}{n'_{v_i}} - \sum_{v\in V}\frac{1}{n_v}\right| \leq \varepsilon n$$

Dividing both sides by $\frac{n}{s}$,

$$\left|\sum_{i=1}^{s}\frac{1}{n'_{v_i}} - \frac{s}{n}\sum_{v\in V}\frac{1}{n_v}\right| \leq \varepsilon s$$

By previous lemma and triangle inequality, it is enough to prove that with high probability $\geq \frac{3}{4}$

$$\left|\sum_{i=1}^{s}\frac{1}{n'_{v_i}} - \frac{s}{n}\sum_{v\in V}\frac{1}{n'_v}\right| \leq \frac{\varepsilon s}{2}$$

For the ith estimate, have a random variable $X_i = \frac{1}{n'_v}$ with probability $\frac{1}{n}$, $a_i = 0$, and $b_i = 1$. We can use Hoeffding's inequality. So $X = \sum_{i=1}^{s} X_i$.

$$\mathbb{E}[X] = \sum_{i=1}^{s}\mathbb{E}[X_i] = \sum_{i=1}^{s}\sum_{j=1}^{n}\frac{1}{n'_{v_j}}\cdot\frac{1}{n} = \frac{s}{n}\sum_{j=1}^{n}\frac{1}{n'_{v_j}}$$

This is exactly the quantity in the absolute value. So pick $l = \varepsilon \cdot s/2$

$$P[\text{fail}] \leq 2\exp\left(-\frac{\frac{2\varepsilon^2 s^2}{4}}{s}\right) = 2\exp\left(\frac{-\varepsilon^2 s}{2}\right) \leq \frac{1}{4}$$

Chapter 8

Random Walks and Markov Chains

8.1 Random Walks

Definition: Random Walk

Given a graph $G = (V, E)$, a random walk starts from a vertex v_0 and at each time step, it moves uniformly to a random neighbour of the current vertex in G.

$$v_{t+1} \leftarrow_R N_G(v_t)$$

Basic questions for random walks:

- Stationary distribution: does the random walk converge to a stable distribution?
- Mixing time: how long does it take for the walk to converge to the stationary distribution?
- Mean hitting time: starting from v_0, what is the expected number of steps until it reaches another vertex v_f?
- Cover time: how long does it take to reach every vertex of the graph at least once?

Example: Suppose $G = K_n$ and $a, b \in V$. We can ask all the questions above.

- Mean hitting time: Let X be the number of steps for random walk from $a \to b$.

$$P[X = 1] = \frac{1}{n-1}$$
$$P[X = 2] = \frac{n-2}{n-1} \cdot \frac{1}{n-1}$$
$$P[X = k] = \left(\frac{n-2}{n-1}\right)^{k-1} \cdot \frac{1}{n-1}$$

X is a geometric random variable with $p = \frac{1}{n-1}$

$$\mathbb{E}[X] = \frac{1}{p} = n - 1$$

- Stationary distribution: If at time t, we have $\left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$, then at $t+1$, the probability that we come back to any vertex is $(n-1) \cdot \frac{1}{n-1} \cdot \frac{1}{n} = \frac{1}{n}$.
- Cover time: is similar to the coupon collector problem.

8.2 Markov Chains

A random walk is a special kind of stochastic process:

$$P[X_t = v_t | X_0 = v_0, \ldots, X_{t-1} = v_{t-1}] = P[X_t = v_t | X_{t-1} = v_{t-1}]$$

in that the probability that we are at v_t at time t depends only on the state of our process at time $t-1$.

Definition: Markov Chain

A process in which the probability of an event only depends on the previous event.

A Markov chain can be seen as weighted directed graph, where the vertex is a state of the Markov chain and an edge (i, j) corresponds to the transition probability from i to j.

Definition: Irreducible

A Markov chain is irreducible if the underlying directed graph is strongly connected.

Definition: Transition Matrix

The weighted adjacency matrix $P \in \mathbb{R}^{n \times n}$ of the weighted directed graph of a Markov chain.
$P_{i,j}$ corresponds to the transition probability from i to j.

Definition: Probability Vector

$p_t \in \mathbb{R}^n$ is the probability vector where $p_t(i) := P[\text{state } i \text{ at time } t]$.

Definition: Transition

$$p_{t+1} = p_t \cdot P$$

Definition: Period

The period of a state i is

$$\gcd\{t \in \mathbb{N} : P^t_{i,i} > 0\}$$

That is, the GCD of all times t such that the probability of starting at i and being back at i at time t is positive.

The period of the square graph on 4 vertices is 2, since for any vertex, we have to take an even number of steps to get back to that vertex.

Definition: Aperiodic

State i is aperiodic if its period is equal to 1.
A Markov chain is aperiodic if all states are aperiodic, otherwise it is periodic.

Bipartite graphs yield periodic Markov chains.

Lemma

For any finite, irreducible, and aperiodic Markov chain, there exists $T < \infty$ such that

$$P^t_{i,j} > 0 \text{ for any } i, j \in V \text{ and } t \geq T$$

8.3 Stationary Distributions

Definition: Stationary Distribution

A stationary distribution of a Markov chain is a probability distribution $\pi \in \mathbb{R}^n$ such that

$$\pi P = \pi$$

Informally, π is an equilibrium/fixed point state, as we have $\pi = \pi P^t$ for any $t \geq 0$.

The intuition is if we run a finite, irreducible, and aperiodic Markov chain long enough, we will converge to a unique stationary distribution.

Definition: Total Variational Distance

Given two distributions $p, q \in \mathbb{R}^n$,

$$\Delta_{TV}(p, q) = \frac{1}{2}\sum_{i=1}^{n} |p_i - q_i| = \frac{1}{2}\|p - q\|_1$$

Proposition

p_t converges to q if and only if $\lim_{t\to\infty} \Delta_{TV}(p_t, q) = 0$.

8.3.1 Mixing Time

Definition: Mixing Time

The ε-mixing time of a Markov chain is the smallest t such that

$$\Delta_{TV}(p_t, \pi) \leq \varepsilon$$

regardless of the initial starting distribution p_0.

For the complete graph K_n, we have 0s along the diagonal of P and $\frac{1}{n-1}$ everywhere else. Let J be the all one's matrix and I be the identity matrix,

$$P = \frac{1}{n-1}J - \frac{1}{n-1}I = \frac{1}{n-1}\mathbf{1}^T\mathbf{1} - \frac{1}{n-1}I$$

The eigenvalue $\lambda_1 = 1$ gives $1P = \mathbf{1}$. The characteristic polynomial of P is

$$P = \lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T + \cdots + \lambda_n v_n v_n^T$$

$$P^t = \lambda_1^t v_1 v_1^T + \lambda_2^t v_2 v_2^T + \cdots + \lambda_n^t v_n v_n^T$$

$\lambda_2 = \cdots = \lambda_n = -\frac{1}{n-1}$ and the corresponding $v_1, \ldots, v_n$ are orthonormal.

Definition: Spectral Gap

The spectral gap λ is given by

$$\lambda = \min\{1 - \alpha_2, 1 - |\alpha_n|\}$$

so that $|\alpha_i| \leq 1 - \lambda$ for $2 \leq i \leq n$.

Theorem

The ε-mixing time of a random walk is upper bounded by

$$\frac{1}{\lambda}\log\left(\frac{n}{\varepsilon}\right)$$

8.4 Hitting Time

Definition: Hitting Time

Given states i, j in a Markov chain,

$$T_{i,j} := \min\{t \geq 1 : X_t = j, X_0 = i\}$$

$T_{i,j} = \infty$ if the Markov chain never visits j starting from i.

Definition: Mean Hitting Time

$$\tau_{i,j} := \mathbb{E}[T_{i,j}]$$

Lemma (Hitting Time Lemma)

For any finite, irreducible, and aperiodic Markov chain and for any two states i, j, not necessarily distinct, we have

$$P[T_{i,j} < \infty] = 1 \text{ and } \mathbb{E}[T_{i,j}] < \infty$$

Proof. We know we can find $M < \infty$ such that $(P^M)_{i,j} > 0$ for all i, j.

8.5 Linear Algebra Background

Definition: Eigenvalue/Eigenvector

Given a square matrix $A \in \mathbb{R}^{n \times n}$, $\lambda \in \mathbb{C}$ is an eigenvalue of A if there is a vector $v \in \mathbb{C}^n$ such that $Av = \lambda v$.
v is the eigenvector corresponding to λ.

Definition: Spectral Radius

Denoted $\rho(A)$, is the maximum absolute value of the eigenvalues of A.

Theorem (Gelfand's Formula)

$$\rho(A) = \lim_{t \to \infty} \left\| A^t \right\|_F^{1/t}$$

where $\|A\|_F = tr(A^T A)^{1/2}$.

Definition: Geometric Multiplicity

An eigenvalue λ has geometric multiplicity k if the space of eigenvectors of A with eigenvalue λ has dimension k.
That is, if the dimension of the nullspace of $A - \lambda I$ is k.

Definition: Algebraic Multiplicity

An eigenvalue λ of A has algebraic multiplicity k if $(t - \lambda)^k$ is the highest power of $t - \lambda$ dividing $\det(tI - A)$.

Lemma (Positivity Lemma)

If $A \in \mathbb{R}^{n\times n}$ is a positive matrix and $u, v \in \mathbb{R}^n$ are distinct vectors such that $u \geq v$, then $Au > Av$.
Moreover, there exists $\varepsilon > 0$ such that $Au > (1+\varepsilon)Av$.

Proof.

$$[A(u-v)]_i = \sum_j A_{ij}(u-v)j \geq \min_{i,j} A_{ij} \sum_{j=1}^{n} (u-v)_j$$

Since $u_j \geq v_j$, there is one index k such that $u_k > v_k$, we have

$$\sum_j (u-v)_j \geq u_k - v_k > 0$$

8.5.1 Perron-Frobenius

Theorem (Perron)

Let $A \in \mathbb{R}^{n\times n}$ be a positive matrix. Then, the following hold:

- $\rho(A)$ is an eigenvalue, and it has a positive eigenvector.
- $\rho(A)$ is the only eigenvalue in the complex circle $|\lambda| = \rho(A)$.
- $\rho(A)$ has geometric multiplicity 1.
- $\rho(A)$ has algebraic multiplicity 1.

Proof. (first point) By definition of $\rho(A)$, there is an eigenvalue $\lambda \in \mathbb{C}$ such that $|\lambda| = \rho(A)$. Let v be the corresponding eigenvector. Let u be the vector defined by $u_i = |v_i|$. Then we have

$$(Au)_i = \sum_j A_{ij}u_j \geq \left|\sum_j A_{ij}v_j\right| = |\lambda v_i| = \rho(A)\cdot u_i$$

so $Au \geq \rho(A)u$.

If the inequality is strict, then we have

$$A^2u > \rho(A)\cdot Au$$

and there is some positive $\varepsilon > 0$ such that

$$A^2u \geq (1+\varepsilon)\rho(A)Au$$

By induction, we have

$$A^nu \geq (1+\varepsilon)^{n-1}\cdot\rho(A)^{n-1}\cdot Au$$

By Gelfand's formula, we would have

$$\rho(A) = \lim_{n\to\infty} \|A^n\|_f^{1/n} \geq (1+\varepsilon)\rho(A)$$

which is a contradiction. So equality must hold.

(second point) We just proved that $\rho(A)$ is an eigenvalue, with eigenvector $u \geq 0$. Note that $u > 0$ since $\rho(A)u_i = (Au)_i > 0$. The only eigenvalue on the complex circle $|\mu| = \rho(A)$ is $\rho(A)$. If we had another eigenvalue $\lambda \neq \rho(A)$ in the circle $|\mu| = \rho(A)$, where z is the eigenvalue corresponding to λ, then we know that w defined as $w_i = |z_i|$ satisfies

$$Aw = \rho(A)w \Leftrightarrow \sum_j A_{ij}w_j = \rho(A) \cdot |z_i| = |\lambda z_i| = \left|\sum_j A_{ij}z_j\right|$$

for every $1 \leq i \leq n$.

Lemma

If the conditions above hold, then there is $\alpha \in \mathbb{C}$ nonzero such that $\alpha z \geq 0$.

But if $\alpha z \geq 0$ and a nonzero vector, we have

$$\lambda(\alpha z) = \alpha(\lambda z) = \alpha(Az) = A(\alpha z) \geq 0$$

since A is positive and $\alpha z \geq 0$. Thus, we know that λ is a nonnegative number. However, $\rho(A)$ is the only nonnegative number in the circle $|\mu| = \rho(A)$.

(third point) Suppose not. Let u, v be two linearly independent eigenvectors for $\rho(A)$. We can assume that both u, v are real vectors. If u was complex, then $u = \varphi + i\psi$, but $\rho(A)u = Au = A\varphi + iA\psi = \rho(A)\varphi + i\rho(A)\psi$, so we can just choose $u = \varphi$.

Let $\beta > 0$ such that $u - \beta v \geq 0$ and at least one entry is 0. $u - \beta v \neq 0$ since the vectors are linearly independent. But for each $1 \leq i \leq n$,

$$\rho(A) \cdot (u - \beta v)_i = (A(u - \beta v))_i > 0$$

which contradicts our choice of β. Thus, there cannot be two linearly independent vectors.

Theorem (Perron-Frobenius)

If a nonnegative matrix $A \in \mathbb{R}^{n \times n}$ is aperiodic and irreducible, then the following hold:

- $\rho(A)$ is an eigenvalue and it has positive eigenvector.
- $\rho(A)$ is the only eigenvalue in the complex circle $|\lambda| = \rho(A)$.
- $\rho(A)$ has geometric multiplicity 1.
- $\rho(A)$ has algebraic multiplicity 1.

Proof. A being aperiodic and irreducible implies that there is $m > 0$ such that A^m has all positive entries. Apply Perron's theorem to A^m and note that the eigenvalues of A^m are λ_i^m, where λ_i are the eigenvalues of A.

8.6 Fundamental Theorem of Markov Chains

Definition: Return Time

For state i

$$T_{i,i} := \min\{t \geq 1 : X_t = i, X_0 = i\}$$

Definition: Expected Return Time

$$\tau_{i,i} := \mathbb{E}[T_{i,i}]$$

The return time from state i to itself is $T_{i,i}$ and the expected return time is $\tau_{i,i} := \mathbb{E}[T_{i,i}]$.

Theorem (Fundamental Theorem of Markov Chains)

Any finite, irreducible, and aperiodic Markov chain has the following properties:

1. There exists a unique stationary distribution π, where $\pi_i > 0$ for all $i \in [n]$.
2. The sequence of distributions $\{p_t\}_{t \geq 0}$ will converge to π, no matter what the initial distribution is, i.e. for every distribution $p_0 \in \mathbb{R}^n_{\geq 0}$,
$$\lim_{t \to \infty} p_0 P^t = \pi$$
3. $$\pi_i = \lim_{t \to \infty} P^t_{i,i} = \frac{1}{\tau_{i,i}}$$

Proof. If our underlying graph is undirected, it is easy to guess the stationary distribution:

$$\pi_i = \frac{d_i}{2m}, m = |E|$$

If A_G is the adjacency matrix of G and $D = \text{diag}(d_1, \ldots, d_n)$ is the transition matrix, then

$$P = D^{-1} A_G$$

- P is not symmetric, but it is similar to a symmetric matrix
$$D^{1/2} P D^{-1/2} = D^{1/2} D^{-1} A_G D^{-1/2} = D^{-1/2} A_G D^{-1/2} = P'$$
- P and P' have the same eigenvalues and P' has only real eigenvalues.
- Eigenvectors of P are $D^{-1/2} v_i$ where v_i are eigenvectors of P'.
  Moreover, v_i can be taken to form an orthonormal basis.
- If a graph is strongly connected, then Perron-Frobenius for irreducible nonnegative matrices.

  - Unique eigenvector whose eigenvalue has maximum absolute value.
  - Eigenvector has all positive coordinates.
  - Eigenvalue is positive.
- This eigenvector is π, so all random walks converge to π.

8.7 Page Rank

Definition: Page Rank

A directed graph describing relationships between set of webpages. There is a directed edge (i, j) if there is a link from page i to page j.
Goal: rank how important a page is.

Page Rank Algorithm

1. Each page has pagerank value $\frac{1}{n}$
2. In each step, each page:
    1. Divides its pagerank value equally to its outgoing link
    2. Sends these equal shares to the pages it points to
    3. Updates its new pagerank value to be the sum of shares it receives

Equilibrium of pagerank values equal to probabilities of stationary distribution of random walk

$$P \in \mathbb{R}^{n \times n}, P_{i,j} = \frac{1}{\delta^{out}(i)}$$

Pagerank values and transition probabilities satisfy same equations:

$$p_{t+1}(j) = \sum_{i:(i,j) \in E} \frac{p_t(i)}{\delta^{out}(i)} \implies p_{t+1} = p_t \cdot P$$

If the graph is finite, irreducible, and aperiodic, then the fundamental theorem guarantees stationary distribution.

In practice, the graph may not satisfy the fundamental theorem's conditions. We can modify the original graph as follows:

- Fix a number $0 < s < 1$
- Divide s fraction of its pagerank value to its neighbours
- $1 - s$ fraction of its pagerank value to all nodes evenly

This is equivalent to the random walk. With probability s, we go to one of its neighbours and with probability $1 - s$, we go to a random page. The resulting graph is strongly connected and aperiodic, meaning it has a stationary distribution.

Part III

Mathematical Programming

Chapter 9

Linear Programming

9.1 Introduction

Definition: Mathematical Programming

Problems of the form

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_1(x) \leq 0 \\ & \vdots \\ & g_m(x) \leq 0 \\ & x \in \mathbb{R}^n \end{aligned}$$

Definition: Affine Function

$f : \mathbb{R}^n \to \mathbb{R}$ where

$$f(x) = c_1 x_1 + \cdots + c_n x_n + b = c^T x + b$$

Definition: Linear Programming

A mathematical programming problem where $f, g_1, \ldots, g_m$ are linear functions.

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{R}^n \end{aligned}$$

where we can change $A = \begin{pmatrix} A_1 & A_2 & \cdots & A_m \end{pmatrix}, b = \begin{pmatrix} b_1 & b_2 & \cdots & b_m \end{pmatrix}$ and the constraints are $Ax \leq b$.

Definition: Standard Form

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Stock portfolio optimization: n companies, stock of company i costs $c_i \in \mathbb{R}$. Each company i has expected profit $p_i \in \mathbb{R}$ and the budget is $B \in \mathbb{R}$.

$$\begin{aligned} \max \quad & p^T x \\ \text{s.t.} \quad & c^T x \leq B \\ & x \geq 0 \end{aligned}$$

9.2 Fundamental Theorem of Linear Inequalities

Theorem (Farkas (1894, 1898), Minkowski 1896)

Let $a_1, \ldots, a_m, b \in \mathbb{R}^n$ and $t = \text{rank}(a_1, \ldots, a_m, b)$. Then either

1. b is a convex combination of linearly independent vectors from $a_1, \ldots, a_m$.
2. There exists a hyperplane $H = \{x : c^T x = 0\}$ such that
   - $c^T b < 0$
   - $c^T a_i \geq 0$
   - H contains $t - 1$ linearly independent vectors from $a_1, \ldots, a_m$

Definition: Separating Hyperplane

The hyperplane H in the theorem is a separating hyperplane.

Lemma (Farkas' Lemma)

Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. The following are equivalent:

- There exists $x \in \mathbb{R}^n$ such that $x \geq 0$ and $Ax = b$.
- $y^T b \geq 0$ for each $y \in \mathbb{R}^m$ such that $y^T A \geq 0$.

Lemma (Farkas' Lemma - Variant 1)

Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Then exactly one of the following statements hold:

- There exists $x \in \mathbb{R}^n$ such that $x \geq 0$ and $Ax = b$.
- There exists $y \in \mathbb{R}^m$ such that $y^T b \geq 0$ and $y^T A \leq 0$.

Lemma (Farkas' Lemma - Variant 2)

Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. The following are equivalent:

- There exists $x \in \mathbb{R}^n$ such that $Ax \leq b$.
- $y^T b \geq 0$ for each $y \geq 0$ such that $y^T A = 0$.

Proof. Let $M = \begin{bmatrix} I & A & -A \end{bmatrix}$. Then $Ax \leq b$ has a solution if and only if $Mz = b$ has a nonnegative solution $z \geq 0$. Now apply the original version of the lemma.

9.3 Duality Theory

From Farkas' lemma, we have that $Ax = b$ and $x \geq 0$ if and only if $y^T b \geq 0$ for each $y \in \mathbb{R}^m$ such that $y^T A \geq 0$.

$$\begin{aligned} y^T A \leq c^T &\implies y^T A x \leq c^T x \\ &\implies y^T b \leq c^T x \end{aligned}$$

Thus, if $y^T A \leq c^T$, then we have $y^T b$ is a lower bound on the solution to the linear program.

Consider the primal LP $\min\{c^T x : Ax = b, x \geq 0\}$ and the dual LP $\max\{y^T b : y^T A \leq c^T\}$.

Theorem (Weak Duality)

Let x be a feasible solution of the primal LP (minimization) and y be a feasible solution of the dual LP (maximization). Then

$$y^T b \leq c^T x$$

Let $\alpha^*, \beta^* \in \mathbb{R}$ be the optimal values for the primal and dual respectively.

- If the primal and dual are feasible, then

$$\beta^* \leq \alpha^*$$

- If the primal is **unbounded**, then the dual is **infeasible**.
- If the dual is **unbounded**, then the primal is **infeasible**.

Theorem (Strong Duality)

If the primal LP and dual LP are feasible, then

$$b^T y = \beta^* = \alpha^* = c^T x$$

Proof. Consider the LP

$$\begin{aligned} \min \quad & 0 \\ \text{s.t.} \quad & y^T A \leq c^T \\ & c^T x - y^T b \leq 0 \\ & Ax = b \\ & x \geq 0 \end{aligned}$$

The LP is encoded by

$$B \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} A & 0 \\ -A & 0 \\ c^T & -b^T \\ 0 & A^T \\ -I & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} b \\ -b \\ 0 \\ c \\ 0 \end{pmatrix}$$

Farkas' lemma (variant 2) gives the system has a solution if and only if for each $z = (u^T, v^T, \lambda, w^T, \alpha^T) \geq 0$ such that $zB = 0$ then we have $u^T b - v^T b + w^T c \geq 0$ (inner product with the right hand vector.

If $\lambda > 0$, and $zB = 0$ if and only if $u^T A - v^T A + \lambda c^T - \alpha^T = 0$ or $u^T A - v^T A + \lambda c^T \geq 0$ since $\alpha \geq 0$. Also, $w^T A^T = \lambda b^T$ or $Aw = \lambda b$. Solving, we have

$$\lambda[(u^T - v^T)b + c^T w] \geq 0$$

If $\lambda = 0$, let x, y be feasible solutions. Then $x \geq 0, Ax = b, y^T A \leq c^T$. Thus,

$$c^T w \geq y^T A w = 0 \geq (v^T - u^T)Ax = (v^T - u^T)b$$

Lemma (Affine Farkas' Lemma)

Let the system $Ax \leq b$ have at least one solution, and suppose that the inequality

$$c^T x \leq \delta$$

holds whenever x satisfies $Ax \leq b$. Then for some $\delta' \leq \delta$, the linear inequality

$$c^T x \leq \delta'$$

is a nonnegative linear combination of the inequalities of $Ax \leq b$.

9.3.1 Complementary Slackness

Theorem (Complementary Slackness)

If the optima in both the primal and dual is finite and x, y are feasible solutions, then the following are equivalent:

- x, y are optimal solutions to the primal and dual.
- $c^T x = y^T b$.
- If $x_i > 0$, then the corresponding inequality $y^T A_i \leq c_i$ is an equality; $y^T A_i = c_i$.

9.4 Applications of LP Duality

9.4.1 Game Theory - Minimax Theorems

Let there be two players Alice and Bob where each player has a finite set of strategies $S_A = \{1, \ldots, m\}$ and $S_B = \{1, \ldots, n\}$. There are payoff matrices $A, B \in \mathbb{R}^{m \times n}$ for Alice and Bob, respectively, where if Alice plays i and Bob plays j, then

- Alice gets A_{ij}
- Bob gets B_{ij}

Assume players are rational, i.e. want to maximize their payoffs.

Definition: Nash Equilibrium

A strategy profile (i, j) is called a Nash equilibrium if the strategy played by each player is optimal, given the strategy of the other player. That is

- $A_{ij} \geq A_{kj}$ for all $k \in S_A$.
- $B_{ij} \geq B_{il}$ for all $l \in S_B$

Battle of the Sexes Game: $(2, 1), (1, 2)$ are Nash equilibria.

	Football	Opera
Football	$(2, 1)$	$(0, 0)$
Opera	$(0, 0)$	$(1, 2)$

Table 9.1: Battle of the sexes Payoff Matrix

Prisoner's Dilemma: The number in each cell is for example the number of years of jail. $(-5, 5)$ is the Nash equilibrium.

	Silent	Snitch
Silent	$(-1,-1)$	$(-10,0)$
Snitch	$(0,-10)$	$(-5,-5)$

Table 9.2: Prisoner's Dilemma

Definition: Mixed Strategy

A mixed strategy is a probability distribution over a set of pure strategies S. If Alice's strategies are $S_A = \{1,\dots,n\}$, her mixed strategies are

$$\Delta_A := \{x \in \mathbb{R}^n : x \geq 0, \|x\|_1 = 1\}$$

Models situation where players choose their strategy "at random".

Definition: Expected Gain

Payoffs for each player in mixed strategy.

(x,y) is the profile of mixed strategies used by Alice and Bob. We have

$$v_A(x,y) = \sum_{(i,j)\in S_A\times S_B} A_{ij}x_iy_j = x^TAy$$
$$v_B(x,y) = \sum_{(i,j)\in S_A\times S_B} B_{ij}x_iy_j = x^TBy$$

Definition: (Mixed) Nash Equilibrium

A strategy profile $x \in \Delta_A, y \in \Delta_B$ is called a mixed Nash equilibrium if the strategy played by each player is optimal, given the strategy of the other player. That is

- $x^TAy \geq z^TAy$ for all $z \in \Delta_A$.
- $x^TBy \geq x^TBw$ for all $w \in \Delta_B$.

	Jump left	Jump right
Kick left	$(-1,1)$	$(1,-1)$
Kick right	$(1,-1)$	$(-1,1)$

Table 9.3: Penalty Kick

There is no pure Nash equilibrium.

Definition: Zero-Sum Game

Payoff matrices satisfy $A = -B$.

Since it is a zero-sum game, $A = -B$, then Alice wants to maximize her payoff, while Bob wants to maximize his payoff, but $B = -A$, so it is equivalent to minimizing his payoff.

Theorem (Minimax Theorem)

In a zero-sum game, for any payoff matrix $A \in \mathbb{R}^{m \times n}$:

$$\max_{x \in \Delta_A} \min_{y \in \Delta_B} x^T A y = \min_{y \in \Delta_B} \max_{x \in \Delta_A} x^T A y$$

Proof. For given $x \in \Delta_A$,

$$\min_{y \in \Delta_B} x^T A y = \min_{j \in S_B} (x^T A)_j$$

The left hand side can be written as

$$\begin{aligned} \max \quad & s \\ \text{s.t.} \quad & s \leq (x^T A)_j, \ \forall j \in S_B \\ & \sum_{i \in S_A} x_i = 1 \\ & x \geq 0 \end{aligned}$$

For given $y \in \Delta_B$,

$$\max_{x \in \Delta_A} x^T A y = \max_{i \in S_A} (Ay)_i$$

The right hand side can be written as

$$\begin{aligned} \min \quad & t \\ \text{s.t.} \quad & t \geq (Ay)_i, \ \forall i \in S_A \\ & \sum_{j \in S_B} y_j = 1 \\ & y \geq 0 \end{aligned}$$

There programs are duals of each other.

Proof. (Dual) Use the second LP,

$$\begin{aligned} \min \quad & t_+ - t_- \\ \text{s.t.} \quad & t_+ - t_- w_i - (Ay)_i = 0, \ \forall i \in S_A \\ & \sum_{j \in S_B} y_j = 1 \\ & y \geq 0 \end{aligned}$$

which is

$$\begin{aligned} \min \quad & (e_1 - e_2)^T z \\ \text{s.t.} \quad & C = \begin{bmatrix} 0 & 0 & I & 0 \\ 1 & -1 & -A & -I \end{bmatrix} \begin{bmatrix} t_+ \\ t_- \\ y \\ w \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ & z \geq 0 \end{aligned}$$

finding the dual results in the first LP

9.4.2 Learning Theory - Boosting

Consider the classification problem over $\mathcal{X} = \{x_1, \ldots, x_m\}$ where the set of hypothesis $\mathcal{H} := \{h : \mathcal{X} \to \{0, 1\}\}$ and each $x \in \mathcal{X}$ has a correct value $c(x) \in \{0, 1\}$. Data is sampled from an unknown distribution $q \in \Delta_{\mathcal{X}}$.

Weak Learning Assumption

For any distribution $q \in \Delta_{\mathcal{X}}$, there is a hypothesis $h \in \mathcal{H}$ which is wrong less than half the time.

$$\exists \gamma > 0, \forall q \in \Delta_{\mathcal{X}}, \exists h \in \mathcal{H}, P_{x \sim q}[h(x) \neq c(x)] \leq \frac{1-\gamma}{2}$$

Weak learning assumption implies something much stronger: it is possible to combine classifiers in $\mathcal{H}$ to construct a classifier that is always right (strong learning)

Theorem

Let $\mathcal{H}$ be a set of hypotheses satisfying weak learning assumption. Then there is a distribution $p \in \Delta_{\mathcal{H}}$ such that the weighted majority classifier

$$c_p(x) := \begin{cases} 1 & \text{if } \sum_{h \in \mathcal{H}} p_h h(x) \geq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

is always correct. That is, $c_p(x) = c(x)$ for all $x \in \mathcal{X}$.

Proof. Let $M \in \{-1, 1\}^{m \times n}$ where $m = |\mathcal{X}|$ and $n = |\mathcal{H}|$.

$$M_{ij} = \begin{cases} +1 & \text{if classifier } h_j \text{ wrong on } x_i \\ -1 & \text{otherwise} \end{cases}$$

Weak learning:

$$\sum_{1 \leq i \leq m} q_i \cdot \delta_{h_j(x_i) \neq c(x_i)} \leq \frac{1-\gamma}{2}$$

Note that $M_{ij} = 2 \cdot \delta_{h_j(x_i) \neq c(x_i)} - 1$, where $\delta_{h_j(x_i) \neq c(x_i)} = 1$ if they are different and 0 if they are the same, and

$$q^T M e_j \leq -\gamma \implies q^T M p \leq -\gamma$$

for any $p \in \Delta_{\mathcal{H}}$. By minimax, we have

$$\max_{q \in \Delta_{\mathcal{X}}} \min_{p \in \Delta_{\mathcal{H}}} q^T M p = \min_{p \in \Delta_{\mathcal{H}}} \max_{q \in \Delta_{\mathcal{X}}} q^T M p \leq -\gamma$$

In particular, the right hand side implies weighted classifier given by optimal solution p^* is always correct.

Proof. Pick an image x_i is equivalent to choosing $q = e_i$. We know that $e_i^T M p^* \leq -\gamma$.

$$\sum_{j=1}^{n} M_{ij} p_j^* \leq -\gamma \iff \sum_{j=1}^{n} p_j^* \delta_{h_j(x_i) \neq c(x_i)} \leq \frac{1-\gamma}{2}$$

9.4.3 Combinatorics - Bipartite Matching

Given a bipartite graph $G = (L \cup R, E)$, does it have a perfect matching? We previously saw that we can randomly isolate a perfect matching, but we want to remove the randomness. This would lead to a fast parallel algorithm for matchings.

Definition: Circulation

Given an even cycle $C = (e_1, e_2, \ldots, e_{2k})$, the circulation of C is given by

$$circ(C) = |w(e_1) - w(e_2) + \cdots + w(e_{2k-1}) - w(e_{2k})|$$

Lemma

If we assign weights $w(e_i)$ such that $circ(C) \neq 0$ for each cycle C of the bipartite graph G, then we get the minimum weight perfect matching is unique.

Suppose we have a weight assignment w. Let G_w be the subgraph of G given by the union of all min w-weight perfect matchings in G.

Claim

Circulation of each even cycle in G_w is zero.

Proof. Use LP duality. The primal problem is

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e x_e \\ \text{s.t.} \quad & \sum_{e \in \delta(u)} x_e = 1, \ \forall e \in L \cup R \\ & x \geq 0 \end{aligned}$$

The dual is

$$\begin{aligned} \max \quad & \sum_{u \in L \cup R} y_u \\ \text{s.t.} \quad & y_u + y_v \leq w_e, \ \forall e = (u, v) \in E \end{aligned}$$

By complementary slackness, $x_e \neq 0$ in the primal means $y_u + y_v = w_e$ is the dual to be optimal.

Chapter 10

Semidefinite Programming

10.1 Positive Semidefinite Matrices

Definition: Symmetric Matrix

A matrix $S \in \mathbb{R}^{n \times n}$ is symmetric if $S_{ij} = S_{ji}$ for all $i, j \in [n]$.

Theorem (Spectral Theorem)

Any symmetric matrix in $\mathbb{R}^{n \times n}$ has n real eigenvalues and an orthonormal basis in $\mathbb{R}^n$ for the eigenvectors, i.e. we can write

$$S = \sum_{i=1}^{n} \lambda_i v_i v_i^T$$

where $\lambda_i \in \mathbb{R}$ and $v_i \in \mathbb{R}^n$ such that $\langle v_i, v_j \rangle = \delta_{ij}$.

Definition: Positive Semidefinite (PSD)

A symmetric matrix $S \in \mathbb{R}^{n \times n}$ having only nonnegative eigenvalues. We write $S \succeq 0$.

Lemma (Equivalent Characterizations of PSD Matrices)

- All eigenvalues of S are nonnegative.
- $S = Y^T Y$ for some $Y \in \mathbb{R}^{d \times n}$ where $d \leq n$ (smallest d is equal to rank).
- $x^T S x \geq 0$ for all $x \in \mathbb{R}^n$.
- $S = LDL^T$, where D is a nonnegative diagonal matrix and L is the unit lower-triangular matrix (only 1's along diagonal, other values in the lower-triangular part).
- S is in the convex hull of the set

$$\{uu^T : u \in \mathbb{R}^n\}$$

- $S = U^T D U$ where D is a nonnegative diagonal matrix and $U \in \mathbb{R}^{n \times n}$ is an orthonormal matrix ($U^T U = I$).
- Any principal minor of A has nonnegative determinant (choose $A \subseteq [n]$, and remove all rows/columns not in A).

10.2 Semidefinite Programming Formulation

Let $\mathcal{S}^m := \mathcal{S}^m(\mathbb{R})$ be the space of all $m \times m$ real symmetric matrices.

Definition: Semidefinite Programming (SDP)

Deals with problems of the form

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & x_1 \cdot A_1 + \cdots + x_n \cdot A_n \succeq B \\ & x \in \mathbb{R}^n \\ & A_i, B \in \mathcal{S}^m(\mathbb{R}) \end{aligned}$$

$C \succeq D$ denotes that $C - D \succeq 0$, i.e. $C - D$ is PSD.

10.2.1 LP vs. SDP

Let the LP $\min\{a^T x : Cx \geq b : x \in \mathbb{R}^n\}$ and the SDP $\min\{c^T x : x_1 \cdot A_1 + \cdots + x_n \cdot A_n \succeq B, x \in \mathbb{R}^n\}$. If $(Ax)_i \geq b_i$, we can set A_j's to be diagonal matrices where each entry in diagonal is A_{ij} and $B = diag(b_1, \ldots, b_m)$. Then

$$\sum_j x_j A_j - B = diag(\sum A_{1j} x_j - b_1, \ldots, \sum A_{mj} x_j - b_m)$$

10.3 Convex Algebraic Geometry

Definition: Linear Matrix Inequalities (LMI)

A linear matrix inequality is an inequality of the form

$$A_0 + \sum_{i=1}^{n} A_i x_i \succeq 0$$

where $A_0, \ldots, A_n$ are symmetric matrices.

Definition: Spectrahedron

A set defined by finite many linear matrix inequality.

$$S = \left\{ x \in \mathbb{R}^n : \sum_{i=1}^{n} A_i x_i \succeq B, A_i, B \in \mathcal{S}^m \right\}$$

It turns out, it is a set defined by only one LMI. If we had $\sum_i A_i x_i \succeq B$ and $\sum_i C_i x_i \succeq D$, then $E_i = \begin{bmatrix} A_i & 0 \\ 0 & C_i \end{bmatrix}$ and $F = \begin{bmatrix} B & 0 \\ 0 & D \end{bmatrix}$, and this all turns into

$$\sum_i E_i x_i \succeq F \implies \begin{bmatrix} \sum_i A_i x_i - B & 0 \\ 0 & \sum_i C_i x_i - D \end{bmatrix}$$

Examples of spectrahedra:

- Circle: $\mathcal{C} = \{(x, y) \in \mathbb{R}^n : x^2 + y^2 \leq 1\}$. Then
$$\begin{bmatrix} 1 - x & y \\ y & 1 + x \end{bmatrix} \succeq 0$$
Principal minors determinants: $1 - x \geq 0, 1 + x \geq 0, 1 - x^2 - y^2 \geq 0$.
- Hyperbola: $\mathcal{H} = \{(x, y) \in \mathbb{R}^2 : xy \geq 1, x \geq 0, y \geq 0\}$. Then
$$\begin{bmatrix} x & 1 \\ 1 & y \end{bmatrix}$$
Principal minors determinants: $xy - 1 \geq 0, x \geq 0, y \geq 0$.
- Elliptic Curve: $\mathcal{E} = \{(x, y) \in \mathbb{R}^2 : -2y^2 - x^3 - 3x^2 + x + 3 = 0\}$. Then
$$\begin{bmatrix} x + 1 & 0 & y \\ 0 & 2 & -x - 1 \\ y & -x - 1 & 2 \end{bmatrix}$$
Principal minors determinants: $x + 1 \geq 0, 4 - (x + y)^2 \geq 0, 2x + 2 - y^2 \geq 0, -2y^2 - x^3 - 3x^2 + x + 3 \geq 0$.

Definition: Projected Spectrahedron

A set $S \in \mathbb{R}^n$ that has the form

$$S = \left\{ x \in \mathbb{R}^n : \exists y \in \mathbb{R}^t \text{ s.t. } \sum_{i=1}^{n} A_i x_i + \sum_{j=1}^{t} B_j y_j \succeq C, A_i, B_j, C \in \mathcal{S}^m \right\}$$

Examples of projected spectrahedra:

- Projected hyperbola: $\mathbb{R}_{>0} = \left\{ x \in \mathbb{R} : \exists y \in \mathbb{R}, \begin{bmatrix} x & 1 \\ 1 & y \end{bmatrix} \succeq 0 \right\}$.
- Projected quadratic cone intersected with halfspace:

$$S = \left\{ (x, y) \in \mathbb{R}^2 : \exists z \in \mathbb{R}, \begin{bmatrix} z+y & 2z-x \\ 2z-x & z-y \end{bmatrix} \succeq 0 \right\}$$

10.3.1 Testing Points in Spectrahedron

To be able to optimize, we must be able to test whether a point $x \in \mathbb{R}^n$ is inside our spectrahedron

$$S = \left\{ x \in \mathbb{R}^n : \sum_{i=1}^{n} A_i x_i \succeq B, A_i, B \in \mathcal{S}^m \right\}$$

By definition of $x \in S$, it is equivalent to saying $Z = \sum_{i=1}^{n} A_i x_i - B \succeq 0$. We can use **Symmetric Gaussian Elimination**.

Let $Z = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1n} \\ z_{12} & & \cdots & \\ \vdots & & \cdots & \\ z_{1n} & & \cdots & \end{bmatrix}$, $L_1 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -z_{12}/z_{11} & 1 & \cdots & 0 \\ \vdots & 0 & \cdots & 0 \\ -z_{1n}/z_{11} & 0 & \cdots & 1 \end{bmatrix}$. Then $L_1 Z L_1^T = diag(z_{11}, Z_1)$ is symmetric. If $z_{11} = 0$, then the first row and column must all be 0, otherwise Z is not PSD.

This algorithm has strongly polynomial runtime.

10.4 Control Theory

Definition: Linear Difference Equation

$$x(t+1) = Ax(t), \; x(0) = x_0$$

When A and x_0 are nonnegative, this is a Markov chain.

When $t \to \infty$, under what conditions will $x(t)$ remain bounded? Or go to zero?

Definition: Stable

When the system $x(t)$ converges to zero.

Proposition

The system $x(t)$ is stable if and only if $|\lambda_i(A)| < 1$.

Definition: Lyapunov Functions

Generalize energy in systems. Functions on $x(t)$ decrease monotonically on trajectories of the system.
For discrete-time system,

$$V(x(t)) = x(t)^T P x(t)$$

To make it monotonically decreasing, we need

$$\begin{aligned} V(x(t+1)) \leq V(x(t)) &\iff x(x+1)^T P x(t+1) - x(t)^T P x(t) \leq 0 \\ &\iff x(t)^T A^T P A x(t) - x(t)^T P x(t) \leq 0 \\ &\iff A^T P A - P \preceq 0 \end{aligned}$$

Theorem

Given a matrix $A \in \mathbb{R}^{m \times m}$, the following conditions are equivalent:

1. All eigenvalues of A are inside the unit circle, i.e. $|\lambda_i(A)| < 1$.
2. There is $P \in \mathcal{S}^m$ such that $P \succ 0$ and $A^T P A - P \prec 0$.

Proof. ($\implies$) Let $P = \sum_{i=1}^{\infty} (AA^T)^i \succ 0$.

$$A^T P A - P = \sum_{i=1}^{\infty} (A^T A)^i - \sum_{i=0}^{\infty} (AA^T)^i = -I + \sum_{i \geq 1} (A^T A) - \sum_{i \geq 1} (AA^T) \prec 0$$

($\impliedby$) We know $A^T P A - P \prec 0$. Then $x^T (A^T P A - P) x < 0$. If $x = v_i$ be the eigenvector corresponding to eigenvalue λ_i. Then,

$$(Ax)^T P (Ax) - x^T P x < 0 \implies \lambda_i v_i^T P \lambda_i v_i - v_i^T P v_i < 0 \implies (|\lambda_i|^2 - 1) v_i^T P v_i < 0$$

Since P is positive definite, then $v_i^T P v_i > 0$, then $|\lambda_i| < 1$ for the inequality to hold.

Definition: Linear Difference Equation with Control Input

Let $A \in \mathbb{R}^{m \times m}, B \in \mathbb{R}^{m \times k}$,

$$x(t+1) = Ax(t) + Bu(t), \; x(0) = x_0$$

If we properly choose control input $u(t)$, we can make our system $x(t)$ behave in a way that we want. We want to set the control input to be $u(t) = Kx(t)$ for some fixed K. This

is equivalent as replacing A with $A + BK$. This is harder to solve via simple eigenvalue description, but still solved in the same way via Lyapunov functions. We want $P \succ 0$ such that

$$(A + BK)^T P(A + BK) - P \prec 0$$

We can make this into an SDP with some matrix manipulations.

10.5 Duality Theory

Definition: Frobenius Inner Product

Let $A, B \in \mathcal{S}^m$, then

$$\langle A, B \rangle := tr[AB] = \sum_{i,j} A_{ij} B_{ij}$$

Definition: Frobenius Norm

$$\|A\|_F = \sqrt{\langle A, A \rangle} = \sqrt{\sum_{i,j} A_{ij}^2}$$

Definition: Polar Dual

Given a spectrahedron $S \subseteq \mathcal{S}^m$,

$$S^\circ = \{Y \in \mathcal{S}^m : \langle Y, X \rangle \leq 1, \forall X \in S\}$$

Definition: Standard Semidefinite Program

$$\begin{aligned} \min \quad & \langle C, X \rangle \\ \text{s.t.} \quad & \langle A_i, X \rangle = b_i \\ & X \succeq 0 \end{aligned}$$

The variables are a positive semidefinite matrix X, each constraint is given by the Frobenius inner product $\langle A_i, X \rangle = b_i$ and we can obtain the LP standard form by making X and A_i's to be diagonal.

Standard Primal as LMI: $\langle A_i, X \rangle = \sum_{j \leq k} A_{ijk} x_{jk}$, then $L = diag(\sum A_{1jk} x_{jk}, \ldots, \sum A_{tjk} x_{jk})$,

$$\begin{bmatrix} L - diag(b) & 0 & 0 \\ 0 & -L + diag(b) & 0 \\ 0 & 0 & X \end{bmatrix}$$

$$\begin{aligned} \min \quad & \langle C, X \rangle \\ \text{s.t.} \quad & \sum_{i,j} \Gamma_{ij} x_{ij} + \Gamma_0 \succeq 0 \\ & X \in \mathbb{R}^{\binom{m+1}{2}} \end{aligned}$$

$$\Gamma_0 = \begin{bmatrix} -diag(b) & 0 & 0 \\ 0 & diag(b) & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ and } \Gamma_{jk} = diag(diag(A_{ijk}), -diag(A_{ijk}), E_{jk} + E_{kj}).$$

Proposition

If $X \succeq 0$ and $A \succeq B$, then $\langle A, X \rangle \succeq \langle B, X \rangle$.

Consider the SDP and multiply ith equality by variable y_i:

$$\sum_{i=1}^{t} y_i \langle A_i, X \rangle = \sum_{i=1}^{t} y_i b_i \implies \left\langle \sum_{i=1}^{t} y_i A_i, X \right\rangle = y^T b$$

Thus, if $\sum_{i=1}^{t} y_i A_i \preceq C$, then

$$y^T b = \left\langle \sum_{i=1}^{t} y_i A_i, X \right\rangle \leq \langle C, X \rangle$$

Therefore, $y^T b$ is a lower bound on the SDP.

Definition: Dual Semidefinite Program

$$\begin{aligned} \max \quad & y^T b \\ \text{s.t.} \quad & \sum_{i=1}^{t} y_i A_i \preceq C \end{aligned}$$

Theorem (Weak Duality)

Let X be a feasible solution of the primal SDP and y be a feasible solution of the dual SDP. Then

$$y^T b \leq \langle C, X \rangle$$

Theorem (Complementary Slackness)

Let X be a feasible solution of the primal SDP and y be feasible solution of the dual SDP. If (X, y) satisfy the complementary slackness condition

$$\left(C - \sum_{i=1}^{t} y_i A_i \right) X = 0$$

Then (X, y) are primal and dual optimum solutions of the SDP problem.

Strong duality for SDPs are more complex than LPs; both the primal and dual may be feasible, but strong duality may not hold.

Definition: Strictly Feasible

The primal SDP is strictly feasible if there is a feasible solution $X \succ 0$.

The dual SDP is strictly feasible if there is a feasible solution y such that $\sum_{i=1}^{t} y_i A_i \prec C$.

Theorem (Strong Duality under Slater Conditions)

If the primal SDP and dual SDP are both strictly feasible, then the optimal value of the primal equals the optimal value of the dual.

Part IV

Approximation Algorithms

Chapter 11

Linear Programming Relaxations and Rounding

Many optimization problems are NP-hard. We want to find approximate solutions in polynomial time and even for problems in P.

Definition: Integer Linear Program (ILP)

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{N}^n \end{aligned}$$

ILPs capture many combinatorial optimization problems, but also capture NP-hard problems.

11.1 Independent Set

Maximum Independent Set Problem

Let $G = (V, E)$ be a graph. Find an maximal independent set $S \subseteq V$ such that $u, v \in S \implies \{u, v\} \notin E$.

Maximum Independent Set ILP

$$\begin{aligned} \max \quad & \sum_{v \in V} x_v \\ \text{s.t.} \quad & x_u + x_v \leq 1, \ \forall \{u, v\} \in E \\ & x_v \in \{0, 1\}, \ \forall v \in V \end{aligned}$$

To get efficient (exact or approximate) algorithms for problems, the following strategy is useful:

1. Formulate combinatorial optimization problem as ILP.
2. Derive LP from ILP by removing integral constraints (LP relaxation).
3. We are minimizing over the same objective function, by over a potentially larger set of solutions.
$$\text{opt(LP)} \leq \text{opt(ILP)}$$
4. Solve the LP using an efficient algorithm. If the solution is integral, then we have our optimal solution, otherwise, if we have fractional values, we have to round our answer by some procedure so that
$$\text{opt(LP)} \leq \text{rounded solution} \leq c \cdot \text{opt(ILP)}$$
where c is our *approximation* factor

Let $P = \{x \in \mathbb{R}^n_{\geq 0} : Ax = b\}$ be the polytope defined by the LP $\min\{c^T x : Ax = b, x \geq 0\}$.

Definition: Vertex Solutions

A solution $x \in P$ is a vertex solution if $\nexists y \neq 0$ such that $x + y \in P$ and $x - y \in P$.

Definition: Extreme Point Solutions

$x \in P$ is an extreme point solution if $\exists u \in \mathbb{R}^n$ such that x is the unique optimum solution to the LP with constraint P and objective $u^T x$.

Definition: Basic Solutions

Let $supp(x) := \{i \in [n] : x_i > 0\}$ be the set of nonzero coordinates of x. Then $x \in P$ is a basic solution if and only if the columns of A indexed by $supp(x)$ are linearly independent.

11.2 Vertex Cover

Minimum Vertex Cover Problem

Input: graph $G = (V, E)$ (can have cost c_v for all $v \in V$).
Output: minimum set $S \subseteq V$ such that for each $\{u, v\} \in E$, we have

$$|S \cap \{u, v\}| \geq 1$$

Weighted Version: each vertex $v \in V$ has a cost $c_v \in \mathbb{R}_{\geq 0}$.

Vertex Cover ILP

$$\begin{aligned} \min \quad & \sum_{u \in V} c_u x_u \\ \text{s.t.} \quad & x_u + x_v \geq 1, \ \forall \{u, v\} \in E \\ & x_u \in \{0, 1\}, \ \forall u \in V \end{aligned}$$

11.2.1 Unweighted Simple 2-Approximation

1: List edges E in any order
2: $S = \emptyset$
3: **for** $\{u, v\} \in E$ **do**
4: **if** $S \cap \{u, v\} = \emptyset$ **then**
5: $S = S \cup \{u, v\}$
6: **return** S

Correctness: By construction S is a vertex cover. If added elements to S k times, then $|S| = 2k$ and G has a matching of size k, which means that optimum vertex cover is at least k. So we get a 2-approximation.

11.2.2 LP Relaxation

Drop the integrality constraints

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & x_u + x_v \geq 1, \ \forall \{u, v\} \in E \\ & 0 \leq x_u \leq 1, \ \forall u \in V \end{aligned}$$

Solve this LP and get the optimal solution z and round z_v to the nearest integer to get the solution y. Since each edge is covered, at least one of z_u, z_v is $\geq 1/2$ by the feasibility of LP.

Cost of y is

$$\sum_{u \in V} c_u y_u \leq \sum_{u \in V} c_u (2 z_u) \leq 2OPT(\text{ILP})$$

11.3 Set Cover

Minimum Set Cover Problem

Input: a finite set U and a collection $S_1, \ldots, S_n$ of subsets of U.
Output: fewest collection of sets $I \subseteq [n]$ such that

$$\bigcup_{j \in I} S_j = U$$

Weighted version: each set S_i has a weight $w_i \in \mathbb{R}_{\geq 0}$.

Set Cover ILP

$$\begin{aligned} \min \quad & \sum_{i \in [n]} w_i x_i \\ \text{s.t.} \quad & \sum_{i : v \in S_i} x_i \geq 1, \ \forall v \in U \\ & x_i \in \{0, 1\}, \ \forall i \in [n] \end{aligned}$$

11.3.1 LP Relaxation

$$\begin{aligned} \min \quad & \sum_{i \in [n]} w_i x_i \\ \text{s.t.} \quad & \sum_{i : v \in S_i} x_i \geq 1, \ \forall v \in U \\ & 0 \leq x_i \leq 1, \ \forall i \in [n] \end{aligned}$$

Suppose we get a feasible solution $z \in [0, 1]^n$, can we just round each z_i to the nearest integer?

No, say $v \in U$ is in 20 sets and $z_i = \frac{1}{20}$ for each of the sets $v \in S_i$, then rounding would not select any S_i where $v \in S_i$.

11.3.2 Random Pick

Think of z_i as the probability we would pick S_i. z describes an optimal probability distribution over ways to choose S_i.

Algorithm 4 Random Pick

Input: $z = (z_1, \ldots, z_n) \in [0,1]^n$ such that z is optimal to LP.
Output: set cover for U
$I = \emptyset$
for $i = 1$ to n **do**
    $I = I \cup \{i\}$ with probability z_i
return I

Consider the random pick process from point of view $v \in U$. Let $v \in S_1, \ldots, S_k$. We select S_i with probability z_i such that $\sum_{i=1}^{k} z_i \geq 1$.

For example, let $v \in S_1, S_2$ and probability of picking S_i is $P[S_i] = \frac{1}{2}$.

$$P[v \text{ not covered}] = P[\neg S_1] \cdot P[\neg S_2] = (1 - z_1)(1 - z_2) = \frac{1}{4}$$

Then the probability of being covered is

$$P[v \text{ covered}] = 1 - \frac{1}{4} = \frac{3}{4}$$

Lemma (Probability of Covering an Element)

In a sequence of k independent experiments, in which the ith experiment has success probability p_i and

$$\sum_{i=1}^{k} p_i \geq 1$$

then there is a probability $\geq 1 - 1/e$ that at least one experiment is successful.

Proof. Probability that we fail is $(1 - p_1) \cdots (1 - p_k)$. Note that $1 - x \leq e^{-x}$ so $(1 - p_1) \cdots (1 - p_k) \leq e^{-\sum p_i} \leq e^{-1}$.

11.4 Randomized Rounding

Input: $z = (z_1, \ldots, z_n) \in [0,1]^n$ such that z is a solution to our LP
Output: set cover for U
$I = \emptyset$
while there is $v \in U$ uncovered **do**
    for $i = 1$ to n **do**
        $I = I \cup \{i\}$ with probability z_i
return I

Lemma (Probability Decay)

Let $t \in \mathbb{N}$. The probability that the for loop will be executed more than $\ln(|U|) + t$ times is at most e^{-t}.

Proof. The probability that j is not covered is at most $e^{-(t+\log m)}$ by previous lemma. By union bound on all m elements is equal to $\frac{1}{m}e^{-t}$ which is at most e^{-t}.

11.4.1 Cost of Rounding

We now know will cover with high probability, so we just need to bound the cost of the solution. At each iteration of the for loop, our expected cover weight is

$$\sum_{i=1}^{n} w_i z_i$$

After t iterations of the for loop, the expected weight is

$$\omega = t \sum_{i=1}^{n} w_i z_i$$

By Markov,

$$P[X \geq 2\mathbb{E}[X]] \leq \frac{1}{2}$$

Lemma (Cost of Rounding)

Given z optimal for the LP, our randomized rounding outputs, with probability ≥ 0.45, a feasible solution to set cover with $\leq 2(\ln |U| + 3) \cdot OPT(\text{ILP})$ cost.

Proof. Let $t = \ln(|U|) + 3$. There is a probability at most $e^{-3} < 0.05$ that the while loop runs more than t steps. After t steps, the expected weight is

$$\omega = t \sum w_i z_i \leq tOPT(\text{ILP})$$

Markov's inequality shows that probability that our solution has weight $\geq 2\omega$ is $\leq \frac{1}{2}$. By union bound, with probability ≤ 0.55, either run for more than t times, or our solution has weight $\geq 2\omega$. Thus, with probability ≥ 0.45, we stop at t iterations and construct a solution to set cover with cost $\leq 2tOPT(\text{ILP})$.

Chapter 12

Semidefinite Programming Relaxations and Rounding

Definition: Quadratic Program (QP)

$$\begin{aligned} \min \quad & g(x) \\ \text{s.t.} \quad & q_i(x) \geq 0 \end{aligned}$$

where $g(x)$ and each $q_i(x)$ are quadratic functions on x.

QPs can formulate optimization problems but can capture NP-hard problems. In an ILP, if $x \in \{0, 1\}$, then the quadratic constraint is $x(x - 1) = 0$.

We can relax QPs to SDPs to get better approximation algorithms.

12.1 Maximum Cut

Maximum Cut Problem

Given a graph $G = (V, E)$, find a cut $S \subseteq V$ of maximum size.

The ILP is

$$\begin{aligned} \max \quad & \sum_{e \in E} w_e z_e \\ \text{s.t.} \quad & x_u + x_v \geq z_e, \ \forall e = \{u, v\} \in E \\ & 2 - x_u - x_v \geq z_e, \ \forall e = \{u, v\} \in E \\ & x_v \in \{0, 1\}, \ \forall v \in V \end{aligned}$$

To get efficient exact or approximate algorithms for problems, the following strategy is useful:

1. Formulate the optimization problem as a quadratic program.
2. Derive the SDP from the QP by going to higher dimensions and imposing the PSD constraint; this is called an SDP relaxation.
3. We are still maximizing the same objective function but over a potentially larger set of solutions, thus,
$$OPT(SDP) \geq OPT(QP)$$
4. Solve the SDP approximately using an efficient algorithm.
   - If the solution to SDP is integral and one-dimensional, then it is a solution to QP and we are done.
   - If the solution has higher dimension, then we have to devise a rounding procedure that transforms it to an integral and one-dimensional solution. The rounded SDP solution $\geq c \cdot OPT(QP)$.

Suppose $\sum_{e \in E} = 1$ in the problem.

Proposition

$OPT(ILP) = 1$ if and only if G is bipartite.

Proposition

$OPT(ILP) \geq \frac{1}{2}$.

Proof. $z_e = 1$ with probability $\frac{1}{2}$ and $z_e = 0$ with probability $\frac{1}{2}$.

$$\begin{aligned}
\mathbb{E}_{x_u \sim \{0,1\}} \left[\left| E(S, \overline{S}) \right| \right] &= \sum_{e \in E} w_e \mathbb{E}[z_e] \\
&= \sum_{e \in E} w_e \cdot \frac{1}{2} \\
&= \frac{1}{2}
\end{aligned}$$

Proposition

If G is a complete graph, then $OPT = \frac{1}{2} + \frac{1}{2(n-1)}$.

The LP relaxation:

$$\begin{aligned} \max \quad & \sum_{e \in E} w_e z_e \\ \text{s.t.} \quad & x_u + x_v \geq z_e, \ \forall e = \{u, v\} \in E \\ & 2 - x_u - x_v \geq z_e, \ \forall e = \{u, v\} \in E \\ & 0 \leq x_v \leq 1, \ \forall v \in V \\ & 0 \leq z_e \leq 1, \ \forall e \in E \end{aligned}$$

Setting $x_v = \frac{1}{2}, z_e = 1$, we always get $OPT(LP) = 1$, thus this is not helpful.

QP: Consider $1 - (x_u x_v + (1 - x_u)(1 - x_v))$, this is 1 if $x_u \neq x_v$ and 0 if $x_u = x_v$ which is selecting an edge when x_u is in the other side of the cut to x_v.

$$\begin{aligned} \max \quad & \sum_{e=\{u,v\} \in E} w_e(-2x_u x_v + x_u + x_v) \\ \text{s.t.} \quad & x_v(x_v - 1) = 0, \ \forall v \in V \end{aligned}$$

If we instead choose that $x_v = \{-1, 1\}$ instead of $\{0, 1\}$, we can get a better looking program:

$$\begin{aligned} \max \quad & \sum_{e=\{u,v\} \in E} \frac{w_e}{2}(1 - x_u x_v) \\ \text{s.t.} \quad & x_v^2 = 1, \ \forall v \in V \end{aligned}$$

We can transform the QP into higher dimension relaxation with variable $y_v \in \mathbb{R}^d$ and get the SDP relaxation (Delorme, Poljak 1993):

$$\begin{aligned} \max \quad & \sum_{e=\{u,v\} \in E} \frac{w_e}{2}(1 - \langle y_u, y_v \rangle) \\ \text{s.t.} \quad & \|y_v\|^2 = 1, \ \forall v \in V \end{aligned}$$

The SDP is

$$\begin{aligned} \max \quad & \sum_{\{u,v\} \in E} w_{uv} \left(\frac{1 - X_{uv}}{2} \right) \\ \text{s.t.} \quad & X_{uv} = 1 \\ & X \preceq 0 \end{aligned}$$

where $X = Y^T Y$ and Y is a matrix where each column is y_i.

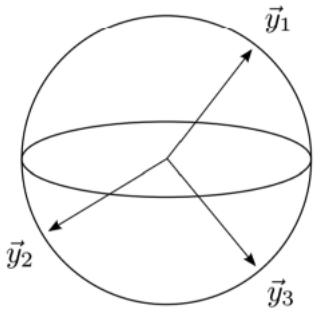
Note that the constraints form a unit sphere in $\mathbb{R}^d$.

12.1.1 SDP Explanation

Let $\gamma_{uv} = y_u^T y_v = \cos(y_u, y_v)$. For any edge, we want $\gamma_{uv} \approx -1$ as this maximizes the weight. Geometrically, we want vertices from our max-cut S to be as far away from $\overline{S}$ as possible. If all y_v's are in one-dimensional space, then we get original quadratic program

$$OPT(SDP) \geq \text{Weight of max-cut}$$

Figure 12.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.



Example: Consider the complete graph $G = K_3$ with equal weights on edges. Embed $y_1, y_2, y_3 \in \mathbb{R}^2$ in a unit circle at 120° apart. We get $OPT_{SDP}(K_3) = \frac{1}{3} \cdot \frac{1}{2} \cdot 3(1 - \cos(120°)) = \frac{1}{6} \cdot 3 \cdot \frac{3}{2} = \frac{3}{4}$ and $OPT_{\text{max-cut}}(K_3) = \frac{2}{3}$. So we get approximation $\frac{8}{9}$ which is better than the LP relaxation.

12.1.2 Rounding

Let $z_u \in \mathbb{R}^n$ be an optimal solution to the SDP. To convert it into a cut, we need to pick sides.

Theorem (Goemans, Williamson 1994)

Choose a random hyperplane through the origin of the unit sphere.

We choose a normal vector $g \in \mathbb{R}^n$ from a Gaussian distribution and set $x_u = sign(\langle g, z_u \rangle) = sign(g^T z_u)$ as the solution.

We can pick a random hyperplane through the origin in polynomial time by sampling a vector $g = (g_1, \ldots, g_n)$ by drawing $g_i \in \mathcal{N}(0, 1)$. If g' is the projection of g onto a two dimensional plane, then $g' / \|g'\|_2$ is uniformly distributed over the unit circle in this plane.

Analysis

The probability that the edge $\{u, v\}$ crosses the cut is the same as the probability that z_u, z_v fall in different sides of the hyperplane, i.e. $P[\{u, v\} \text{ crosses cut}] = P[g \text{ splits } z_u, z_v]$.

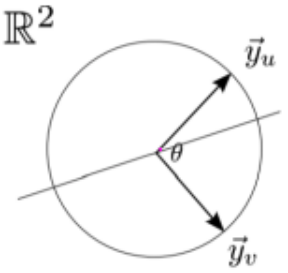
If we look at the problem in the plane:

The probability of splitting z_u, z_v is

$$P[\{u, v\} \text{ crosses cut}] = \frac{\theta}{\pi} = \frac{\cos^{-1}(z_u^T z_v)}{\pi} = \frac{\cos^{-1}(\gamma_{uv})}{\pi}$$

Let X be the random variable for value of the cut which is $X = \sum_{\{u,v\} \in E} w_{uv} X_{uv}$, and the

Figure 12.2: Plane of two vectors being cut by hyperplane.



expected value of the cut is

$$\mathbb{E}[\text{value of cut}] = \sum_{\{u,v\}\in E} w_{uv}\mathbb{E}[X_{uv}] = \sum_{\{u,v\}\in E} \frac{\cos^{-1}(\gamma_{uv})}{\pi}$$

Recall that

$$OPT_{SDP} = \sum_{\{u,v\}\in E} \frac{1}{2} w_{uv}(1 - z_u^T z_v) = \sum_{\{u,v\}\in E} \frac{1}{2} w_{uv}(1 - \gamma_{uv})$$

If we find α such that

$$\frac{\cos^{-1}(\gamma_{uv})}{\pi} \geq \alpha \frac{1-\gamma_{uv}}{2}, \ \forall \gamma_{uv} \in [-1, 1]$$

Then we have an α-approximation algorithm. For $x \in [-1, 1]$, we have

$$\frac{\cos^{-1}(x)}{\pi} \geq 0.878 \cdot \frac{1-x}{2}$$

The **Random Hyperplane Cut algorithm** is a rounding procedure for SDPs if the constraints form a unit circle.

Chapter 13

Hardness of Approximation

Definition: α-Approximation Algorithm

For $\alpha \geq 1$, an algorithm is α-approximate for a minimization (maximization) problem if on every input instance, the algorithm finds a solution with cost $\leq \alpha \cdot OPT$ ($\geq \frac{1}{\alpha} \cdot OPT$).

For some problems, it is possible to prove that even the design of approximation algorithms for certain values of α is impossible, unless $P = NP$ (we would have an exact algorithm).

Definition: Reduction

To prove a combinatorial problem $\mathcal{C}$ is NP-hard, pick an NP-complete combinatorial problem L and show a reduction such that

- maps every YES instance of L to a YES instance of $\mathcal{C}$.
- maps every NO instance of L to a NO instance of $\mathcal{C}$.

Definition: Approximation Reduction

To prove a combinatorial problem $\mathcal{C}$ is NP-hard, pick an NP-complete combinatorial problem L and show a reduction such that

- maps every YES instance of L to a YES instance of $\mathcal{C}$.
- maps every NO instance of L to a VERY-MUCH-NO instance of $\mathcal{C}$.

13.1 Traveling Salesman Problem

Traveling Salesman Problem (TSP)

Input: set of points $\mathcal{X}$ and a symmetric distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$.
For any path $p_0 p_1, \ldots, p_t$ in $\mathcal{X}$, the length of the path is the sum of distances travelled $\sum_{i=1}^{t-1} d(p_i, p_{i+1})$.
Output: find a cycle that reaches all points in $\mathcal{X}$ of shortest length.

Definition: General TSP-NR

TSP, but visit every vertex exactly once.

If $P \neq NP$, then there is no poly-time constant-approximation algorithm for General TSP-NR. In general, if there is any function $f : \mathbb{N} \to \mathbb{N}$ such that $r(n)$ is computable in polynomial time, then it is hard to $r(n)$-approximate General TSP-NR, if we assume that $P \neq NP$.

Theorem

If there is an algorithm M which solves TSP-NR with α-approximation, then $P = NP$.

Definition: Hamiltonian Cycle Problem

Given a graph $G = (V, E)$, decide whether there exists a cycle C which passes through every vertex at most once.

Reduction of TSP to Hamiltonian Cycle Problem: If we had an algorithm M which solved the α-approximate TSP-NR problem, then from G, construct the weighted graph $H = (V, F, w)$ such that H is the complete graph on V with

$$w(u, v) = \begin{cases} 1 & \text{if } \{u, v\} \in E \\ (1 + \alpha) \cdot |V| & \text{if } \{u, v\} \notin E \end{cases}$$

If G has a Hamiltonian cycle, then OPT for TSP is of value $\leq |V|$. If G has no Hamiltonian cycle, then OPT for TSP must use an edge not in E, thus value is $\geq (1 + \alpha) \cdot |V|$. Thus, M on input H will output a Hamiltonian cycle of G, if G has one, or it will output a solution with value $\geq (1 + \alpha) \cdot |V|$.

If G has a Hamiltonian cycle, then H has a cycle of length $|V|$. If G has no Hamiltonian cycle, then any solution to TSP-NR(H) has length $\geq (1 + \alpha) \cdot |V|$.

For TSP-NR(H, n), if the algorithm finds a solution $\leq \alpha |V|$ (α-approximation), then G has a Hamiltonian cycle, and if $\geq (1 + \alpha) \cdot |V|$, then G has no Hamiltonian cycle. However, these two conditions combined shows that TSP-NR does not have an α-approximation.

13.2 Complexity Classes

Definition: Non-deterministic Polynomial (NP)

Set of languages $L \subseteq \{0,1\}^*$ such that there exists a poly-time Turing Machine V, such that

$$x \in L \iff \exists w \in \{0,1\}^{poly(|x|)} \text{ s.t. } V(x,w) = 1$$

Definition: Bounded Probabilistic Polynomial (BPP)

Set of languages $L \subseteq \{0,1\}^*$ such that there exists a poly-time Turing Machine M, such that for every $x \in \{0,1\}^*$, we have

$$P_{R \in \{0,1\}^{poly(|x|)}}[M(x,R) = L(x)] \geq \frac{2}{3}$$

Definition: Randomized Polynomial (RP)

Set of languages $L \subseteq \{0,1\}^*$ such that there exists a poly-time Turing Machine M, such that

$$\begin{aligned} x \in L &\implies P_{R \in \{0,1\}^{poly(|x|)}}[M(x,R) = 1] \geq \frac{2}{3} \\ x \notin L &\implies P_{R \in \{0,1\}^{poly(|x|)}}[M(x,R) = 1] = 0 \end{aligned}$$

Definition: Complement Randomized Polynomial (co-RP)

Languages $L \subseteq \{0,1\}^*$ such that $\overline{L} \in RP$.

13.3 Proof Systems

Definition: Proof System

A prover and a verifier agree on the following:

- The prover must provide proofs in a certain format.
- The verifier can use algorithms from a certain complexity class for verification.

A statement is given to both the prover and verifier. The prover writes down a proof of the statement, while the verifier uses an algorithm of their choice to check the statement and proof, and accepts/rejects accordingly.

NP as a proof system:

- $L \subseteq \{0,1\}^n$ is the language, verifier can use any poly-time Turing Machine.
- Given an element x, the prover gives a proof (also known as a *witness*) $w \in \{0,1\}^{poly(|x|)}$.
- Verifier picks a poly-time Turing Machine V and outputs TRUE if $V(x,w) = 1$ and FALSE otherwise.

Definition: Completeness

True statements have a proof in the system.

Definition: Soundness

False statements do not have a proof in the system.

NP:

- Completeness: $x \in L \implies \exists w \in \{0,1\}^{poly(|x|)}$ such that $V(x,w) = 1$.
- Soundness: $x \notin L \implies \forall w \in \{0,1\}^{poly(|x|)}$ we have $V(x,w) = 0$.

13.4 Probabilistic Proof Systems

Definition: Probabilistic Proof System

In a probabilistic proof system, the verifier has a randomized algorithm V for which

- Given language L (language of correct statements).
- $x \in L \implies$ there exists proof w such that $P[V^w(x) = 1] = 1$.
- $x \notin L \implies$ for any proof w, we have $P[V^w(x) = 1] \leq \frac{1}{2}$.

Definition: Probabilistic Checkable Proofs (PCPs)

The class of Probabilistic Checkable Proofs consists of languages L that have a randomized poly-time verifier V such that

- $x \in L \implies$ there exists proof w such that $P[V^w(x) = 1] = 1$.
- $x \notin L \implies$ for any proof w, we have $P[V^w(x) = 1] \leq \frac{1}{2}$.

$PCP[r(n), q(n)]$ consists of $L \in PCP$ such that on inputs x of length n, it uses $O(r(n))$ random bits and examines $O(q(n))$ bits of a proof w.

Note that n does not depend on w, only on x.

$PCP[0, \text{poly}(n)] = NP$ since we use 0 randomness and polynomial size prover.

Definition: Interactive Proof Systems

The class IP consists of all languages L that have an interactive proof system (P, V) where

- the verifier V is a randomized, polynomial time algorithm.
- there is an honest prover P (who can be all powerful).
- for any $x \in \{0,1\}^*$
  - $x \in L \implies$ for an honest prover P, the proof Π_P satisfies
  $$P[V^{\Pi_P}(x) = 1] = 1$$
  - $x \notin L \implies$ for any prover P', the proof $\Pi_{P'}$ satisfies
  $$P[V^{\Pi_{P'}}(x) = 1] \leq \frac{1}{2}$$

Theorem (PCP Theorem AS'98, ALMSS'98)

$$PCP[\log n, 1] = NP$$

13.5 Approximability of Max 3SAT

Definition: Max 3SAT

Input: a 3CNF formula φ on Boolean variables $x_1, \ldots, x_n$ and m clauses.
Output: the maximum number of clauses of φ which can be simultaneously satisfied.

Theorem

The PCP theorem implies that there is an $\varepsilon > 0$ such that there is no polynomial time $(1+\varepsilon)$-approximation algorithm for Max 3SAT, unless $P = NP$.
Moreover, if Max 3SAT is hard to approximate within a factor of $(1+\varepsilon)$, then the PCP theorem holds.

The theorem states that the PCP theorem and hardness of approximation of Max 3SAT are equivalent.

Proof. Let us assume the PCP theorem holds. Let $L \in PCP[\log n, 1]$ be an NP-complete problem. Let V be the $(O(\log n), q) = (\gamma \log n, q)$ verifier for L, where q is a constant.

Now we reduce L to Max 3SAT which has a gap. Given an instance x of L, we construct a 3-conjunctive normal form (CNF) formula φ_x with m clauses such that for some ε such that $x \in L \implies \varphi_x$ is satisfiable and $x \notin L \implies$ no assignment satisfies more than $(1-\varepsilon)m$

clauses of φ_x.

Enumerate all random inputs R for the verifier V. The length of each random string is $O(\log n)$,s o the number of such random inputs is $2^{\gamma \log n} = n^\gamma = \text{poly}(n)$. For each R, V chooses q positions $i_1^R, \ldots, i_q^R$ and a Boolean function $f_R : \{0,1\}^q \to \{0,1\}$ and accepts if and only if $f_R(w_{i_1^R}, \ldots, w_{i_q^R}\} = 1$.

Simulate the computation f_R of the verifier for different random inputs R and witnesses w as a Boolean formula. A CNF formula has size 2^q and converting to 3-CNF, we have size $q2^q$. For all R, $n^\gamma q 2^q$ clauses.

If $x \in L$, then there is a witness w such that $V(x, w)$ accepts for every random string R. φ_x is satisfiable.

If $x \notin L$, then the verifier returns NO for half of the random strings R. For each such string, at least one of its clauses fail. Thus, at least $\varepsilon = \frac{n^\gamma/2}{n^\gamma q 2^q} = \frac{1}{2q2^q}$ of the clauses of φ_x fails.

Part V

Online Algorithms

Chapter 14

Online Algorithms and Paging

Definition: Competitive Analysis

Measures performance of an algorithm against the best algorithm that could see into the future.
It is a worst-case analysis.

Given an input sequence $s = s_1, s_2, \ldots, s_n$ of events. Let $C_{opt}(s)$ be the minimum cost that any algorithm could achieve for s. Let $C_A(s)$ be the cost of the online algorithm on s.

Definition: Deterministic Competitive Ratio

A deterministic online algorithm A is k-competitive (has competitive ratio k), if for all inputs s, we have

$$C_A(s) \leq kC_{opt}(s) + O(1)$$

Definition: Randomized Competitive Ratio

A randomized online algorithm A is k-competitive (has competitive ratio k), if for all inputs s, we have

$$\mathbb{E}[C_A(s)] \leq kC_{opt}(s)$$

14.1 Ski Rental Problem

Ski Rental Problem

Decide whether to buy all the equipment or rent the equipment at the resort.

Suppose buying costs \$1000 dollars and renting at the resort cost \$100 per day.

Buying or renting depends on how many times you go skiing. If you go skiing 9 times or less, then it is better to rent. If you go skiing at least 11 times, then it is better to buy. If

you go exactly 10 times, then it does not matter.

This is an online algorithm because the algorithm decides when to buy, knowing only that we have gone skiing t times.

Any deterministic algorithm will rent $t-1$ times and buy on the tth time. Thus, the cost is $100(t-1)+1000$. If you go $s < t$ times, then the cost is $100s$ and the deterministic algorithm will buy at the beginning.

If $t \leq 9$, then rent and when $t = 10$, we buy. This is a 1.9-competitive algorithm.

Analysis: If $t \leq 9$, then the best strategy is to rent, so the cost is

$$\frac{C_A}{C_{opt}} = \frac{100t}{100t} = 1$$

If $t \geq 10$, we buy at the 10th time, so cost is

$$\frac{C_A}{C_{opt}} = \frac{100(9) + 1000}{1000} = 1.9$$

14.2 Dating Problem

Dating Problem

There are n people you are interested in dating and would like to date the best person. Maximize the probability of dating the best person.

You could go out with all of them at the same time, but this is not possible. Thus, we have to go with one at a time. This is an online setting.

Consider the following algorithm:

1. Assume you ranked all the people from $1, \ldots, n$.
2. Pick random order of n people: π.
3. Go out with n/e of the people and reject them.
4. After the first n/e dates, you will decide to settle if the person you found is better than anyone else you have dated before.

The algorithm picks the best person with probability $\approx 1/e$.

The more general algorithm is given a time t, go on t dates and from date $t+1$ onward, you decide to settle with a person who is better than the previous ones.

Suppose we pick a person at time k, then

$$P_k = P[\pi(k) = 1 \text{ and pick person at time } k]$$

Then our final success probability will be $P = \sum_{k>t}^{n} P_k$.

If $\pi(k) = 1$, then $1 - P_k$ is the probability we picked a person between $[t+1, k-1]$, which means someone in this range is better than the first t people.

$$P_k = P[\pi(k) = 1 \cap \min\{\pi(1), \ldots, \pi(k-1)\} \in \{\pi(1), \ldots, \pi(t)\}] = \frac{(n-1)!}{n!} \cdot \frac{t}{k-1} = \frac{1}{n} \cdot \frac{t}{k-1}$$

We get

$$P = \sum_{k>t}^{n} \frac{1}{n} \cdot \frac{t}{k-1} = \frac{t}{n} \sum_{k>t}^{n} \frac{1}{k-1} \approx \frac{t}{n} \cdot (\ln n - \ln t) = \frac{t}{n} \ln \frac{n}{t}$$

Optimizing the above and we set $t = n/e$, which gives $1/e$ probability.

To make it competitive

- Minimize the rank.
- The previous algorithm would either pick the best person or the last person, if $\pi(k) = 1$ is within the first t people.
- With constant probability, the rank of the last person is $\Omega(n)$, which is within the bottom percentile.
- The expected rank is $\Omega(n)$.

This is not good. There is an algorithm that picks the person of average rank $O(1)$, so $O(1)$-competitive.

Based on computing time steps $t_0 \leq t_1 \leq \cdots$ and between time steps t_k and t_{k+1}, we will be willing to pick the person who is $\leq k+1$ best from the current seen list.

14.3 Online Paging Problem

Computer memory is hierarchical: cache $\to$ L1 $\to$ L2 $\to$ main memory. The memory can be modelled in the following way:

- Each layer of memory is an array with a certain number of pages.
- A page stores the content of the item and its location in main memory.
- When we get a request, we first lookup in cache, then L1, then L2, then main memory.
- If the request is in the cache, we have a hit (which takes negligible time).
- Otherwise, we have a miss and need to fetch it in slower memory.

- In the negligible extra time, we copy the new data and location to the cache.
- If the cache is full, we must delete an old entry before copying in the new data and location.

Paging Problem

Which entry in the cache do we delete when the cache is full?

The cost function is the number of cache misses. Assume that there is only cache and main memory.

14.3.1 Heuristics

Definition: Least Recently Used (LRU)

Delete page in cache whose most recent request happened furthest in the past.

Definition: Random

Delete random page.

Definition: First-In First-Out (FIFO)

Delete page that has been in cache the longest.

Definition: Least Frequently Used (LFU)

Delete page in cache which has requested least often.

- LRU is k-competitive.
- Random is k-competitive.
- FIFO is k-competitive.
- LFU is not competitive.

Theorem

For a cache of size k, LRU is k-competitive.

Upper bound: divide input sequence into phases. The first phase starts immediately after the algorithm first faults and ends right after the algorithm faults k more times. Second phase starts after first phase and ends after k more faults.

We prove the optimal algorithm faults at least once per phase. This will give $C_A \leq kC_{opt}$.
Proof. Denote $s^{(i)}$ be the ith phase. Then

$$C_{LRU}(s) = \sum_{i \geq 1} C_{LRU}(s^{(i)}) \leq \sum_{i \geq 1} kC_{opt}(s^{(i)}) = kC_{opt}(s)$$

Analysis: Same page faulted twice in one phase. Let $s_1, \ldots, s_t$ be a phase and $p = s_1 = s_t$. In order for page p to be faulted twice, then at least $k-1$ of the pages in cache have to be queried in $s_2, \ldots, s_{t-1}$, then p would be least recently used so on s_t, p faults. Thus, you would need to query at least $k+1$ distinct elements. By the pigeonhole principle, OPT must fault at least once.

Each page faulted once in a phase. Let $s_1, \ldots, s_t$ be a phase. Let page p comes before s_1. In the beginning of each phase, cache of OPT and cache of A must have p. Since OPT and A has a common page, then OPT must have faulted, so $t = k$ distinct pages in the phase. Since there are $k-1$ spots (not p) and we have k distinct pages, by pigeonhole principle, we have 1 fault.

Theorem

Any deterministic algorithm for paging with k pages is at least k-competitive.

Proof. Proof by trolling. Use $k+1$ pages and let A be paging algorithm. At each step, request a page that A does not have. A faults every single time.

The offline algorithm: on a cache miss, delete page which is requested furthest in the future. When the offline algorithm deletes a page, the next delete happens after at least k steps.

Chapter 15

Multiplicative Weights Update Method

Learning From Experts: We want to invest money on the stock market, and our objective is to get rich, but we do not know much about stock markets. We also have access to n experts who know about the stock market.

- Each morning, before the market opens, experts predict whether the price of a stock will go up or down.
- By the time the market closes, we can check who was right or wrong. Experts who were right earn 1 dollar and those who were wrong lose 1 dollar.

If we follow the good expert, then we would make a lot of money. We want to to do as well as the best expert.

If we are trading for T days, guessing correctly in expectation is $T/2$ times and it also is optimal.

Say we knew that there was one expert which will be right every time. At each trading day, take the majority vote of the opinions of the experts. If we made the right trade, do nothing. If the trade was bade, at the end of the trading day, discard all experts that made a mistake that day.

Every time we made a bad trade, we discard half of the experts. After $\log n$ bad trades, only the expert who is always right will remain. Total money we made is $\geq T - \log n$ and total money best expert made is T.

15.1 Multiplicative Weights Update Algorithm

Whenever an expert makes a mistake, consider their opinions with less importance. Let $w_t : [n] \to \mathbb{R}_+$ be a function from each expert to the nonnegative reals and $0 < \varepsilon < 1/2$.

$w_t(i)$ is the weight of expert i at time t.

In the beginning, every expert has weight $w_1(i) = 1$. If an expert makes a mistake on day t, make $w_{t+1}(i) = w_t(i) \cdot (1 - \varepsilon)$. Each trading day, choose to trade based on weighted majority of the decisions of the experts.

Multiplicative Weights Update Algorithm

1. Setup: a binary decision and we have access to n experts, index by $[n]$. At each time step t, expert takes decision $d_t(i) \in \{-1, +1\}$ and a parameter $0 < \varepsilon < 1/2$.
2. $w_t : [n] \to \mathbb{R}_+$ be weight function. In the beginning, every expert has weight $w_1(i) = 1$.
3. At each time step $t = 1, \ldots, T$:
   (a) Make decision based on weighted majority

$$\begin{cases} +1 & \text{if } \sum_{i=1}^{n} w_t(i) \cdot d_t(i) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

   (b) If an expert makes a mistake at time t, make

$$w_{t+1}(i) = w_t(i) \cdot (1 - \varepsilon)$$

15.2 Analysis

Theorem (Multiplicative Weights Update)

Let M_t be the number of mistakes that our algorithm makes until time t, and let $m_t(i)$ be the number of mistakes that expert i made until time t.
Then, for any expert $i \in [n]$, we have

$$M_t \leq 2(1 + \varepsilon)m_t(i) + \frac{2 \log n}{\varepsilon}$$

Proof. We have the potential function

$$\Phi_t = \sum_{i=1}^{n} w_t(i)$$

Intuition: If we make a mistake, Φ_{t+1} decreases by a multiplicative factor with respect to Φ_t. Φ_t is monotonically decreasing. Initially $\Phi_1 = n$ and $\Phi_t \geq 0$ for all t. If we made a mistake,

at least half the weight was on the wrong answer. Thus

$$\Phi_{t+1} = \sum_{i \text{ right}} w_t(i) + (1-\varepsilon) \cdot \sum_{j \text{ wrong}} w_t(j) = \Phi_t - \varepsilon \underbrace{\sum_{i \text{ wrong}} w_t(i)}_{\geq \Phi_t/2} \leq \left(1 - \frac{\varepsilon}{2}\right) \Phi_t$$

and also $\Phi_t = \sum_{i=1}^{n} w_t(i) \geq w_t(i) = (1-\varepsilon)^{m_t(i)}$,

$$(1-\varepsilon)^{m_t(i)} \Phi_t \leq \Phi_1 \left(1 - \frac{\varepsilon}{2}\right)^{M_t} = n \left(1 - \frac{\varepsilon}{2}\right)^{M_t}$$

Then we get

$$m_t(i) \log(1-\varepsilon) \leq \log n + M_t \log\left(1 - \frac{\varepsilon}{2}\right)$$

And for $x \in (0, 1/2)$,

$$-x - x^2 \leq \log(1-x) \leq -x$$

$$(-\varepsilon - \varepsilon^2) m_t(i) \leq \log n - M_t \frac{\varepsilon}{2} \implies M_t \leq \frac{2 \log n}{\varepsilon} + 2(1+\varepsilon) m_t(i)$$

Multiplicative Weights Update – General

1. Setup: have access to n experts, index by $[n]$. At each time step t, each expert will guess a value $m_t(i) \in [-w, +w]$ (mistake is $+w$). Cost of ith expert answer at time t is $m_t(i)$. Parameter $0 < \varepsilon < 1/2$.

2. $p_t : [n] \to \mathbb{R}_+$ be weight function (normalized to sum to 1). $p_t(i)$ is the weight of expert i at time t and in the beginning, every expert has weight $p_1(i) = 1/n$. The total cost is $\sum_t \langle p_t, m_t \rangle$.

   This implies the update rule is

   $$w_{t+1} = \left(1 - \varepsilon \cdot \frac{m_t(i)}{w}\right) w_t(i), p_{t+1}(i) = \frac{w_{t+1}(i)}{\Phi_{t+1}}$$

3. Minimize total cost: $\sum_{t=1}^{T} \langle p_t, m_t \rangle$.

Theorem (Multiplicative Weights Update)

With the setup, after T rounds, for any expert $i \in [n]$, we have

$$\sum_{t=1}^{T} \langle p_t, m_t \rangle \leq \sum_{t=1}^{T} m_t(i) + \varepsilon \sum_{t=1}^{T} |m_t(i)| + \frac{w \ln n}{\varepsilon}$$

15.3 Solving Linear Programs

Assume we are given a feasibility LP $Ax \geq b, x \geq 0$, the optimization version reduces to feasibility version by binary search.

We can think of each inequality $A_i x \geq b_i$ as an expert. Each constraint would like to be the hardest constraint, i.e. the one that is violated the most by current proposed solution $x^{(t)}$. Precisely, the cost of the ith constraint $A_i x - b_i$.

Definition: Oracle

Let $A \in \mathbb{R}^{m \times n}$. $\mathcal{O}$ is an oracle of width w for A if given a linear constraint

$$pAx \geq pb, x \geq 0$$

$\mathcal{O}(p)$ will return $y \geq 0$ such that

$$|A_i y - b_i| \leq w, \ \forall i \in [m]$$

We would like to maximize

$$\min_{1 \leq i \leq m} A_i x - b_i$$

The multiplicative weights update provides a way to combine all constraints into one constraint. It finds a probability distribution over experts (normalized weights) which in our case are inequalities. Thus, we only have to deal with the constraint $p^{(t)} Ax \geq p^{(t)} b$, where

$$p^{(t)} = \frac{1}{\sum_i w_t(i)} \cdot (w_t(1), \ldots, w_t(n))$$

MWU shows over the long run, the total violation of weighted constraints will be close to total violation of worst violated constraint.

MWU shows that over the long run, for any inequality $i \in [m]$:

$$\sum_{t=1}^{T} \left\langle p^{(t)}, Ax^{(t)} - b \right\rangle < \frac{w \log m}{\varepsilon} + \sum_{t=1}^{T} (A_i x^{(t)} - b_i) + \varepsilon \sum_{t=1}^{T} \left| A_i x^{(t)} - b_i \right|$$

We can return the solution

$$z = \frac{1}{T} \sum_{t=1}^{T} x^{(t)}$$

Plug it back in, we have

$$-\varepsilon w - \frac{w \log m}{T \varepsilon} \leq A_i z - b_i$$

If we set $T = \frac{w^2 \log m}{\delta^2}$ and $\varepsilon = \frac{\delta}{2w}$, and the above becomes $A_i z - b_i \geq -\delta$.

Theorem (Multiplicative Weights Update)

Let $\delta > 0$ and suppose we are given an oracle with weight w for A. The MWU algorithm either finds a solution $y \geq 0$ such that

$$A_i y \geq b_i - \delta, \ \forall i \in [m]$$

or concludes that the system is infeasible (and outputs a dual solution).
The algorithm makes $O(w^2 \log m/\delta^2)$ oracle calls.

Chapter 16

Streaming

Definition: Basic Data Stream Model

In the data stream model:

- A stream of elements $a_1, \ldots, a_N$ each from a known alphabet Σ.
  Each element of Σ takes b bits to represent.
- Basic operations (comparison, arithmetic, bitwise) take $\Theta(1)$ time.
- Single or small number of passes over data.
- Bounded storage (typically $\log^c(N)$ for $c = O(1)$ or N^α for some $0 < \alpha < 1$).
- Allowed to use randomness.
- Usually want approximate answers.

Our goal is to minimize space complexity and processing time.

Examples of streaming problems:

Sum of Elements

Input stream: $a_1, \ldots, a_N$ be integers from the set $[-2^b + 1, 2^b - 1]$.
Task: maintain the current sum of elements seen so far.

Median

Input stream: $a_1, \ldots, a_N$ be integers from the set $[-2^b + 1, 2^b - 1]$.
Task: maintain the current median of elements seen so far.

16.1 Majority Element

Heavy hitters with $\varepsilon = 1/2$.

At time t, we will maintain a set S_t which contains the element that has appeared at least $\frac{N}{2}$ times, if any.

Setup: heavy hitters with $\varepsilon = \frac{1}{2}$
$S_0 = \emptyset$
$c = 0$
while a_t arrives **do**
    if $c = 0$ **then**
        $S_t = \{a_t\}$
        $c = 1$
    else
        if $a_t \in S_{t-1}$ **then**
            $c = c + 1$
        else
            $c = c - 1$ and discard a_t
return Element in S_N

16.1.1 Analysis

If there is no majority element, we could still output a low hitter. Every time we discard a copy of the majority element, we throw away a different element.

For example, consider the stream $3, 1, 2, 1, 1$, the majority element appears more than half the time, so we cannot throw away all the majority elements.

Used $O(b) + O(\log N)$ space; the set S_t which has one element and the counter is $\log N$.

16.2 Heavy Hitters

Heavy Hitters

Input stream: $a_1, \ldots, a_N$ be integers from the set $[-2^b + 1, 2^b - 1]$, $\varepsilon > 0$.
Task: maintain set of elements that contains elements that have appeared at least ε-fraction of the time (heavy hitters).
Constraint: allowed to output false positives (low hitters), but not allowed to miss any heavy hitter.

$k = \left\lceil \frac{1}{\varepsilon} \right\rceil - 1$
T = array of length k where $T[i]$ can hold $x \in \Sigma = [-2^b + 1, 2^b - 1]$
C = array of length k where $C[i]$ can hold a nonnegative integer
$T[i] = NaN$
$C[i] = 0$ for all $i \in [k]$
while a_t arrives **do**
    if $\exists j \in [k]$ such that $a_t = T[j]$ **then**
        $C[j] = C[j] + 1$
    else if $\exists j \in [k]$ such that $C[j] = 0$ **then**
        $T[j] = a_t$
        $C[j] = 1$
    else
        $C[j] = C[j] - 1$ for all $j \in [k]$
        Discard a_t
return T, C

16.2.1 Analysis

For each element $e \in \Sigma$, let $est(e) = \begin{cases} C[j] & \text{if } e = T[j] \\ 0 & \text{otherwise} \end{cases}$.

Lemma

Let $count(e)$ be the number of occurrences of e in stream up to time N.

$$0 \leq count(e) - est(e) \leq \frac{N}{k+1} \leq \varepsilon N$$

Proof. $count(e) \geq est(e)$ since we never increase $C[j]$ for e unless we see e. If we do not increase $est(e)$ by 1 when we see an update to e, then we decrement k counters and discard current update to e. So we drop $k+1$ distinct stream updates, but there are N updates, so we do not increase $est(e)$ by 1 (when we should) at most $\frac{N}{k+1} \leq \varepsilon N$ times.

At any time N, all heavy hitters e in T. For an ε-heavy hitter e, we have $count(e) > \varepsilon N$. $est(e) \geq count(e) - \varepsilon N > 0$ so $est(e) > 0$ implying $e \in T$.

The space used is $O(k(\log \Sigma + \log N)) = O\left(\frac{1}{\varepsilon} \cdot (b + \log N)\right)$ bits.

16.3 Distinct Elements

Distinct Elements

Input stream: $a_1, \ldots, a_N$ be integers from the set $[0, m-1]$, $m = 2^b$.
Task: maintain current number of distinct elements D seen so far.

We can use a strongly 2-universal hash function $h : [0, m-1] \to [0, m^3]$. We have seen with high probability, there are no collisions.

Suppose there are D distinct elements $b_1, \ldots, b_D$, if the D hash values $h(b_1), \ldots, h(b_D)$ are evenly distributed in $[0, m^3]$, then the tth smallest hash value should be close to $\frac{tm^3}{D}$. If we know that tth smallest value is T, then $T \approx \frac{tm^3}{D}$ implying $D \approx \frac{tm^3}{T}$.

1: Choose a random hash function h from a strongly 2-universal hash family
2: **for** each item a_i in the stream **do**
3: Compute $h(a_i)$
4: Update list that stores t smallest hash values
5: After all data has been read, let T be tth smallest value in data stream
6: **return** $Y = \frac{tm^3}{T}$

16.3.1 Analysis

We not store the whole hash table, only store hash function h and k numbers.

Theorem

Setting $k = O(1/\varepsilon^2)$, $Y = \frac{tm^3}{T}$ satisfies

$$(1-\varepsilon)D \leq Y \leq (1+\varepsilon)D$$

with constant probability.

We want to upper bound $P[Y > (1+\varepsilon) \cdot D]$.

$$Y > (1+\varepsilon)D \implies T < \frac{tm^3}{(1+\varepsilon)D} \leq \frac{(1-\varepsilon/2)tm^3}{D}$$

There are at least t hash values are smaller than the quantity above.

Let the random variable

$$X_i = \begin{cases} 1 & \text{if } h(b_i) \leq \frac{(1-\varepsilon/2)tm^3}{D} \\ 0 & \text{otherwise} \end{cases}$$

So

$$\mathbb{E}[X_i] = P\left[h(b_i) \leq \frac{(1-\varepsilon/2)tm^3}{D}\right] = \frac{(1-\varepsilon/2)t}{D}$$

where each $h(b_i)$ is uniformly at random in $[0, m^3]$. If there are D distinct elements,

$$\mathbb{E}\left[\#\text{ elements with hash value} \leq \frac{(1-\varepsilon/2)tm^3}{D}\right] \leq t(1-\varepsilon/2)$$

but we have assumed we have at least t such elements. We show this cannot happen with high probability. If there are D distinct elements, let $X = \sum_{i=1}^{D} X_i$.

$$\mathbb{E}[X] \leq t(1 - \varepsilon/2)$$

The probability we will see $\geq t$ elements smaller than $\frac{(1-\varepsilon/2)tm^3}{D}$. $Var[X] = \sum_{i=1}^{D} Var[X_i]$ since pairwise independence and $Var[X_i] = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 \leq \mathbb{E}[X_i]$. By Chebyshev's inequality

$$\begin{aligned} P[X > t] &= P[X > t \cdot (1 - \varepsilon/2) + \varepsilon \cdot t/2] \\ &\leq P[|X - \mathbb{E}[X]| > \varepsilon \cdot t/2] \\ &\leq \frac{4Var[X]}{\varepsilon^2 t^2} \\ &\leq \frac{4}{\varepsilon^2 t} \end{aligned}$$

The lower bound $P[Y < (1 - \varepsilon) \cdot D]$ is similar. So

$$\begin{aligned} P[Y > (1 + \varepsilon) \cdot D] &\leq \frac{4}{\varepsilon^2 t} \\ P[Y < (1 - \varepsilon) \cdot D] &\leq \frac{4}{\varepsilon^2 t} \end{aligned}$$

Setting $t = 24/\varepsilon^2$ gives

$$P[(1 - \varepsilon) \cdot D \leq Y \leq (1 + \varepsilon) \cdot D] \geq 1 - \frac{8}{\varepsilon^2 t} = \frac{2}{3}$$

The total space used is $O\left(\frac{1}{\varepsilon^2} \log m\right)$ bits since we stored $O(1/\varepsilon^2)$ hash values each of $\log m$ bits and the hash function only requires $O(\log m)$ bits to store.

The runtime per operation is $O(\log m + 1/\varepsilon^2)$ steps since we compute hash in $O(\log m)$ time and we keep track of $O(1/\varepsilon^2)$ elements and need to update the list, this takes $O(1/\varepsilon^2)$ time.

16.4 Weighted Heavy Hitters

Weighted Heavy Hitters

Input stream: $(a_1, w_1), \ldots, (a_N, w_N)$ tuples of integers from $\Sigma = [-2^b + 1, 2^b - 1]$, parameter $q \in \mathbb{N}$.
Total weight

$$Q = \sum_{t=1}^{N} w_t$$

Total weight of $e \in \Sigma$

$$Q(e) = \sum_{t:a_t=e} w_t$$

Task: find all elements e such that $Q(e) \geq q$.
Constraint: allowed to output low hitters, but now allowed to miss any heavy hitter.

Setup:

- All heavy hitters reported.
- If $Q(e) \leq q - \varepsilon Q$, then e is reported with probability at most δ.
- k, ℓ are parameters to be chosen.
- Pick k hash functions $h_1, \ldots, h_k$ where $h_i : \Sigma \to [0, \ell - 1]$.
- Main $k \cdot \ell$ counters $C_{i,j}$ where each $C_{i,j}$ adds the weight of items that are mapped to jth entry by the ith hash function.

  Start with $C_{i,j} = 0$ for all $1 \leq i \leq k, 1 \leq j \leq \ell$.

1: Given (a_t, w_t) for each $1 \leq i \leq k$, set $C_{i,h_i(a_t)} = C_{i,h_i(a_t)} + w_t$
2: At the end (need to do a second pass), report all elements e with

$$\min_{1 \leq i \leq k} C_{i,h_i(e)} \geq q$$

16.4.1 Analysis

Heavy hitters are always reported as all their counters are large. Need to show if e is not a heavy hitter, with high probability, we will have one counter $C_{i,h_i(e)} < q$.

If $Q(e) \leq q - \varepsilon Q$ and e is reported, $C_{i,h_i(e)} \geq q$. The contribution from e is $Q(e) \leq q - \varepsilon Q$, so the other elements that map to $h_i(e)$ must have contributed $\geq \varepsilon Q$.

Let Z_i be the value of $C_{i,h_i(e)}$ that was added by other elements. h_i is from a 2-universal hash family, so the probability that another element f is mapped to $h_i(e)$ is $\leq \frac{1}{\ell}$. Thus, $\mathbb{E}[Z_i] \leq \frac{Q}{\ell}$. By Markov's inequality

$$P[Z_i \geq \varepsilon Q] \leq \frac{\mathbb{E}[Z]}{\varepsilon Q} \leq \frac{1}{\varepsilon \ell}$$

Since the hash functions h_i are chosen independently,

$$P\left[\min_{1 \leq i \leq k} Z_i \geq \varepsilon Q\right] \leq \left(\frac{1}{\varepsilon \ell}\right)^k$$

Setting $\ell = 2/\varepsilon$ and $k = \log \delta$, we get the probability $\leq \delta$.

The space for counters is $O\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$. The space to store all hash functions and evaluation time is $O(k\ell)$.

Part VI

Symbolic Computation

Chapter 17

Matrix Multiplication & Exponent of Linear Algebra

17.1 Matrix Multiplication

Matrix Multiplication

Input: matrices $A, B \in \mathbb{F}^{n \times n}$.
Output: product $C = AB$.

Naive algorithm computes n matrix vector multiplications, which results in $O(n^3)$ runtime.

Strassen's Algorithm

Suppose $n = 2^k$. Let $A, B, C \in \mathbb{F}^{n \times n}$ such that $C = AB$. Divide them into blocks of size $\frac{n}{2}$:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Define the following matrices:

$$S_1 = A_{21} + A_{22}, S_2 = S_1 - A_{11}, S_3 = A_{11} - A_{21}, S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11}, T_2 = B_{22} - T_1, T_3 = B_{22} - B_{12}, T_4 = T_2 - B_{21}$$

Compute the 7 products:

$$P_1 = A_{11}B_{11}, P_2 = A_{12}B_{21}, P_3 = S_4B_{22}, P_4 = A_{22}T_4, P_5 = S_1T_1, P_6 = S_2T_2, P_7 = S_3T_3$$

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} P_1 + P_2 & P_1 + P_3 + P_5 + P_6 \\ P_1 - P_4 + P_6 + P_7 & P_1 + P_5 + P_6 + P_7 \end{bmatrix}$$

To compute $AB = C$, there were 8 additions for S_i, T_i, 7 multiplications for P_i, and 10

additions for C_{ij}. Therefore, the recurrence is

$$MM(n) \leq 7 \cdot MM(n/2) + 18c(n/2)^2 \implies MM(2^k) \leq 7 \cdot MM(2^{k-1}) + 18c \cdot 2^{2k-2}$$

Use Master theorem to get $MM(n) = O(n^{\log 7}) \approx O(n^{2.807})$.

17.1.1 Matrix Multiplication Exponent

Definition: Matrix Multiplication Exponent

ω or ω_{mult} for matrix multiplication exponent.
If an algorithm for $n \times n$ matrix multiplication has runtime $O(n^\alpha)$, then $\omega \leq \alpha$. For any $\varepsilon > 0$, there is an algorithm for $n \times n$ matrix multiplication running in time $O(n^{\omega+\varepsilon})$.

As of today, $2 \leq \omega < 2.373$. Finding right ω is an open question in computer science.

We can similarly define ω_P for a problem P.

17.2 Matrix Inversion

Matrix inversion is at least as hard as matrix multiplication. This can proven by reduction; if we can invert matrices quickly, then we can multiply two matrices quickly.

Suppose we had an algorithm to invert matrices. Consider trying to multiply A and B. Define $M = \begin{bmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{bmatrix}$. Use the algorithm to find $M^{-1} = \begin{bmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{bmatrix}$.

So if we could invert in time $O(n^\alpha)$, then we can multiply two matrices in time $O(n^\alpha)$.

Matrix multiplication is at least as hard as matrix inversion.

Suppose we have an algorithm that performs matrix multiplication. Let $n = 2^k$, divide the matrix M into blocks of size $\frac{n}{2}$, i.e.

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

The inverse of M is

$$M^{-1} = \begin{bmatrix} I & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{bmatrix} \cdot \begin{bmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{bmatrix}$$

where $S := D - CA^{-1}B$ is the Schur complement and assuming A and S are invertible. Thus, to invert M, we need to invert A, compute $S := D - CA^{-1}B$, invert S, and then perform a constant number of multiplications.

The recurrence relation for this is

$$I(n) \leq 2 \cdot I(n/2) + C \cdot (n/2)^\omega$$

We know that $2 \leq \omega < 3$, then

$$I(2^k) \leq 2 \cdot I(2^{k-1}) + C \cdot 2^{\omega(k-1)}$$

Thus,

$$\begin{aligned} I(n) = I(2^k) &\leq 2^k \cdot I(1) + C \sum_{i=1}^{k-1} 2^{\omega i} \\ &\leq C' \left(2^k + \frac{2^{\omega k} - 1}{2^\omega - 1} \right) \\ &\leq C'' \cdot 2^{\omega k} \\ &= C'' n^\omega \end{aligned}$$

17.3 Determinant

Definition: Determinant

Given a matrix $M \in \mathbb{F}^{n \times n}$,

$$\det(M) = \sum_{\sigma \in \mathcal{S}_n} (-1)^\sigma \cdot \prod_{i=1}^{n} M_{i\sigma(i)}$$

Definition: Minor

Given a matrix $M \in \mathbb{F}^{n \times n}$ and $(i, j) \in [n]^2$, the (i, j)-minor of M, $M^{(i,j)}$, is obtained by removing the ith row and jth column of M.

Definition: Laplace Expansion of Determinant

Given a row i,

$$\det(M) = \sum_{j=1}^{n} (-1)^{i+j} M_{ij} \cdot \det(M^{(i,j)})$$

Proposition (Minor Determinant With Derivative of Determinant)

$$\det(M^{(i,j)}) = (-1)^{i+j} \partial_{i,j} \det(M)$$

Definition: Adjugate Matrix

The adjugate matrix $N \in \mathbb{F}^{n \times n}$ of M is

$$N_{ij} = (-1)^{i+j} \det(M^{(j,i)})$$

Proposition

Let $M, N \in \mathbb{F}^{n \times n}$ where N is the adjugate matrix of M, then

$$MN = \det(M) \cdot I \text{ or } M^{-1} = \frac{1}{\det(M)} \cdot N$$

Since entries of the adjugate matrix (determinants of minors) are related to derivatives of the determinant of M, if we could compute the determinant and all its partial derivatives, then we can compute the adjugate and the inverse.

Suppose we have an algorithm which computes the determinant in $O(n^\alpha)$ operations. Then we can compute the determinant and all its partial derivatives in $O(n^\alpha)$ operations and compute the inverse by computing

$$\frac{\det(M^{(i,j)})}{\det(M)}$$

17.4 Partial Derivatives

Definition: Algebraic Circuit in Ring R

A directed acyclic graph Φ with

- input gates labeled by variables $x_1, \ldots, x_n$ or elements of R.
- other gates labeled $+, \times, \div$.
- gates compute polynomial.

Definition: Size of Algebraic Circuit

Number of edges in the circuit, denoted $\mathcal{S}(\Phi)$.

Definition: Partial Derivative

Let $f(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$, then

$$\partial_i x_j^d = \begin{cases} dx_j^{d-1} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Theorem

If f can be computed using L operations of $+, -, \times$, then we can compute all partial derivatives simultaneously using $4L$ operations.

Computing partial derivatives are ubiquitous in computational tasks such as gradient descent methods or Newton iterations.

Definition: Chain Rule

$$\partial_i f(g_1, \ldots, g_m) = \sum_{j=1}^{m} (\partial_i f)(g_1, \ldots, g_m) \cdot \partial_i g_j$$

Chain rule has $2m$ partial derivatives, but if each function has m being constant, then the chain rule is cheap. Many of partial derivatives are zero or have already been computed. In backpropagation, we have to calculate partial derivatives in reverse.

Consider $P_1 = x_1 + x_2, P_2 = x_1 + x_3, P_3 = P_1 P_2, P_4 = x_4 P_3$. Computing all partial derivatives per operation directly is slow.

Replace $P_1 = y$, so $Q_2 = x_1 + x_3, Q_3 = yQ_2, Q_4 = x_4 Q_3$. Consider the algebraic circuit computing all partial derivatives of the circuit. We can transform it into one that computes all partial derivatives of P_4 by using chain rule. Note that

$$Q_4(x_1, x_2, x_3, x_4, y = P_1) = P_4$$

By chain rule,

$$\begin{aligned} \partial_i P_4 &= \sum_{j=1}^{4} (\partial_j Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i x_j) + (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1) \\ &= (\partial_j Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot 1 + (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1) \end{aligned}$$

P_1 depends on *at most* 2 variables. By induction, we know a circuit of size $\leq 4(L-1)$ which computes all the $\partial_i Q_4$. P_1 is of the form $\alpha x_i + \beta x_j, xI, x_j, \alpha x_i + \beta$. We can compute P_1 and its derivatives with at most 4 operations. So the circuit computes all partial derivatives of P_4 with size

$$\leq 4(L-1) + 4 = 4L$$

Part VII

Cryptography

Chapter 18

Zero-Knowledge Proofs

In cryptography, we want to communicate with other people/entities, which we may not trust. We may also not trust the channel of communication. Someone may eavesdrop messages, messages could be corrupted or someone could try to impersonate.

Situation: Alice has all her files encrypted. Bob requests the content of one of Alice's files. Alice can send the decrypted file to Bob, but Bob has no way of knowing that this message comes from Alice. Alice could prove to Bob it is the right file by sending the encryption key. But then Bob has access to her entire database. We want a way for Alice to convince Bob that she gave the right file without giving any more knowledge beyond that she gave the right file.

Definition: Zero-Knowledge Proofs

Proofs in which the verifier gains no knowledge beyond the validity of the assertion.

Knowledge has to do with your *computational ability*. If you could have found the answer without help, then you gained *no knowledge*.

If Bob asks Alice whether a graph G is Eulerian, Bob gains no knowledge since he could have computed it by himself (using Euler's theorem). However, if Bob asks Alice if G has a Hamiltonian cycle, then Bob gains knowledge. (If $P \neq NP$, then Bob could not compute it.)

In either case, Alice conveyed **information**.

Definition: Knowledge

Related to computational difficulty and about publicly known objects.

One gains knowledge when one obtains something one could not compute before.

Definition: Information

Unrelated to computational difficulty and about partially known objects.

One gains information when one obtains something one could not access before.

18.1 Classical Proofs

A claim $\mathcal{C}$ is given and a prover P writes down a proof that $\mathcal{C}$ is correct. Prover P sends proof to verifier V. The verifier has procedures (axioms and derivation rules) to check validity of proof. Verifier will accept or reject based on rules.

This is a one-way communication scheme. The verifier does not trust the prover. The proofs are not interactive.

18.1.1 NP

A claim $\mathcal{C} := x \in L$ is given. Prover P writes down a proof/witness w that $x \in L$. Prover sends w to verifier V. The verifier has a deterministic, polynomial time algorithm to check validity of w. Verifier accepts if and only if $V(x, w) = 1$.

18.1.2 Factoring

A claim $\mathcal{C} := N$, a product of two primes, is given. Prover P writes down a proof that there are two primes p, q such that $N = pq$. P sends (p, q) to verifier V. V computes pq and checks p, q are prime using a deterministic, polynomial time algorithm. Verifier accepts if and only if p, q are prime and $N = pq$.

18.1.3 Graph Isomorphism

A claim $\mathcal{C} := G_0, G_1$ is given, where G_0, G_1 are isomorphic. Prover P writes down an isomorphism ρ such that $\rho(G_0) = G_1$. P sends rho to verifier V. V checks that ρ is a permutation of vertices and $\rho(G_0) = G_1$ using a deterministic, polynomial time algorithm. Verifier accepts if and only if rho is a permutation of vertices and $\rho(G_0) = G_1$.

18.2 Zero-Knowledge Proofs

The proofs will be interactive, instead of one-way and the verifier is allowed private randomness.

18.2.1 Graph Isomorphism

A claim $\mathcal{C} := G_0, G_1$ that are isomorphic. Prover P produces a random graph H for which it can an isomorphism ρ_0, ρ_1 from G_0 and G_1 to H respectively.

This is possible if and only if G_0, G_1 are isomorphic.

The verifier picks a random bit $b \in \{0, 1\}$. The prover gives isomorphism ρ_b. The verifier checks that $\rho_b(H) = G_b$. The verifier will not learn the isomorphism between G_0 and G_1.

If the claim is true, then the prover can always give an isomorphism. If the claim is false, then we can catch a bad proof with probability $\frac{1}{2}$. By repeating the protocol, we can amplify this probability.

We can use simulation to model that the verifier does not gain knowledge. The key idea is if the claim is true, then the verifier's view of the proof could have been simulated by the verifier alone.

Simulation: The verifier privately produces a random permutation ρ and a bit b and outputs $H = \rho(G_b)$. Verifier then picks bit b from previous step and gives isomorphism ρ^{-1}. The verifier checks that $\rho^{-1}(H) = G_b$. V gained no new information.

Definition: Perfect Zero Knowledge – Strict

A valid proof system (P, V) is perfect zero-knowledge for language L if for every polynomial time, randomized verifier V^*, there is a randomized algorithm M^* such that for every $x \in L$, the following random variables are identically distributed:

- $\langle P, V^* \rangle (x)$: output of interaction between prover P and verifier V^* on input x.
- $M^*(x)$: output of algorithm M^* (simulation) on input x.

The previous definition is a bit too strict to be useful. We allow the simulator to fail with small probability (by outputting $\perp$).

Definition: Perfect Zero Knowledge

A valid proof system (P, V) is perfect zero-knowledge for language L if for every polynomial time, randomized verifier V^*, there is a randomized algorithm M^* such that for every $x \in L$, the following hold:

1. With probability $\leq \frac{1}{2}$, $M^*(x) = \perp$.
2. Conditioned on $M^*(x) \neq \perp$, the following variables are identically distributed:
   - $\langle P, V^* \rangle (x)$: output of interaction between prover P and verifier V^* on input x.
   - $M^*(x)$: output of algorithm M^* (simulation) on input x.

Simulation: Produces a random permutation ρ and outputs $H = \rho(G_0)$. Simulator picks

random bit b and if $b \neq 0$, then output $\perp$. Otherwise, the simulator gives isomorphism ρ^{-1} and the simulator checks that $\rho^{-1}(H) = G_0$. This is perfect zero knowledge for prover P.

Part VIII

Distributed Computing

Chapter 19

Distributed Algorithms

Definition: Distributed Algorithms

Algorithms which run on a network or multiprocessors within a computer which share memory.

Think of processors as vertices of a directed graph. Each processor has its own memory. Each processor can send messages to its outgoing neighbours. Processors communicate in synchronous rounds. There may or may not have failures.

Definition: Synchronous Model

$\Sigma \cup \{\perp\}$ is the message alphabet plus the special symbol $\perp$. For each vertex $i \in [n]$, a processor consists of

- S_i: non-empty set of states
- σ_i: start state
- $\mu_i : S_i \times out_i \to \Sigma \cup \{\perp\}$: messages function
- $\tau_i : S_i \times (\Sigma \cup \{\perp\})^{in_i} \to S_i$: transition function

Definition: Complexity Measure

Number of rounds/total data communicated to solve problem.

Assume processors have unlimited internal resources and that each processor is deterministic.

19.1 Leader Election

Leader Election problem

Input: network of processors.
Output: want to distinguish exactly one processor as the leader.

Consider when the processors are a ring network with bidirectional communication and they are numbered clockwise. All processors are identical and deterministic, so it is impossible to elect a leader. To show this, we look at execution and check all processors will be at identical states.

Leader Election Algorithm

Assume each processor has a unique ID (UID) and they do not know the size of the network. Idea is for each processor to send its UID in a message, relayed around the ring. When a processor receives a UID, compare it with its own.

1. If it is bigger, pass it on.
2. If is is smaller, discard.
3. If it is equal, declare itself leader.
4. Leader notifies every processor by relaying in network.

At the end, elect the processor with largest UID to be the leader.

After n rounds, the processor with maximum UID will declare itself leader. The number of rounds is $O(n)$ and communication takes $O(n^2)$. We can reduce communication to $O(n \log n)$ by successively doubling.

19.2 Consensus Problem

Consensus Problem

Input: each processor has one bit of input: 1 (attack) or 0 (don't attack).
Output: all should have same decision bit b satisfying weak validity (if at least one processor has bit 0, then 0 is only allowed decision).

Definition: Stopping Failure

All processors are good, but some may not be able to communicate (crashed).

Definition: Byzantine Failure

Some processors are malicious.

Byzantine Consensus Problem

Input: each processor has one bit of input: 1 (attack) or 0 (don't attack). Faulty processors can behave arbitrarily.
Output: all non-faulty processors should terminate and have same decision bit b (agreement) and if all non-faulty processors start with bit a, then b must be equal to a (weak validity).

Complexity measures: number of rounds and communication (messages in bits).

19.2.1 Complete Graph Byzantine Consensus

Assume all vertices can talk to any other vertex.

First attempt: Send our value to other non-faulty vertices, then take majority.

Let $n = 3, f = 1, p_1 = 1, p_2 = 0, p_3 = 0$. If p_3 sends 1 to p_1, then p_1 gets 101 and decides 1. But if p_3 sends 0 to p_2, then p_2 gets 100 and decides 0. This violates the agreement property.

Instead, make all vertices gossip, i.e. each vertex will keep track of what each vertex has told another. At each round, each vertex broadcasts its knowledge. After a number of rounds, everyone will make a decision.

Consider a bad example on a complete graph on 3 vertices v_1, v_2, v_3 with 1 faulty vertex. Consider the following scenarios.

Scenario 1: v_1, v_2 good with value 1, v_3 faulty with value 0.

Round 1: All vertices truthful.

Round 2: v_3 lies to v_1 saying v_2 said 0. All other communications are truthful.

Round 3: v_1, v_2 must decide 1.

Scenario 2: v_2, v_3 good with value 0, v_1 faulty with value 1.

Round 1: All vertices truthful.

Round 2: v_1 lies to v_3 saying v_2 said 1. All other communications are truthful.

Round 3: v_2, v_3 must decide 0.

Scenario 3: v_1, v_3 good with values $1, 0$ respectively, v_2 faulty with value 0.

Round 1: v_2 tells v_1 its value is 1 and tells v_3 its value is 0.

Round 2: All truthful.

Scenarios 1 and 3 are identical to v_1, so it must return 1 by weak validity. Scenarios 2 and 3 are identical to v_3, so it must return 0 by weak validity. This contradicts agreement in Scenario 3.

19.2.2 Exponential Information Gathering Algorithm

Assume $n > 3f$ where $n = |V|$ and f is the number of faulty vertices. We want to perfectly gossip.

If $n \leq 3f$, then no algorithm can reach consensus.

Definition: Exponential Information Gathering (EIG) Tree

A tree $T_{n,f}$ where the depth is $f+1$ and each tree node at level $k+1$ is labeled by string $i_1 i_2 \cdots i_k$ ($i_a \neq i_b$). Node $i_1 \cdots i_k$ wil store value v if the following happens: i_k told you that i_{k-1} told i_k that i_{k-2} told i_{k-1} ... that i_1 told i_2 that its initial value was v.

EIG Algorithm

1. Each vertex has its own EIG tree $T_{n,f}$ with root labeled with its own value.
2. Relay messages for $f+1$ rounds.
   - At round r, each vertex sends the values of level r of its EIG tree.
   - Each vertex decorates values of its $(r+1)$th level with values from messages.
3. After $f+1$ rounds, redecorate tree bottom-up, taking strict majority of children. If none, set value of tree node to $\perp$.
4. Output label on the root of EIG tree after redecoration.

Example: $n = 4, f = 1$. Initially p_3 is faulty and initial values are $p_1 = p_2 = 1$, $p_3 = p_4 = 0$.

Round 1: p_3 lies to p_2 and p_4.

Round 2: p_3 lies to p_2 about p_1 and lies to p_1 about p_2.

T_1, T_2, T_4 will output 1.

19.2.3 Analysis

Lemma (Consistency of Non-Faulty Messages)

If i, j, k are non-faulty, then $T_i(x) = T_j(x)$ whenever label x ends with k.

Lemma (Consistency of Upwards Relabeling)

If label x ends with non-faulty process, then for any two non-faulty processors i, j, the new values of $T_i(x)$ and $T_j(x)$ are the same.

Proof. Base case: if x is the label of leaf, the consistency of non-faulty messages lemma handles this.

Induction: $|x| = t \leq f$.

By induction, if ℓ is a non-faulty element, the new value of $T_i(x \circ \ell)$ is the same for any $i \in [n]$. So label x has same labeled children across trees, if x_ℓ is honest. The number of children of x is $n - t > 3f - f = 2f$. Since at most f are faulty, by taking the majority, we get new values $T_i(x) = T_j(x)$.

Termination: After $f + 1$ rounds, all of them will decide.

Proof. Every label x which has no faulty processor is able to update its value.

Validity: If all vertices start with b, then each label x with no faulty processor will be updated to b.

Proof. Analogous to the proof of the lemma.

Agreement: All vertices must agree on the same value.

Proof. By first lemma, all values in the leaves x are consistent across processors so long as x ends on a non-faulty process. By the second lemma, the majority will cause all values in nodes from level r ending in non-faulty nodes to be the same across processors. Induction and $n > 3f$ ensure that labels in level 1 will look the same on non-faulty nodes implying agreement.