Generalizations of the Unanimous Vote Problem

Evan Brody

Joint work with Lisa Hellerstein

08.07.2025

Warm-up:

- Suppose I give you the biased coins shown below, along with their corresponding Pr[heads]

Warm-up:

- Suppose I give you the biased coins shown below, along with their corresponding Pr[heads]

- Your task:
  - Choose an order to flip the coins until one lands on heads
  - (Or until you run out of coins)
- But: must pay $1 per coin you flip
- Rules:
  - Cannot "re-flip" a coin
  - Cannot change your mind about the ordering once you start flipping coins (*nonadaptive*)

Warm-up:

- Suppose I give you the biased coins shown below, along with their corresponding Pr[heads]

- Your task:
  - Choose an order to flip the coins until one lands on heads
  - (Or until you run out of coins)
- But: must pay \$1 per coin you flip
- Rules:
  - Cannot "re-flip" a coin
  - Cannot change your mind about the ordering once you start flipping coins (*nonadaptive*)
- **What's your strategy?**

A (much) harder problem:

- Same setup, but a change to when you can stop flipping coins
- Now: must see a heads *and* a tails
- Recall: our strategy must be *nonadaptive*
  - "No takesies-backsies"

A (much) harder problem:

- Same setup, but a change to when you can stop flipping coins
- Now: must see a heads *and* a tails
- Recall: our strategy must be *nonadaptive*
  - "No takesies-backsies"
- Equivalent to:
  - Set of Y/N voters; know their probabilities of voting Y or N
  - Goal: check if their votes were unanimous as quickly as you can
  - Must choose an order to check their ballots
- This is called the **Unanimous Vote Problem**
- Recently solved!
  [Duman Keles, Hellerstein, Marwaha, Musco, & Yang]

These last two problems were purely Boolean.
Are there non-Boolean versions of the Unanimous Vote Problem?

These last two problems were purely Boolean.
Are there non-Boolean versions of the Unanimous Vote Problem?

Many!

We focus on three "dice rolling" problems.

General problem statement:

- You are given n independent, biased, d-sided dice
  - Always: $d \leq n$
  - d could be a function of n, e.g. $d = \left\lceil \frac{n}{2} \right\rceil$
- You are told their biases upfront

General problem statement:

- You are given n independent, biased, d-sided dice
  - Always: $d \leq n$
  - d could be a function of n, e.g. $d = \left\lceil \frac{n}{2} \right\rceil$
- You are told their biases upfront
- Your task is to fix a permutation $\pi : [n] \to [n]$ of these dice, then roll them *in that order* until a stopping condition is met (or you run out of dice)
  - Stopping condition could be, e.g., "see three greens"
  - Cannot re-roll a die
  - Cannot change order partway through rolling
- Goal is to minimize the number of dice you need to roll

Our "dice rolling" problems, from easiest to hardest:

- *d*-ary Unanimous Vote
- Finite Coupon Collection Problem (FCCP)
- All-Or-Nothing Finite Coupon Collection Problem

Our "dice rolling" problems, from easiest to hardest:

- ***d*-ary Unanimous Vote**
  - Goal: check if dice (vote) outcomes are identical
  - Early stopping condition: see two different colors
- Finite Coupon Collection Problem (FCCP)

- All-Or-Nothing Finite Coupon Collection Problem

Our "dice rolling" problems, from easiest to hardest:

- *d*-ary Unanimous Vote
  - Goal: check if dice (vote) outcomes are identical
  - Early stopping condition: see two different colors
- **Finite Coupon Collection Problem (FCCP)**
  - Goal: collect as many different colors as you can
  - Early stopping condition: see every color at least once
- All-Or-Nothing Finite Coupon Collection Problem

Our "dice rolling" problems, from easiest to hardest:

- *d*-ary Unanimous Vote
  - Goal: check if dice (vote) outcomes are identical
  - Early stopping condition: see two different colors
- Finite Coupon Collection Problem (FCCP)
  - Goal: collect as many different colors as you can
  - Early stopping condition: see every color at least once
- **All-Or-Nothing Finite Coupon Collection Problem**
  - Goal: collect *every* color or determine that this is impossible
  - Early stopping condition: see every color at least once OR have too few remaining dice to do this

Our "dice rolling" problems, from easiest to hardest:

- d-ary Unanimous Vote
  - Goal: check if dice (vote) outcomes are identical
  - Early stopping condition: see two different colors
- Finite Coupon Collection Problem (FCCP)
  - Goal: collect as many different colors as you can
  - Early stopping condition: see every color at least once
- All-Or-Nothing Finite Coupon Collection Problem
  - Goal: collect *every* color or determine that this is impossible
  - Early stopping condition: see every color at least once OR have too few remaining dice to do this

When $d = 2$, these each become the Unanimous Vote Problem

Some terminology:

- We use n discrete random variables $x_1, \ldots, x_n$ to represent the outcome of each die roll
  - x_j is the result of rolling die j
  - When clear from context, we may refer to the index j, RV x_j, and die j interchangeably
- We use *realization* to mean an outcome of n dice rolls
  - Visualized like so:

- Usually use $\pi : [n] \to [n]$ to denote a permutation of the dice

Some (more) terminology:

- $\text{cost}(\pi, \alpha)$ is the number of dice rolls necessary if you roll dice in the order $x_{\pi(1)}, \ldots, x_{\pi(n)}$ and the outcomes match α
- E.g., suppose α is as shown below, and π goes from left to right. In the context of the d-ary Unanimous Vote Problem, $\text{cost}(\pi, \alpha) = 4$

- Can now restate what we'd like to compute:

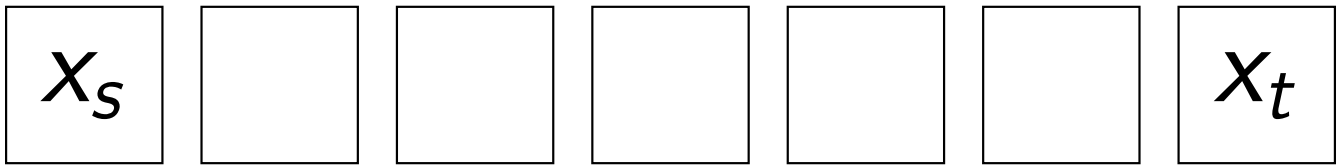
$$\operatorname*{argmin}_{\pi} \mathbb{E}[\text{cost}(\pi, \alpha)]$$

First up: *d*-ary Unanimous Vote

Recall our early stopping condition: see 2 different colors

Our 4-approximation algorithm is simple:

- For all possible pairs (x_s, x_t), $s \neq t$, generate a permutation $\pi_{s,t}$ that starts with s and ends with t



- Determining the "middle" of $\pi_{s,t}$:
  - Fill empty slots from $\pi(2)$ to $\pi(n-1)$ by greedy rule (see next slide)
- After all $\pi_{s,t}$ are generated: output the one with the least expected cost
- Easy to compute expected cost via dynamic programming

Greedy rule:

- Note: if we need to roll the k^{th} die, then the first $k-1$ dice must have had identical outcomes
- Choose die with highest probability of meeting stopping condition, conditioned on previous dice being identical
- E.g., suppose our colors are red, green, and blue.
  Choose x_j that maximizes:

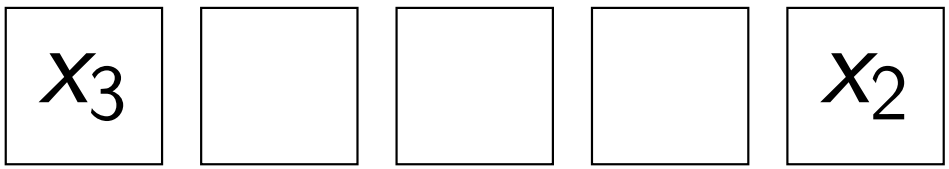
$$\begin{aligned}
&\Pr[x_{\pi(1)}, \ldots, x_{\pi(k-1)} \text{ all red}] \Pr[x_j \text{ not red}] \\
+&\Pr[x_{\pi(1)}, \ldots, x_{\pi(k-1)} \text{ all green}] \Pr[x_j \text{ not green}] \\
+&\Pr[x_{\pi(1)}, \ldots, x_{\pi(k-1)} \text{ all blue}] \Pr[x_j \text{ not blue}]
\end{aligned}$$

Quick example:

- $d = 3$, $n = 5$
- We are generating $\pi_{3,2}$

Quick example:

- d = 3, n = 5
- We are generating $\pi_{3,2}$

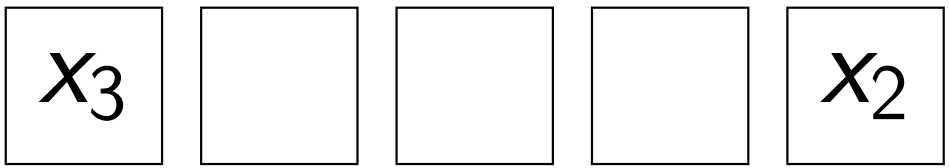


3/4
1/8
1/8

x_1	x_4	x_5
1/8	**1/2**	**1/3**
3/4	**1/4**	**1/3**
1/8	**1/4**	**1/3**

Quick example:

- d = 3, n = 5
- We are generating $\pi_{3,2}$



3/4
1/8
1/8

- After rolling x_3:
  - Pr[only seen red] = **3/4**
  - Pr[only seen green] = **1/8**
  - Pr[only seen blue] = **1/8**

x_1	x_4	x_5
1/8	**1/2**	**1/3**
3/4	**1/4**	**1/3**
1/8	**1/4**	**1/3**

Quick example:

- d = 3, n = 5
- We are generating $\pi_{3,2}$

x_3				x_2

3/4
1/8
1/8

- After rolling x_3:
  - Pr[only seen red] = **3/4**
  - Pr[only seen green] = **1/8**
  - Pr[only seen blue] = **1/8**

x_1	x_4	x_5
1/8	**1/2**	**1/3**
3/4	**1/4**	**1/3**
1/8	**1/4**	**1/3**

Quick example:

- d = 3, n = 5
- We are generating $\pi_{3,2}$

x_3	x_1			x_2

		x_4	x_5
3/4	**1/8**	**1/2**	**1/3**
1/8	**3/4**	**1/4**	**1/3**
1/8	**1/8**	**1/4**	**1/3**

- After rolling x_1:
  - Pr[only seen red] = **3/32**
  - Pr[only seen green] = **3/32**
  - Pr[only seen blue] = **1/64**

Quick example:

- d = 3, n = 5
- We are generating $\pi_{3,2}$

x_3	x_1			x_2

3/4	**1/8**	**1/2**	**1/3**	
1/8	**3/4**	**1/4**	**1/3**	
1/8	**1/8**	**1/4**	**1/3**	

- After rolling x_1:
  - Pr[only seen red] = **3/32**
  - Pr[only seen green] = **3/32**
  - Pr[only seen blue] = **1/64**

Quick example:

- d = 3, n = 5
- We are generating $\pi_{3,2}$

x_3	x_1	x_5		x_2
3/4	**1/8**	**1/3**		
1/8	**3/4**	**1/3**		
1/8	**1/8**	**1/3**		

- After rolling x_5:
  - Pr[only seen red] = **3/96**
  - Pr[only seen green] = **3/96**
  - Pr[only seen blue] = **1/192**

x_4
1/2
1/4
1/4

Quick example:

- d = 3, n = 5
- We are generating $\pi_{3,2}$

x_3	x_1	x_5		x_2
3/4	**1/8**	**1/3**		
1/8	**3/4**	**1/3**		
1/8	**1/8**	**1/3**		

x_4
1/2
1/4
1/4

Quick example:

- d = 3, n = 5
- We are generating $\pi_{3,2}$

x_3	x_1	x_5	x_4	x_2
3/4	**1/8**	**1/3**	**1/2**	
1/8	**3/4**	**1/3**	**1/4**	
1/8	**1/8**	**1/3**	**1/4**	

Why brute-force the first and last dice?

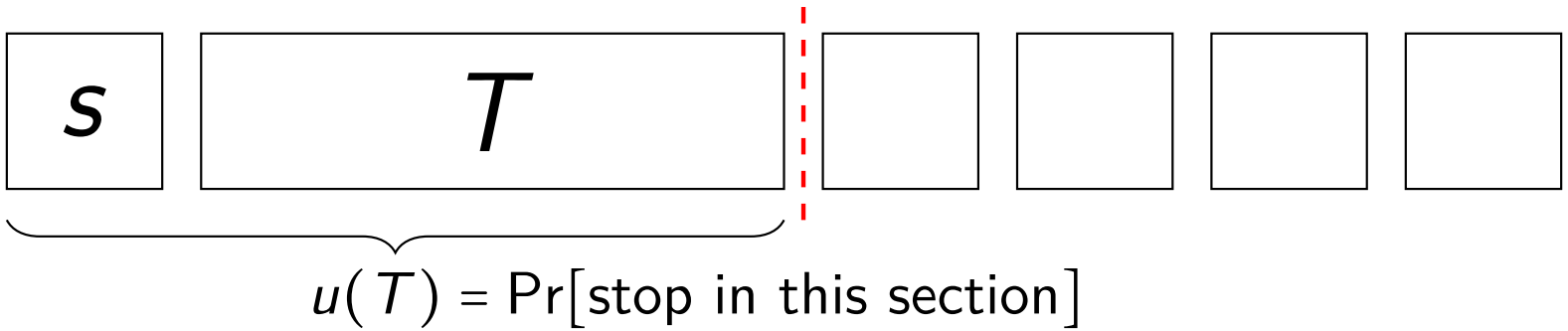
- Suppose a little birdie brings you the "correct" dice to use at the start and end of your permutation
- Remaining sub-problem is determining $\pi(i)$ for $i \in [\![2, n-1]\!]$
- Within this range, our problem becomes an instance of Min-Sum Submodular Cover with a natural utility function (see next slide)
  - If first or last test isn't fixed – this is no longer true
  - Our greedy rule is equivalent to maximizing this utility function
  - [Streeter & Golovin, 2008] implies a 4-approximation
- We don't have a little birdie, so we try everything and see what works best

Where precisely do we have submodularity?

- Let available dice be $A = [n] \backslash \{s, t\}$
- Let $u : 2^A \to [0, 1]$ be our utility function
- For $T \subseteq A$, $u(T)$ is the probability that rolling die s **and** every die in T allows us to stop

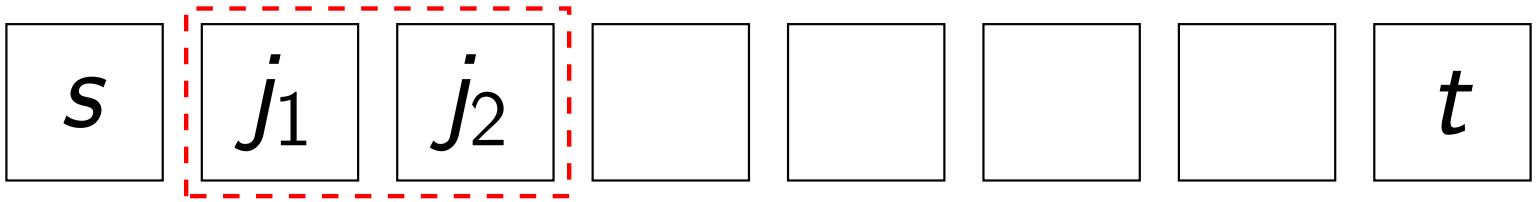
Where precisely do we have submodularity?

- Let available dice be $A = [n] \backslash \{s, t\}$
- Let $u : 2^A \to [0, 1]$ be our utility function
- For $T \subseteq A$, $u(T)$ is the probability that rolling die s **and** every die in T allows us to stop

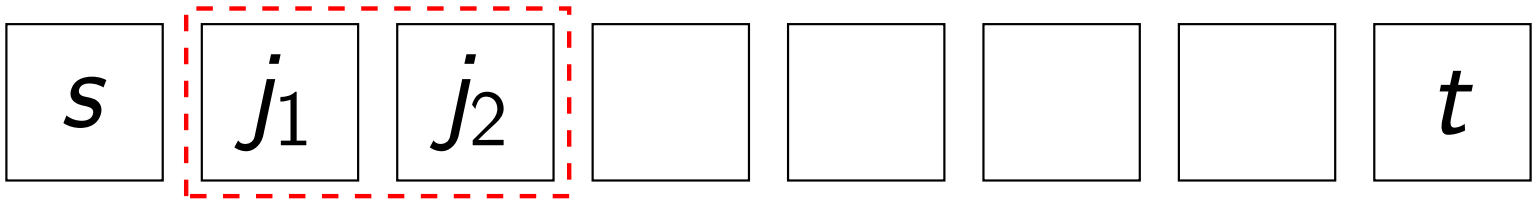


- If we included the first die in A:
  - Marginal value for first choice is zero – not submodular!
- If we included the last die in A:
  - After final choice, $u(T) = 1$ (out of dice)
  - Just before final choice, $u(T)$ could be arbitrarily small
  - Difference could violate submodularity

Suppose we currently have $T = \{j_1, j_2\}$:

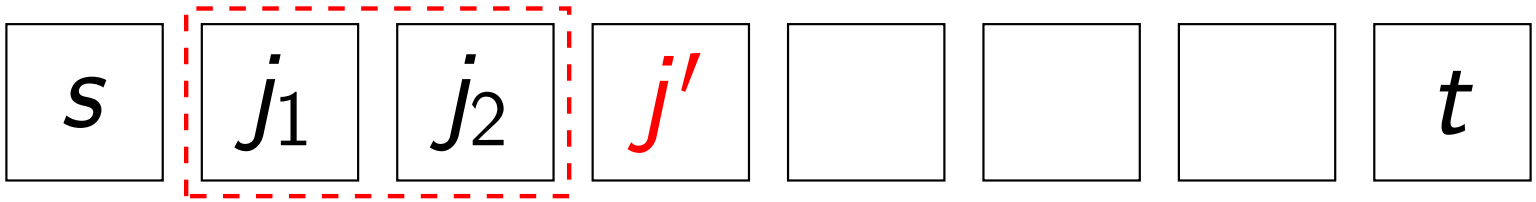


Suppose we currently have $T = \{j_1, j_2\}$:



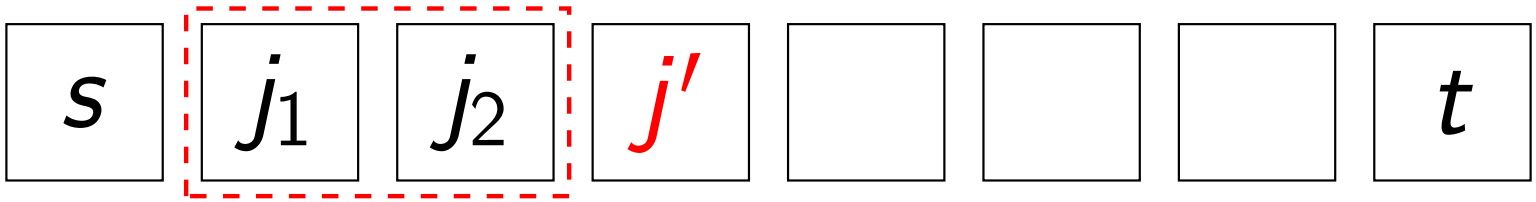
- Recall: $u(T) = \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$

~~Suppose we currently have $T = \{j_1, j_2\}$:~~



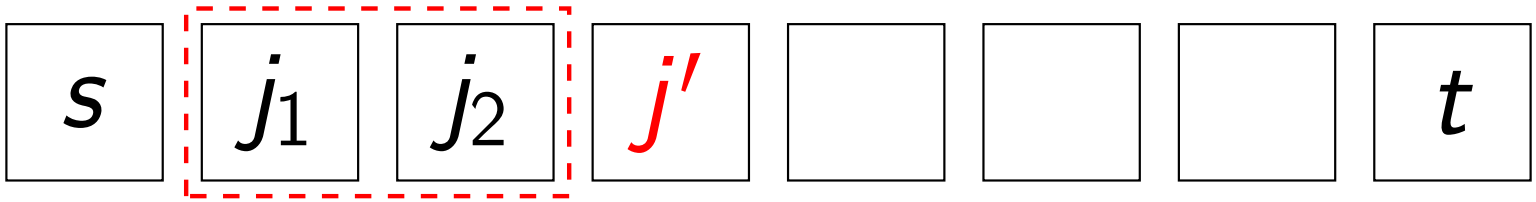
- Recall: $u(T) = \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$
- If we add die j'...

~~Suppose we currently have $T = \{j_1, j_2\}$:~~



- Recall: $u(T) = \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$
- If we add die j'...
  - Marginal increase: $u_T(j') = u(T \cup \{j'\}) - u(T)$

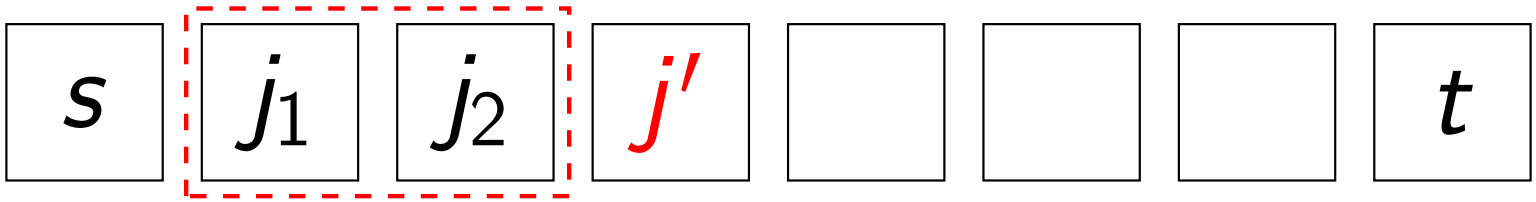
~~Suppose we currently have $T = \{j_1, j_2\}$:~~



- Recall: $u(T) = \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$
- If we add die j'...
  - Marginal increase: $u_T(j') = u(T \cup \{j'\}) - u(T)$

$$= \Pr[\text{cost}(\pi, \alpha) \leq |T| + 2] - \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$$

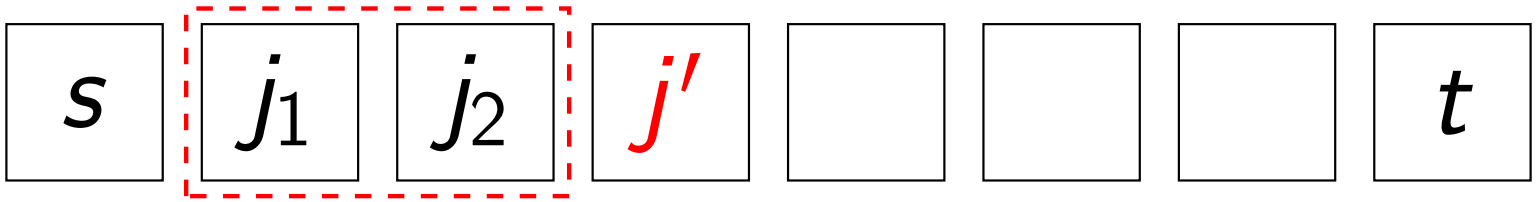
~~Suppose we currently have $T = \{j_1, j_2\}$:~~



- Recall: $u(T) = \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$
- If we add die j'...
  - Marginal increase: $u_T(j') = u(T \cup \{j'\}) - u(T)$

$$= \Pr[\text{cost}(\pi, \alpha) \leq |T| + 2] - \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$$
$$= \Pr[\text{cost}(\pi, \alpha) = |T| + 2]$$

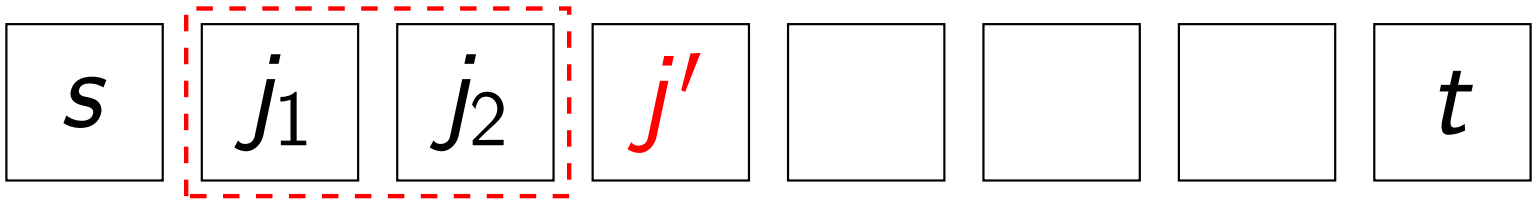
~~Suppose we currently have $T = \{j_1, j_2\}$:~~



- Recall: $u(T) = \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$
- If we add die j'...
  - Marginal increase: $u_T(j') = u(T \cup \{j'\}) - u(T)$

$$= \Pr[\text{cost}(\pi, \alpha) \leq |T| + 2] - \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$$
$$= \Pr[\text{cost}(\pi, \alpha) = |T| + 2]$$
$$= \Pr[\pi \text{ stops right after rolling } j']$$

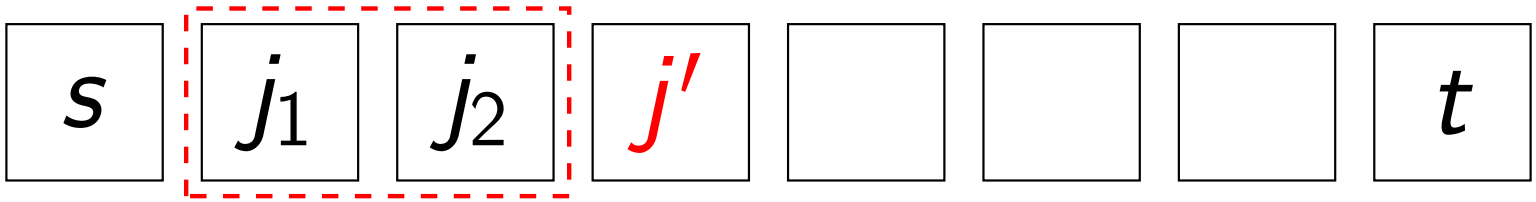
~~Suppose we currently have $T = \{j_1, j_2\}$:~~



- Recall: $u(T) = \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$
- If we add die j'...
  - Marginal increase: $u_T(j') = u(T \cup \{j'\}) - u(T)$

$$= \Pr[\text{cost}(\pi, \alpha) \leq |T| + 2] - \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$$
$$= \Pr[\text{cost}(\pi, \alpha) = |T| + 2]$$
$$= \Pr[\pi \text{ stops right after rolling } j']$$
$$= \Pr[\{s\} \cup T \text{ colored identically, } j' \text{ is not this color}]$$

~~Suppose we currently have $T = \{j_1, j_2\}$:~~



- Recall: $u(T) = \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$
- If we add die j'...
  - Marginal increase: $u_T(j') = u(T \cup \{j'\}) - u(T)$

$$= \Pr[\text{cost}(\pi, \alpha) \leq |T| + 2] - \Pr[\text{cost}(\pi, \alpha) \leq |T| + 1]$$
$$= \Pr[\text{cost}(\pi, \alpha) = |T| + 2]$$
$$= \Pr[\pi \text{ stops right after rolling } j']$$
$$= \Pr[\{s\} \cup T \text{ colored identically, } j' \text{ is not this color}]$$
$$= \Pr[\{s\} \cup T \text{ colored identically}] \Pr[j' \text{ is not this color}]$$

We now know:

$$u_T(j') = \Pr[\{s\} \cup T \text{ colored identically}] \Pr[j' \text{ is not this color}]$$

We now know:

$$u_T(j') = \Pr[\{s\} \cup T \text{ colored identically}] \Pr[j' \text{ is not this color}]$$

- If for all $T' \supseteq T$ such that $j' \notin T'$, $u_{T'}(j') \leq u_T(j')$, we have submodularity

We now know:

$$u_T(j') = \Pr[\{s\} \cup T \text{ colored identically}] \Pr[j' \text{ is not this color}]$$

- If for all $T' \supseteq T$ such that $j' \notin T'$, $u_{T'}(j') \leq u_T(j')$, we have submodularity
- So, consider:

$$u_{T'}(j') = \Pr[\{s\} \cup T' \text{ colored identically}] \Pr[j' \text{ is not this color}]$$

We now know:

$$u_T(j') = \Pr[\{s\} \cup T \text{ colored identically}] \Pr[j' \text{ is not this color}]$$

- If for all $T' \supseteq T$ such that $j' \notin T'$, $u_{T'}(j') \leq u_T(j')$, we have submodularity

Equal!

- So, consider:

$$u_{T'}(j') = \Pr[\{s\} \cup T' \text{ colored identically}] \Pr[j' \text{ is not this color}]$$

We now know:

$$u_T(j') = \Pr[\{s\} \cup T \text{ colored identically}] \Pr[j' \text{ is not this color}]$$

- If for all $T' \supseteq T$ such that $j' \notin T'$, $u_{T'}(j') \leq u_T(j')$, we have submodularity

Smaller!

- So, consider:

$$u_{T'}(j') = \Pr[\{s\} \cup T' \text{ colored identically}] \Pr[j' \text{ is not this color}]$$

Next up: Finite Coupon Collector's Problem

Recall early stopping condition: See all d distinct colors

For this problem:

- Can separate realizations α into:
  - $\mathcal{G} = \{\alpha : \text{all colors present in } \alpha\}$
    - e.g., for $d = 3$: $\alpha =$ 
  - $\mathcal{B} = \{\alpha : \text{not all colors present in } \alpha\}$
    - e.g., $\alpha =$ 

For this problem:

- Can separate realizations α into:
  - $\mathcal{G} = \{\alpha : \text{all colors present in } \alpha\}$
    - e.g., for $d = 3$: $\alpha =$ 
  - $\mathcal{B} = \{\alpha : \text{not all colors present in } \alpha\}$
    - e.g., $\alpha =$ 
- Realizations in $\mathcal{B}$:
  - We are doomed to n rolls...
- So: makes sense to optimize over distribution induced by conditioning on $\alpha \in \mathcal{G}$

For this problem:

- Can separate realizations α into:
  - $\mathcal{G} = \{\alpha : \text{all colors present in } \alpha\}$
    - e.g., for $d = 3$: $\alpha =$ 
  - $\mathcal{B} = \{\alpha : \text{not all colors present in } \alpha\}$
    - e.g., $\alpha =$ 
- Realizations in $\mathcal{B}$:
  - We are doomed to n rolls...
- So: makes sense to optimize over distribution induced by conditioning on $\alpha \in \mathcal{G}$
- Can treat as a special case of **Submodular Ranking** [Azar & Gamzu, 2011]

- For each realization $\alpha \in \mathcal{G}$, define the utility function:

  $$u_\alpha(T) = \#\text{distinct colors from dice in } T \text{ on realization } \alpha$$

- This utility function allows us to apply Azar & Gamzu's greedy algorithm
  - Since maximum value of any u_α is d, their analysis implies an $\mathcal{O}(\log d)$-approximation

Applying Azar & Gamzu's greedy algorithm:

- With already selected dice T, our greedy choice is

$$\operatorname*{argmax}_{j \notin T} \mathbb{E}\left[\frac{u_\alpha(T \cup \{j\}) - u_\alpha(T)}{d - u_\alpha(T)} \;\middle|\; u_\alpha(T) < d, \alpha \in \mathcal{G} \right]$$

Applying Azar & Gamzu's greedy algorithm:

- With already selected dice T, our greedy choice is

$$\operatorname*{argmax}_{j \notin T} \mathbb{E}\left[\frac{u_\alpha(T \cup \{j\}) - u_\alpha(T)}{d - u_\alpha(T)} \;\middle|\; u_\alpha(T) < d, \alpha \in \mathcal{G} \right]$$

- Looks messy, but not so bad
- Consider the expression for a fixed $\alpha \in \mathcal{G}$
- Assume dice in T don't have all colors
  - Numerator: # of new colors gained from x_j (either 0 or 1)
  - Denominator: # of colors *not* seen before rolling x_j

Applying Azar & Gamzu's greedy algorithm:

- With already selected dice T, our greedy choice is

$$\operatorname*{argmax}_{j \notin T} \mathbb{E}\left[\frac{u_\alpha(T \cup \{j\}) - u_\alpha(T)}{d - u_\alpha(T)} \;\middle|\; u_\alpha(T) < d, \alpha \in \mathcal{G} \right]$$

- Looks messy, but not so bad
- Consider the expression for a fixed $\alpha \in \mathcal{G}$
- Assume dice in T don't have all colors
  - Numerator: # of new colors gained from x_j (either 0 or 1)
  - Denominator: # of colors *not* seen before rolling x_j
- But how to calculate greedy choice?
  - Tough to compute exactly in poly(n) time
  - Best we can do is poly$(n, 2^d)$ using dynamic programming
  - So: we approximate

$$\operatorname*{argmax}_{j \notin T} \mathbb{E}\left[\frac{u_\alpha(T \cup \{j\}) - u_\alpha(T)}{d - u_\alpha(T)} \,\middle|\, u_\alpha(T) < d, \alpha \in \mathcal{G} \right]$$

Approximating the greedy choice:

- Simulate dice rolls to generate poly(n)-sized set of realizations
- Reject α such that
  - $u_\alpha(T) = d$ (these are already done)
  - $\alpha \notin \mathcal{G}$ (these are hopeless)
- Find j that maximizes the above over this set of realizations
- WHP, this is a $\frac{1}{4}$-approximation to the "exact" maximum
  - Azar & Gamzu's analysis assumes an exact greedy choice, but their bound still holds
  - Sampling analysis mostly drawn from [Ghuge, Gupta, Nagarajan, 2021], with slight differences
  - They use Chernoff bounds, we use Hoeffding's Inequality

A note on our approximation factors:

- Recall: our algorithm for the previous problem (d-ary Unanimous Vote) was deterministic
- This is a randomized algorithm (due to simulated sampling)
- Slight change of statement in our approximation guarantee
- For d-ary Unanimous Vote:
  - 4-approximation, in expectation, over outcomes of dice rolls
- For Finite Coupon Collection Problem:
  - $\mathcal{O}(\log d)$-approximation, in expectation, over outcomes of dice rolls ***AND*** random choices of algorithm

Final problem: All-Or-Nothing Finite Coupon Collection Problem

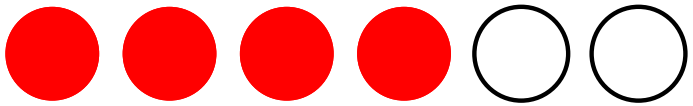
Early stopping condition: see every color at least once
OR determine that this is impossible

Final problem: All-Or-Nothing Finite Coupon Collection Problem

Early stopping condition: see every color at least once
OR determine that this is impossible

Same as the previous problem, except for adding ↑

- Second stopping condition:
  - Becomes impossible to see every color at least once
- For example:
  - $d = 4, n = 6$; our colors are red, green, blue, and yellow
  - Suppose we roll our dice and get:



- Can't get green, blue, and yellow in just two rolls

- Important note on the early stopping conditions:
  - "Collect all colors" condition only applies to $\alpha \in \mathcal{G}$
  - "Impossibility" condition only applies to $\alpha \in \mathcal{B}$
- Our best algorithm so far is *exactly* the previous algorithm
- For $\alpha \in \mathcal{B}$
  - Can show that *any* permutation is a d-approximation
- For $\alpha \in \mathcal{G}$
  - Reduces to the previous problem
- Results in an $\mathcal{O}(d)$-approximation
- "Impossibility" early stopping condition is the bottleneck
  - This motivates our next problem

Open problem: "2-of-a-Kind"

Stopping condition: find two of the same color

Here: $d = n$

- Motivation: this is a special case of the previous problem, All-Or-Nothing FCCP, where $d = n$
  - $\mathcal{O}(d)$-approximation guarantee not so good now...
- "Collect all colors" stopping condition can't let you stop early
- "Impossibility" stopping condition is all that matters
  - Gaining just one duplicate means you stop
- Note: the last two problems were related to the Coupon Collection Problem
- This one is more like the Birthday Problem
  - "Find two people with the same birthday ASAP"

In most "dice rolling / coin flipping" problems like these:

- An initial step is figuring out how to calculate $\mathbb{E}[\text{cost}(\pi, \alpha)]$
- Natural dynamic programming approach is usually easy
- Can compute $\Pr[\text{cost}(\pi, \alpha) > t]$ with some intuitive recurrence relation
- Here, this approach is dead: **#P-hard**
  - Computing $\Pr[\text{cost}(\pi, \alpha) > t]$ is equivalent to computing the permanent of a $t \times t$ nonnegative matrix
- Recall: For $\mathbf{A} \in \mathbb{R}^{t \times t}$,

$$\text{perm}(\mathbf{A}) = \sum_{\sigma \in S_t} \prod_{i=1}^{t} a_{i,\sigma(i)}$$

- $S_t \equiv$ all permutations of $\{1, \ldots, t\}$

Computing $\Pr[\text{cost}(\pi, \alpha) > t]$

- Recall our stopping condition: see two of the same color
- Only way this doesn't happen:
  - Each roll results in a unique color
- For convenience: suppose $t = 3$ and $n = 5$
  (red, green, blue, yellow, purple)
- Event might look like:

- Why this is connected to the permanent?

- Recall: $t = 3,\ n = 5$
- Want probability of first 3 dice having unique colors
- Sub-problem: suppose we fix the 3 colors we will see
- E.g., what's the probability we see exactly green, blue, & purple in the first 3 rolls?
- Easy to see that this is precisely

$$\operatorname{perm}\left(\begin{bmatrix} \Pr[x_{\pi(1)} = G] & \Pr[x_{\pi(2)} = G] & \Pr[x_{\pi(3)} = G] \\ \Pr[x_{\pi(1)} = B] & \Pr[x_{\pi(2)} = B] & \Pr[x_{\pi(3)} = B] \\ \Pr[x_{\pi(1)} = P] & \Pr[x_{\pi(2)} = P] & \Pr[x_{\pi(3)} = P] \end{bmatrix}\right)$$

What if the 3 colors we see aren't fixed?

$$\text{Let } \mathbf{P} = \begin{bmatrix} \Pr[x_{\pi(1)} = R] & \Pr[x_{\pi(2)} = R] & \Pr[x_{\pi(3)} = R] & 1 & 1 \\ \Pr[x_{\pi(1)} = G] & \Pr[x_{\pi(2)} = G] & \Pr[x_{\pi(3)} = G] & 1 & 1 \\ \Pr[x_{\pi(1)} = B] & \Pr[x_{\pi(2)} = B] & \Pr[x_{\pi(3)} = B] & 1 & 1 \\ \Pr[x_{\pi(1)} = Y] & \Pr[x_{\pi(2)} = Y] & \Pr[x_{\pi(3)} = Y] & 1 & 1 \\ \Pr[x_{\pi(1)} = P] & \Pr[x_{\pi(2)} = P] & \Pr[x_{\pi(3)} = P] & 1 & 1 \end{bmatrix}$$

- Consider computing the permanent via Laplace expansion
- Choose columns from right to left
- Continue until you've induced a submatrix like the one on the previous slide

$$\text{So, } \Pr[\text{cost}(\pi, \alpha) > t] = \frac{\text{perm}(\mathbf{P})}{(n-t)!}$$

- Denominator corrects for overcounting

So far, we know:

- Any alg for computing $\Pr[\text{cost}(\pi, \alpha) > t]$ can compute the permanent of a special kind of nonnegative matrix
  - Just multiply output by $(n - t)!$
- The "special kind" it can handle:
  - t 1-normalized columns, then $n - t$ columns of ones
- Only remains to show that this can handle *any* $t \times t$ nonnegative matrix

Suppose you have **ALG** for computing the permanent of this restricted type of matrix

- Have arbitrary $\mathbf{A} \in \mathbb{R}_{\geq 0}^{t \times t}$, we want $\text{perm}(\mathbf{A})$
- If $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_t]$, construct $\hat{\mathbf{A}} = \left[\frac{\mathbf{a}_1}{\|\mathbf{a}_1\|_1} \cdots \frac{\mathbf{a}_t}{\|\mathbf{a}_t\|_1} \right]$
  - $\hat{\mathbf{A}}$ is "normalized" $\mathbf{A}$
- Important:

$$\text{perm}(\hat{\mathbf{A}}) = \frac{\text{perm}(\mathbf{A})}{\prod_{i=1}^{t} \|\mathbf{a}_i\|_1}$$

- We now use **ALG** to compute $\text{perm}(\hat{\mathbf{A}})$

Construct $\mathbf{P}$ as:

$$\mathbf{P} = \left[\begin{array}{ccc|ccc} & & & 1 & \cdots & 1 \\ & \hat{\mathbf{A}} & & \vdots & \ddots & \vdots \\ & & & 1 & \cdots & 1 \\ \hline 0 & \cdots & 0 & 1 & \cdots & 1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & \cdots & 1 \end{array}\right] \begin{array}{l} \left.\vphantom{\begin{array}{c}1\\1\\1\end{array}}\right\} t \\ \left.\vphantom{\begin{array}{c}1\\1\\1\end{array}}\right\} n-t \end{array}$$

$$\underbrace{\phantom{\begin{array}{ccc}0&\cdots&0\\0&\cdots&0\end{array}}}_{t} \quad \underbrace{\phantom{\begin{array}{ccc}1&\cdots&1\\1&\cdots&1\end{array}}}_{n-t}$$

- $\mathbf{P}$ is of the form that **ALG** can work with
- Consider Laplace expansion of $\text{perm}(\mathbf{P})$, from the bottom row up
- It's clear that $\text{perm}(\mathbf{P}) = (n-t)!\,\text{perm}(\hat{\mathbf{A}})$
- So, return:

$$\mathbf{ALG}(\mathbf{P})\frac{\prod_{i=1}^{n}\|\mathbf{a}_i\|_1}{(n-t)!} = \text{perm}(\mathbf{A})$$

Thank you! ☺

References

Matthew Streeter and Daniel Golovin.
An online algorithm for maximizing submodular functions.
Advances in Neural Information Processing Systems, 21, 2008.

Yossi Azar and Iftah Gamzu.
Ranking with Submodular Valuations, pages 1070–1079.

Rohan Ghuge, Anupam Gupta, and Viswanath Nagarajan.
The power of adaptivity for stochastic submodular cover.
In International Conference on Machine Learning, pages 3702–3712. PMLR, 2021.