Random-Order Models

Seminar in Algorithms - Beyond Worst Case Analysis

Alon Alexander

5/6/2023

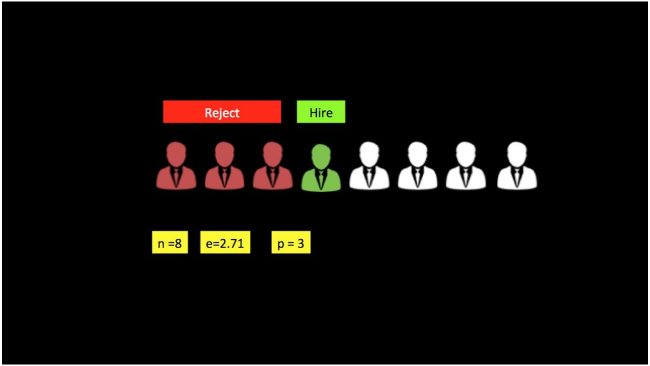
Table of Contents

Introduction

Motivation Through Example

The Secretary Problem

- Reminder



Motivation Through Example

The Secretary Problem

- Reminder
- Worst case is too harsh

Assume M is very large, and we look at the case of $n = 2$.

Example 1

$1, \; M$

Example 2

$1, \; {}^{1}/_{M}$

Motivation Through Example

The Secretary Problem

- Reminder
- Worst case is too harsh

Yao's Lemma

Given a randomized algorithm A, and an input distribution D.

It is true that

$$\max_{x \in D} \mathbb{E}\left[A\left(x\right)\right] \geq \min_{ALG} \mathop{\mathbb{E}}_{x \in D}\left[ALG\left(x\right)\right]$$

So that the min is on all deterministic algorithms.

Motivation Through Example

The Secretary Problem

- Reminder
- Worst case is too harsh

Example Distribution - Bad Deterministic Algorithm

$1,\ 0, \dots,\ 0$

$1,\ M,\ 0, \dots,\ 0$

$\vdots$

$1,\ M,\ M^2, \dots,\ M^k,\ 0, \dots,\ 0$

$\vdots$

$1,\ M,\ M^2, \dots,\ M^{n-1}$

Motivation Through Example

The Secretary Problem

- Reminder
- Worst case is too harsh
- Random-order highlights different aspects

Theorem

There is a random-order algorithm for the secretary problem which chooses the best item with a probability $1/e$.

Discussion

By assuming random-order on the set of requests, we analyze the problem in a different way.

Definitions

- Adversary
- Optimal reward/cost
- Competitive-ratio
- Random-order model

Definitions

- Adversary
- Optimal reward/cost
- Competitive-ratio
- Random-order model

Definition

Given an adversary-chosen **set** $S = \{r_1, \ldots, r_n\}$ of requests, we imagine nature drawing a uniformly random permutation π of $\{1, \ldots, n\}$ and define the input sequence to be $r_{\pi(1)}, \ldots, r_{\pi(n)}$.

Definitions - Cont.

Definition

Given an algorithm A, we define the **competitive-ratio** to be $\frac{OPT}{\mathbb{E}[A]}$ for maximization problems and $\frac{\mathbb{E}[A]}{OPT}$ for minimization problems on an adversary-chosen (worst case) set of inputs.

The expected-value is over all permutations of the input, and the algorithm (in the case it is not deterministic).

Definitions - Cont.

Definition

Given an algorithm A, we define the **competitive-ratio** to be $\frac{OPT}{\mathbb{E}[A]}$ for maximization problems and $\frac{\mathbb{E}[A]}{OPT}$ for minimization problems on an adversary-chosen (worst case) set of inputs.

The expected-value is over all permutations of the input, and the algorithm (in the case it is not deterministic).

The algorithm we know for the secretary problem can be called an e-competitive algorithm.

Our First Theorem

Theorem

The strategy that maximizes the probability of ***picking the highest number*** *can be assumed to be a wait-and-pick strategy.*

Our First Theorem

Note

Our First Theorem - Proof

Definition

We say v_i is **prefix-maximum** (later denoted Pmax) if $\max_{1 \leq j \leq i} v_j = v_i$.

Our First Theorem - Proof

Definition

We say v_i is **prefix-maximum** (later denoted Pmax) if $\max_{1 \leq j \leq i} v_j = v_i$.

Assume we are the best algorithm.

Obviously, if v_i is **not** a prefix-maximum, we should not pick it.

Our First Theorem - Proof

Definition

We say v_i is **prefix-maximum** (later denoted Pmax) if $\max_{1 \leq j \leq i} v_j = v_i$.

Assume we are the best algorithm.

Obviously, if v_i is **not** a prefix-maximum, we should not pick it.

Otherwise, we should pick it only if

$$f(i) := P[v_i \text{ is max} \mid v_i \text{ is Pmax}] \geq$$
$$\text{chance of choosing the maximum later} =: g(i)$$

Let's analyze these probabilities.

Our First Theorem - Proof Cont.

Lemma

Let's calculate the following function:

$$f(i) := P\left[v_i \text{ is } max \mid v_i \text{ is } Pmax\right] = \frac{P\left[v_i \text{ is } max\right]}{P\left[v_i \text{ is } Pmax\right]} = \frac{1/n}{1/i} = \frac{i}{n}$$

Our First Theorem - Proof Cont.

Lemma

Let's calculate the following function:

$$f(i) := P[v_i \text{ is max} \mid v_i \text{ is Pmax}] = \frac{P[v_i \text{ is max}]}{P[v_i \text{ is Pmax}]} = \frac{1/n}{1/i} = \frac{i}{n}$$

Note it increases.

Our First Theorem - Proof Cont.

Lemma

Let's calculate the following function:

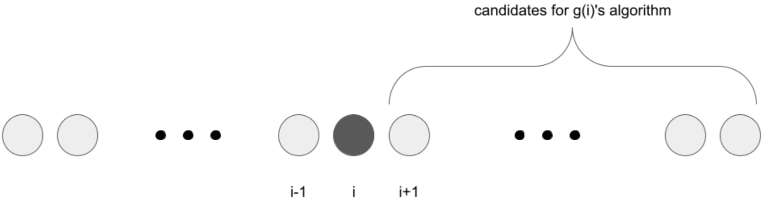
$$f(i) := P\left[v_i \text{ is max} \mid v_i \text{ is Pmax}\right] = \frac{P\left[v_i \text{ is max}\right]}{P\left[v_i \text{ is Pmax}\right]} = \frac{1/n}{1/i} = \frac{i}{n}$$

Note it increases.

Definition

Define $g(i)$ to be the probability that the optimal solution picks the maximum value, assuming it must discard the first i items.

Our First Theorem - Proof Cont.



Our First Theorem - Proof Cont.

Proof.

Reminder, we should pick v_i only if it is prefix-maximum and

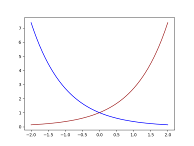
$$f(i) := P[v_i \text{ is max} \mid v_i \text{ is Pmax}]$$
$$\geq$$
$$\text{chance of choosing the maximum later} =: g(i)$$

Our First Theorem - Proof Cont.

Proof.

Reminder, we should pick v_i only if it is prefix-maximum and

$$f(i) := P[v_i \text{ is max} \mid v_i \text{ is Pmax}]$$
$$\geq$$
$$\text{chance of choosing the maximum later} =: g(i)$$

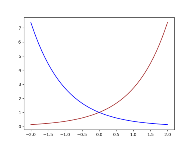


Our First Theorem - Proof Cont.

Proof.

Reminder, we should pick v_i only if it is prefix-maximum and

$$f(i) := P\left[v_i \text{ is max} \mid v_i \text{ is Pmax}\right]$$
$$\geq$$
$$\text{chance of choosing the maximum later} =: g(i)$$



So waiting until $f(i) \geq g(i)$ and then picking the first

Order-Oblivious Algorithms

Order-Oblivious Algorithms

Definition

An order-oblivious algorithm and analysis is defined with the following two-phase structure

1. We give algorithm a uniformly random subset of items, but is not allowed to pick any of these items.
2. Then, the remaining items arrive in an adversarial order, and only now can the algorithm pick items while respecting any constraints that exist.

Order-Oblivious Algorithms - Benefits

- It is easy to design and analyze algorithms in this environment.
- The guarantees of such algorithms can be interpreted as holding even for adversarial arrivals, as long as we have offline access to some samples from the underlying distribution.

Multiple-Secretary Problem

Instead of choosing 1 element, we now choose k elements.

Definitions

Define $S^\star \subseteq [n]$ to be the set of k items of the largest value, and define $V^\star := \sum_{i \in S^\star} v_i$ the total value of the set.

Multiple-Secretary Problem

Instead of choosing 1 element, we now choose k elements.

Definitions

Define $S^\star \subseteq [n]$ to be the set of k items of the largest value, and define $V^\star := \sum_{i \in S^\star} v_i$ the total value of the set.

It is easy to get expected value of $\Omega(V^\star)$ by splitting the data to k equal-sized sections, and running our e-algorithm on each of them.

Multiple-Secretary Problem

Instead of choosing 1 element, we now choose k elements.

Definitions

Define $S^\star \subseteq [n]$ to be the set of k items of the largest value, and define $V^\star := \sum_{i \in S^\star} v_i$ the total value of the set.

It is easy to get expected value of $\Omega(V^\star)$ by splitting the data to k equal-sized sections, and running our e-algorithm on each of them.

We want to do better, and reach the best $V^\star(1 - O(?))$ we can.

Multiple-Secretary Problem

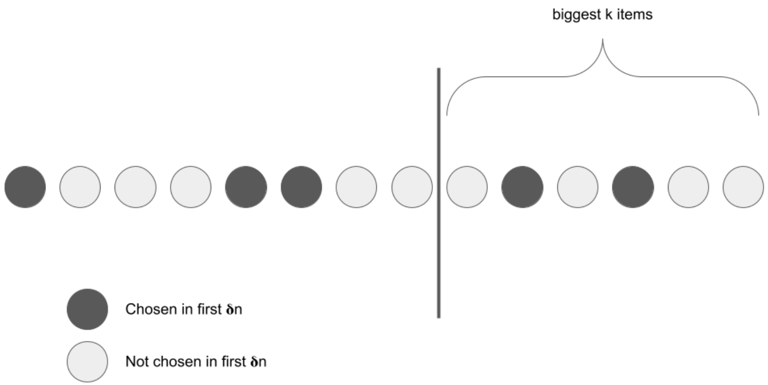
An Order-Oblivious Algorithm

The Algorithm

1. Set $\varepsilon = \delta = O\left(\frac{\log k}{k^{1/4}}\right)$.
2. Ignore the first δn items and set $\tau :=$ the value of the $(1 - \varepsilon)\,\delta k^{\text{th}}$-highest valued item in this set.
3. Pick the first k items that are greater than τ.

Multiple-Secretary Problem

An Order-Oblivious Algorithm



Multiple-Secretary Problem

An Order-Oblivious Algorithm

Theorem

This algorithm has an expected value of $V^\star \left(1 - O\left(\delta\right)\right)$.

Multiple-Secretary Problem

Explaining the Expected Value

Set $v' = \min_{i \in S^\star} v_i$ the minimal value we actually want to pick.

Multiple-Secretary Problem

Explaining the Expected Value

Set $v' = \min_{i \in S^\star} v_i$ the minimal value we actually want to pick.

We fail in 2 cases:

1. If $\tau < v'$
2. If there are less than $k - O(\delta k)$ items from $S^\star$ that are among the last $(1-\delta)n$ items and greater than τ.

Explaining the Expected Value

Bounding the Error

Is τ too low?

Explaining the Expected Value

Bounding the Error

Is τ too low?

Chernoff-Hoeffding concentration bound on the event that $\tau < v'$.

Remember we define τ to be the value of the $(1-\varepsilon)\,\delta k^{\text{th}}$-highest valued item the first δn items.

Explaining the Expected Value

Bounding the Error

Is τ too low?

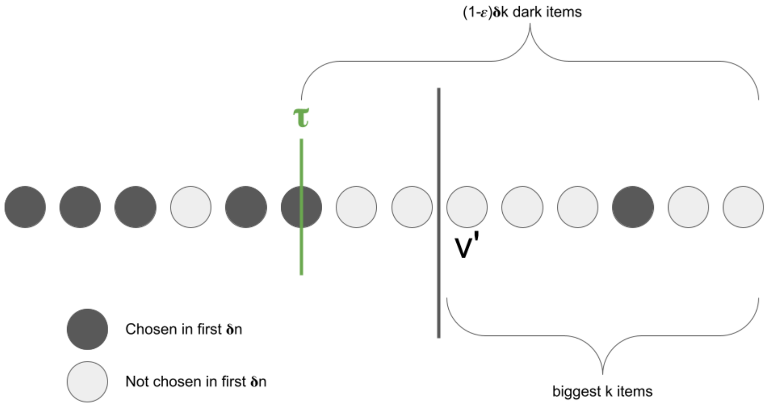
Chernoff-Hoeffding concentration bound on the event that $\tau < v'$.

Remember we define τ to be the value of the $(1 - \varepsilon)\,\delta k^{\text{th}}$-highest valued item the first δn items.

This event means we have fewer than $(1 - \varepsilon)\,\delta k$ elements from $S^\star$ in the first δn locations.

Explaining the Expected Value

Bounding the Error



Explaining the Expected Value

Bounding the Error

Define $X_1, \ldots, X_k$ to be indicators such that $X_i = 1$ iff the highest i'th number is in the first δn locations.

Define $S_k = \sum_{i=1}^{k} X_i$.

Explaining the Expected Value

Bounding the Error

Define $X_1, \ldots, X_k$ to be indicators such that $X_i = 1$ iff the highest i'th number is in the first δn locations.

Define $S_k = \sum_{i=1}^{k} X_i$.

Notice that $\mathbb{E}[X_i] = \delta$ and so $\mathbb{E}[S_k] = \delta k$.

Explaining the Expected Value

Bounding the Error

Define $X_1, \ldots, X_k$ to be indicators such that $X_i = 1$ iff the highest i'th number is in the first δn locations.

Define $S_k = \sum_{i=1}^{k} X_i$.

Notice that $\mathbb{E}[X_i] = \delta$ and so $\mathbb{E}[S_k] = \delta k$.

By the Chernoff bound, we get
$P(S_k \leq (1-\varepsilon)\delta k) \leq \exp\left(\frac{-\varepsilon^2 \delta k}{2}\right) = \exp\left(-\frac{1}{2}\varepsilon^2 \delta k\right)$.

Explaining the Expected Value

Bounding the Error

Is τ too high?

Bad event means there are less than $k - O(\delta k)$ items from $S^{\star}$ that are among the last $(1-\delta)\,n$ items and greater than τ.

Explaining the Expected Value

Bounding the Error

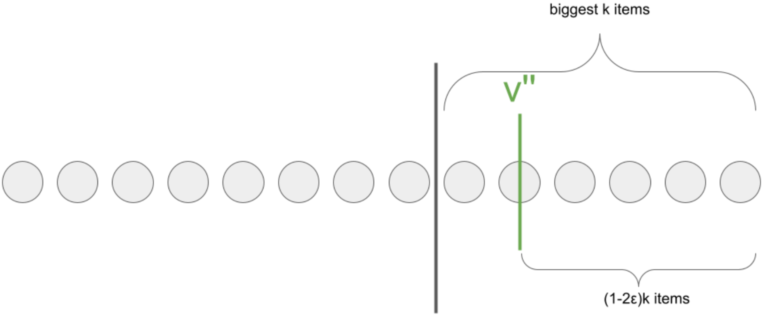
Is τ too high?

Bad event means there are less than $k - O(\delta k)$ items from $S^\star$ that are among the last $(1-\delta)\,n$ items and greater than τ.

Look at $v'' = (1 - 2\varepsilon)\,k^{\text{th}}$-highest value in $S^\star$.

Explaining the Expected Value

Bounding the Error



Explaining the Expected Value

Bounding the Error

What is the probability that $\tau > v''$?

Remember X_i, look at $S_{(1-2\varepsilon)k} = \sum_{i=1}^{(1-2\varepsilon)k} Y_i$ (only items bigger than v'').

Explaining the Expected Value

Bounding the Error

What is the probability that $\tau > v''$?

Remember X_i, look at $S_{(1-2\varepsilon)k} = \sum_{i=1}^{(1-2\varepsilon)k} Y_i$ (only items bigger than v'').

Notice that $\mathbb{E}\left[Y_i\right] = \delta$, and so $\mathbb{E}\left[S_{(1-2\varepsilon)k}\right] = (1 - 2\varepsilon)\,\delta k$.

Explaining the Expected Value

Bounding the Error

What is the probability that $\tau > v''$?

Remember X_i, look at $S_{(1-2\varepsilon)k} = \sum_{i=1}^{(1-2\varepsilon)k} Y_i$ (only items bigger than v'').

Notice that $\mathbb{E}\left[Y_i\right] = \delta$, and so $\mathbb{E}\left[S_{(1-2\varepsilon)k}\right] = (1-2\varepsilon)\,\delta k$.

We are interested in the event $S_{(1-2\varepsilon)k} > (1-\varepsilon)\,\delta k$.

Equivalently: $S_{(1-2\varepsilon)k} > \left(1 + \frac{\varepsilon}{1-2\varepsilon}\right)(1-2\varepsilon)\,\delta k$.

Explaining the Expected Value

Bounding the Error

What is the probability that $\tau > v''$?

Remember X_i, look at $S_{(1-2\varepsilon)k} = \sum_{i=1}^{(1-2\varepsilon)k} Y_i$ (only items bigger than v'').

Notice that $\mathbb{E}\left[Y_i\right] = \delta$, and so $\mathbb{E}\left[S_{(1-2\varepsilon)k}\right] = (1-2\varepsilon)\,\delta k$.

We are interested in the event $S_{(1-2\varepsilon)k} > (1-\varepsilon)\,\delta k$.

Equivalently: $S_{(1-2\varepsilon)k} > \left(1 + \frac{\varepsilon}{1-2\varepsilon}\right)(1-2\varepsilon)\,\delta k$.

From Hoeffding inequality we get:

$$P\left(S_{(1-2\varepsilon)k} > \left(1 + \frac{\varepsilon}{1-2\varepsilon}\right)(1-2\varepsilon)\,\delta k\right) \leq$$

$$\exp\left(\frac{-\left(\frac{\varepsilon}{1-2\varepsilon}\right)^2 (1-2\varepsilon)\delta k}{2+\frac{\varepsilon}{1-2\varepsilon}}\right) = \exp\left(\frac{-\varepsilon^2\delta k}{2-3\varepsilon}\right) \leq \exp\left(-\varepsilon^2\delta k\right)$$

Explaining the Expected Value

Bounding the Error

So we bounded the event that $\tau \leq v''$.

How many items are bigger than v''?

$(1 - 2\varepsilon)\, k = k - 2\varepsilon k \overset{*}{=} k - O(\delta k)$

This means that if $\tau \leq v''$ then we are not too high.

Explaining the Expected Value

Bounding the Error

Why can we use the Hoeffding bound? The choices are not independent...

Explaining the Expected Value

Bounding the Error

Why can we use the Hoeffding bound? The choices are not independent...

2 solutions:

1. Change the algorithm to use "time".

Explaining the Expected Value

Bounding the Error

Why can we use the Hoeffding bound? The choices are not independent...

2 solutions:

1. Change the algorithm to use "time".
2. Don't use the Hoeffding bound...

Explaining the Expected Value

Choosing δ, ε

We want to lose at most $O\left(\delta V^{\star}\right)$ value.

Enough to choose δ, ε so that $\exp\left(-\varepsilon^2\delta^2 k\right) = O\left(\delta\right)$ (we also want $\xrightarrow{k\to\infty} 0$).

This is equivalent to $\varepsilon^2\delta^2 k = O\left(\log\frac{1}{\delta}\right)$.

A clean solution would be $\delta = \varepsilon = \left(\frac{\log k}{k}\right)^{1/4}$.

Then we would get

$$\varepsilon^2\delta^2 k = \left(\left(\frac{\log k}{k}\right)^{1/4}\right)^4 k = \log k = O\left(\log\frac{k}{\log k}\right) = O\left(\log\frac{1}{\delta}\right)$$

Explaining the Expected Value

Choosing δ, ε

We want to lose at most $O\left(\delta V^\star\right)$ value.

Enough to choose δ, ε so that $\exp\left(-\varepsilon^2\delta^2 k\right) = O\left(\delta\right)$ (we also want $\xrightarrow{k\to\infty} 0$).

This is equivalent to $\varepsilon^2\delta^2 k = O\left(\log\frac{1}{\delta}\right)$.

A clean solution would be $\delta = \varepsilon = \left(\frac{\log k}{k}\right)^{1/4}$.

Then we would get

$$\varepsilon^2\delta^2 k = \left(\left(\frac{\log k}{k}\right)^{1/4}\right)^4 k = \log k = O\left(\log\frac{k}{\log k}\right) = O\left(\log\frac{1}{\delta}\right)$$

Explaining the Expected Value

Choosing δ, ε

We want to lose at most $O(\delta V^\star)$ value.

Enough to choose δ, ε so that $\exp\left(-\varepsilon^2\delta^2 k\right) = O(\delta)$ (we also want $\xrightarrow{k\to\infty} 0$).

This is equivalent to $\varepsilon^2\delta^2 k = O\left(\log \frac{1}{\delta}\right)$.

A clean solution would be $\delta = \varepsilon = \left(\frac{\log k}{k}\right)^{1/4}$.

Then we would get

$$\varepsilon^2\delta^2 k = \left(\left(\frac{\log k}{k}\right)^{1/4}\right)^4 k = \log k = O\left(\log \frac{k}{\log k}\right) = O\left(\log \frac{1}{\delta}\right)$$

Explaining the Expected Value

Choosing δ, ε

We want to lose at most $O\left(\delta V^{\star}\right)$ value.

Enough to choose δ, ε so that $\exp\left(-\varepsilon^2\delta^2 k\right) = O\left(\delta\right)$ (we also want $\xrightarrow{k\to\infty} 0$).

This is equivalent to $\varepsilon^2\delta^2 k = O\left(\log\frac{1}{\delta}\right)$.

A clean solution would be $\delta = \varepsilon = \left(\frac{\log k}{k}\right)^{1/4}$.

Then we would get

$$\varepsilon^2\delta^2 k = \left(\left(\frac{\log k}{k}\right)^{1/4}\right)^4 k = \log k = O\left(\log\frac{k}{\log k}\right) = O\left(\log\frac{1}{\delta}\right)$$

Explaining the Expected Value

Choosing δ, ε

We want to lose at most $O(\delta V^\star)$ value.

Enough to choose δ, ε so that $\exp\left(-\varepsilon^2\delta^2 k\right) = O(\delta)$ (we also want $\xrightarrow{k\to\infty} 0$).

This is equivalent to $\varepsilon^2\delta^2 k = O\left(\log \frac{1}{\delta}\right)$.

A clean solution would be $\delta = \varepsilon = \left(\frac{\log k}{k}\right)^{1/4}$.

Then we would get

$$\varepsilon^2\delta^2 k = \left(\left(\frac{\log k}{k}\right)^{1/4}\right)^4 k = \log k = O\left(\log \frac{k}{\log k}\right) = O\left(\log \frac{1}{\delta}\right)$$

Discussion

Is a loss of $k^{1/4}$ of the value the best we can do?

Discussion

Is a loss of $k^{1/4}$ of the value the best we can do?

Question

What would you change, if we don't constrain ourselves to an order-oblivious algorithm?

Order-Adaptive Algorithms

Order-oblivious algorithms are easier to analyze, but they are too limiting.

Order-Adaptive Algorithms

Order-oblivious algorithms are easier to analyze, but they are too limiting.

We want algorithms that can adapt during-execution, and exploit the randomness of the entire sequence.

We call these algorithms **order-adaptive** algorithms.

An Upgrade

Updating the Threshold As We Go

Until now, we ignored the first $\approx k^{-1/4}$ fraction of items, and then set a fixed threshold.

An Upgrade

Updating the Threshold As We Go

Until now, we ignored the first $\approx k^{-1/4}$ fraction of items, and then set a fixed threshold.

The fraction ignored tried to balance 2 measures:

the amount of lost items $\Leftrightarrow$ good estimation of the k^{th} largest item.

An Upgrade

Updating the Threshold As We Go

Until now, we ignored the first $\approx k^{-1/4}$ fraction of items, and then set a fixed threshold.

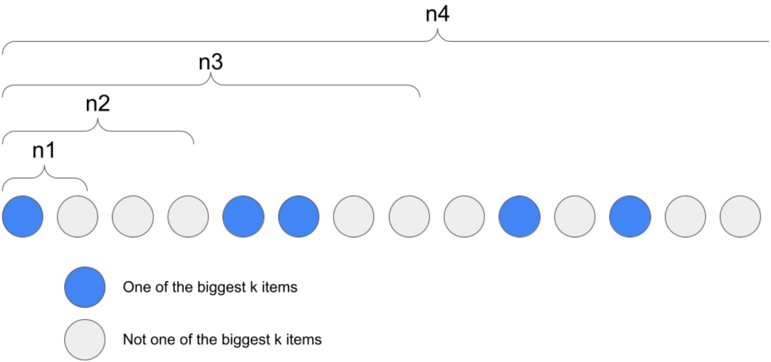
The fraction ignored tried to balance 2 measures:

the amount of lost items $\Leftrightarrow$ good estimation of the k^{th} largest item.

We want to update the threshold as we gain more information.

The Order-Adaptive Algorithm

For the Multiple-Secretary Problem



The Order-Adaptive Algorithm

For the Multiple-Secretary Problem

Order-adaptive algorithm for the multiple-secretary problem

Define $\delta := \sqrt{\frac{\log k}{k}}$ and $n_j := 2^j \delta n$.

1. Ignore the first δn items.
2. For each $j \in \{0, \ldots, \log \frac{1}{\delta}\}$, phase j runs on arrivals in window $W_j := (n_j,\ n_{j+1}]$.
   1. Let $\varepsilon_j := \sqrt{\frac{\delta}{2^j}}$.
   2. Set threshold τ_j to be the $(1 - \varepsilon_j)\, k^{\text{th}}$-largest value among the first n_j items.
   3. Choose any item in window W_j with value above τ_j.

The Order-Adaptive Algorithm

For the Multiple-Secretary Problem

Theorem

The above algorithm has an expected value of $V^\star \cdot \left(1 - O\left(\sqrt{\frac{\log k}{k}}\right)\right)$.

We will not prove this theorem, but it is similar to the way we handled the order-oblivious algorithm (with some union bounds).

A Lower Bound

It turns out the $\sqrt{\log k}$ can be removed, but the loss of $1/\sqrt{k}$ is essential.

More formally: Every algorithm to the multiple-secretary problem will lose at least $V^\star \cdot O\left(1/\sqrt{k}\right)$ value.

A Lower Bound

It turns out the $\sqrt{\log k}$ can be removed, but the loss of $1/\sqrt{k}$ is essential.

More formally: Every algorithm to the multiple-secretary problem will lose at least $V^\star \cdot O\left(1/\sqrt{k}\right)$ value.

Let's see a sketch of why that is.

A Lower Bound

It turns out the $\sqrt{\log k}$ can be removed, but the loss of $1/\sqrt{k}$ is essential.

More formally: Every algorithm to the multiple-secretary problem will lose at least $V^\star \cdot O\left(1/\sqrt{k}\right)$ value.

Let's see a sketch of why that is.

By Yao's minimax lemma, it suffices to give a distribution over instances that causes a large loss for any deterministic algorithm.

A Lower Bound - Cont.

Define a distribution of items as follows:

A Lower Bound - Cont.

Define a distribution of items as follows:

With probability $1 - \frac{k}{n}$, give the item a value of 0.

A Lower Bound - Cont.

Define a distribution of items as follows:

With probability $1 - \frac{k}{n}$, give the item a value of 0.

Otherwise, give it 1 or 2 with equal probability.

A Lower Bound - Cont.

Define a distribution of items as follows:

With probability $1 - \frac{k}{n}$, give the item a value of 0.

Otherwise, give it 1 or 2 with equal probability.

The variance of the amount of non-zero items is
$n \cdot \frac{k}{n}\left(1 - \frac{k}{n}\right) = k - \frac{k^2}{n}$.
So with high probability, the amount of non-zero items is
$k \pm O\left(\sqrt{k}\right)$.

This means $V^\star = \frac{3}{2}k \pm O\left(\sqrt{k}\right)$.

A Lower Bound - Cont.

Optimal solution would take all 2's and fill the remaining $k/2 \pm O\left(\sqrt{k}\right)$ slots with 1's.

A Lower Bound - Cont.

Optimal solution would take all 2's and fill the remaining $k/2 \pm O\left(\sqrt{k}\right)$ slots with 1's.

But an online algorithm doesn't know how many 2's are going to arrive.

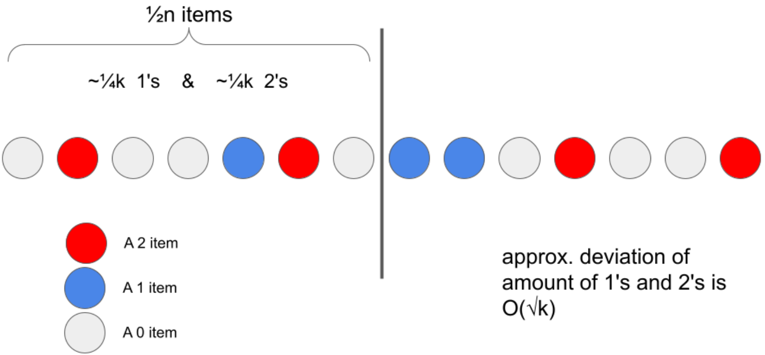
A Lower Bound - Cont.

Optimal solution would take all 2's and fill the remaining $k/2 \pm O\left(\sqrt{k}\right)$ slots with 1's.

But an online algorithm doesn't know how many 2's are going to arrive.

Look at the state of our deterministic algorithm after $n/2$ arrivals.

A Lower Bound - Cont.



A Lower Bound - Cont.

Either we pick too many 1's, and lose $\Theta\left(\sqrt{k}\right)$ 2's in the second half,

or we pick $\Theta\left(\sqrt{k}\right)$ too few 1's in the first half.

A Lower Bound - Cont.

Either we pick too many 1's, and lose $\Theta\left(\sqrt{k}\right)$ 2's in the second half,

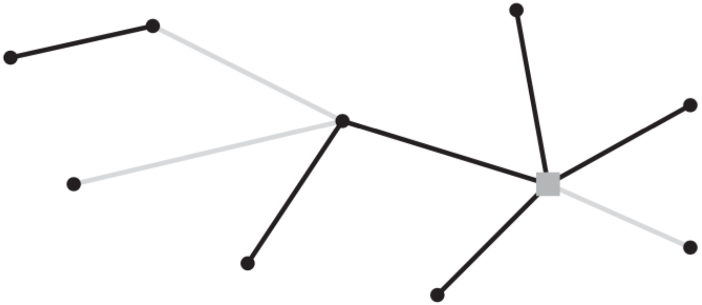
or we pick $\Theta\left(\sqrt{k}\right)$ too few 1's in the first half.

Either way, the algorithm will lose $\Theta\left(\sqrt{k}\right) = \Omega\left(V^{\star}/\sqrt{k}\right)$ value.

Max-Weight Forests

Max-Weight Forests

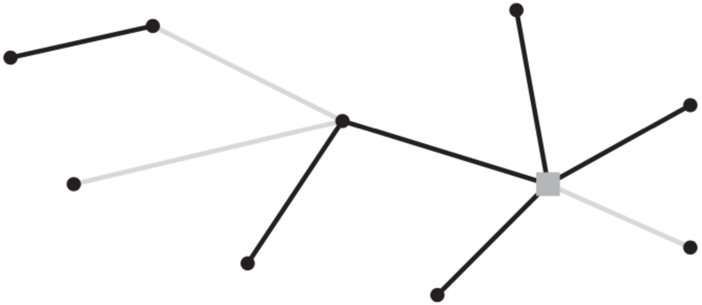
Given a graph $G = (V, E)$, and weights $w : E \to \mathbb{R}^+$, find the forest (acyclic subset of E) with the maximum weight.



Max-Weight Forests

Given a graph $G = (V, E)$, and weights $w : E \to \mathbb{R}^+$, find the forest (acyclic subset of E) with the maximum weight.

In the random-order model, the edges and their weights arrive one by one.

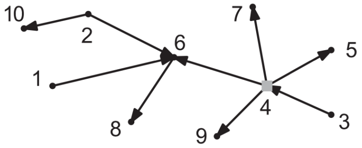


Max-Weight Forests

An Algorithm

1. Choose a uniformly random permutation π of the vertices.

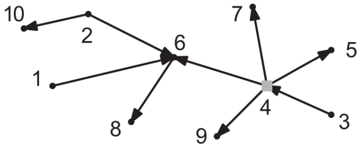
Max-Weight Forests



An Algorithm

1. Choose a uniformly random permutation π of the vertices.
2. For each edge $(u, v) \in E$, direct it from u to v in $\pi(u) < \pi(v)$.

Max-Weight Forests



An Algorithm

1. Choose a uniformly random permutation π of the vertices.
2. For each edge $(u, v) \in E$, direct it from u to v in $\pi(u) < \pi(v)$.
3. Independently for each vertex u, consider the edges directed **towards** u and run the 50%-algorithm on these edges.

Max-Weight Forests

Theorem

This algorithm is 8-competitive.

Max-Weight Forests - Proof

Outline

We need to prove 2 things:

1. The algorithm returns a forest.
2. The expected value of the algorithm is at least $1/8$'th of the optimal value.

Max-Weight Forests - Proof Cont.

The Algorithm Returns a Forest

Assume by contradiction that there is a cycle.

Max-Weight Forests - Proof Cont.

The Algorithm Returns a Forest

Assume by contradiction that there is a cycle.

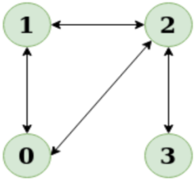
Look at the highest numbered vertex in the cycle (by π), call it $\hat{v}$.

Max-Weight Forests - Proof Cont.

The Algorithm Returns a Forest

Assume by contradiction that there is a cycle.

Look at the highest numbered vertex in the cycle (by π), call it $\hat{v}$.

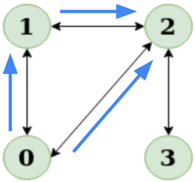


Max-Weight Forests - Proof Cont.

The Algorithm Returns a Forest

Assume by contradiction that there is a cycle.

Look at the highest numbered vertex in the cycle (by π), call it $\hat{v}$.

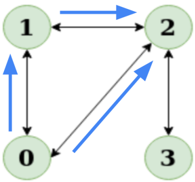


Max-Weight Forests - Proof Cont.

The Algorithm Returns a Forest

Assume by contradiction that there is a cycle.

Look at the highest numbered vertex in the cycle (by π), call it $\hat{v}$.



We chose at most 1 edge pointing to $\hat{v}$, thus contradicting the existance of such circle.

Max-Weight Forests - Proof Cont.

Expected Value is $^1/_8$'th

Since we limit our choice (one incoming edge per vertex), the optimal max-weight might not be feasible.

Max-Weight Forests - Proof Cont.

Expected Value is $^1/_8$'th

Since we limit our choice (one incoming edge per vertex), the optimal max-weight might not be feasible.

Despite this, we claim there is a forest with the one-incoming-edge-per-vertex restriction, and expected value $V^\star/2$.

(Randomness over the permutation)

Proved in a moment - assume for now.

Max-Weight Forests - Proof Cont.

Expected Value is $1/8$'th

Since we limit our choice (one incoming edge per vertex), the optimal max-weight might not be feasible.

Despite this, we claim there is a forest with the one-incoming-edge-per-vertex restriction, and expected value $V^\star/2$.

(Randomness over the permutation)

Proved in a moment - assume for now.

The 50%-algorithm will get $1/4$ of the maximum possible weight for each vertex.

Summing up over all vertices, we get an expected value of $V^\star\frac{1}{2}\cdot\frac{1}{4} = V^\star\frac{1}{8}$ as desired.

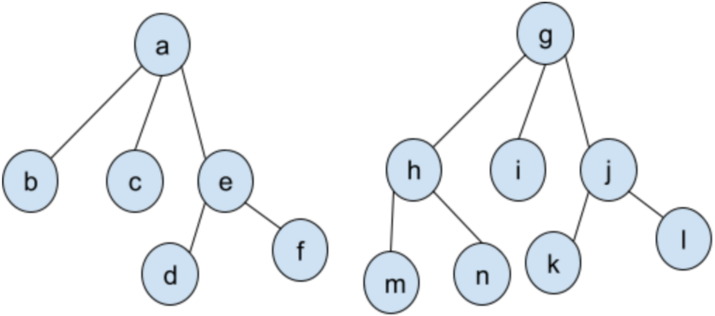
Max-Weight Forests - Proof Cont.

Expected Value is $1/8$'th

Let's prove the expected value of the feasible forest:

Max-Weight Forests - Proof Cont.

Expected Value is 1/8'th



Forest

Max-Weight Forests - Proof Cont.

Expected Value is $1/8$'th

Let's prove the expected value of the feasible forest:

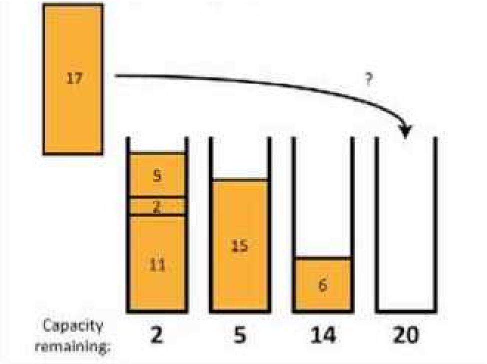
- Choose an arbitrary root for each component in $S^\star$
- and associate each non-root vertex u with the unique edge $e(u)$ of the undirected graph on the path towards the root.
- In our algorithm, for each vertex u, the edge $e(u) = (u, v)$ can be chosen if $\pi(v) < \pi(u)$ (we direct it **into** u).
- This event happens with probability $1/2$ for each vertex, and the claim follows by linearity of expectation.

Max-Weight Forests

We can use the $1/e$-algorithm instead of the 50%-algorithm and get an expected value of $V^\star/2e$.

Bin Packing

Bin Packing



Bin Packing

Definitions

- Each bin is of capacity 1.
- For all $1 \leq i \leq n$, it holds that $s_i \leq 1$.

Bin Packing

An Online Algorithm

Bin Packing

An Online Algorithm

Algorithm: Best-Fit

Bin Packing

An Online Algorithm

Algorithm: Best-Fit

Given the next request with size s_t:

1. If the item does not fit in any currently used bin, put it in a new bin.
2. Else, put into a bin where the resulting empty space is minimized (i.e., where it fits “best”).

Best Fit

Worst Case Cost

OPT must use at least $\lceil \sum s_i \rceil$ bins, because each bin is of unit size.

Best Fit

Worst Case Cost

OPT must use at least $\lceil \sum s_i \rceil$ bins, because each bin is of unit size.

The sum of 2 bins > 1, otherwise we would have never started the second bin.

$\lceil \sum s_i \rceil$ can be considered as "the total weight" and each 2 bins take in at least 1 "weight unit".

So $\lceil 2 \cdot \sum s_i \rceil$ is the maximal amount of bins needed.

Best Fit

Worst Case Cost

OPT must use at least $\lceil \sum s_i \rceil$ bins, because each bin is of unit size.

The sum of 2 bins > 1, otherwise we would have never started the second bin.

$\lceil \sum s_i \rceil$ can be considered as "the total weight" and each 2 bins take in at least 1 "weight unit".

So $\lceil 2 \cdot \sum s_i \rceil$ is the maximal amount of bins needed.

Thus we use no more than $2 \cdot OPT$ in the worst case.

Best Fit

Lower Bound

A sophisticated analysis shows that BEST FIT uses at most $1.7 \cdot OPT + O(1)$ bins, and this multiplicative factor of 1.7 is the best possible.

Best Fit

Lower Bound

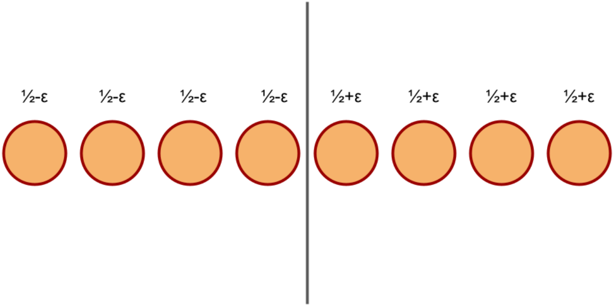
A sophisticated analysis shows that BEST FIT uses at most $1.7 \cdot OPT + O(1)$ bins, and this multiplicative factor of 1.7 is the best possible.

The example showing the lower bound (why this is the "best possible") of $1.7 \cdot OPT + O(1)$ is complex.

We will show an easier lower bound of 1.5, which also highlights why the algorithm does better in the random-order model.

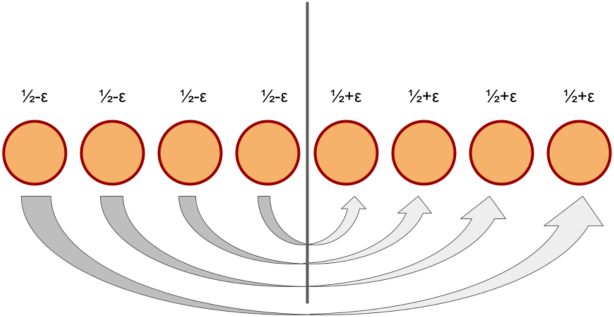
Best Fit - Lower Bound

Example



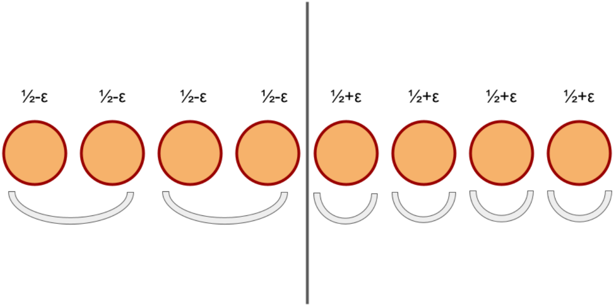
Best Fit - Lower Bound

Example - Optimal Solution



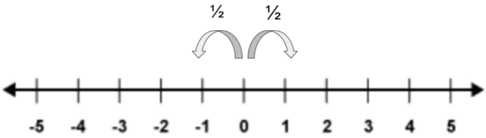
Best Fit - Lower Bound

Example - Adversarial Order



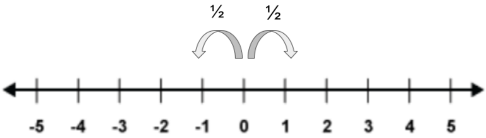
Best Fit - Random Order

Random Walk Equivalent



Best Fit - Random Order

Random Walk Equivalent



Conditioned on starting and ending at the origin.

Best Fit - Random Order

Calculations and Results

The number of $1/2 + \varepsilon$ items that occupy a bin by themselves can be bounded in terms of the maximum deviation from the origin.

Best Fit - Random Order

Calculations and Results

The number of $1/2 + \varepsilon$ items that occupy a bin by themselves can be bounded in terms of the maximum deviation from the origin.

This deviation is bounded by $O\left(\sqrt{n \cdot \log n}\right) = o\left(OPT\right)$ with high probability (tends to 1 as $n \to \infty$).

Best Fit - Random Order

Calculations and Results

The number of $1/2 + \varepsilon$ items that occupy a bin by themselves can be bounded in terms of the maximum deviation from the origin.

This deviation is bounded by $O\left(\sqrt{n \cdot \log n}\right) = o\left(OPT\right)$ with high probability (tends to 1 as $n \to \infty$).

Corollary

The algorithm uses only $(1 + o(1)) \cdot OPT$ bins on this instance.

Best Fit - Random Order

The General Theorem

Theorem

The Best-Fit algorithm uses at most $(1.5 + o(1)) \cdot OPT$ *bins in the random-order setting.*

Summary

Summary

- What is Random-Order?

Summary

- What is Random-Order?
- Why Random-Order?

Summary

- What is Random-Order?
- Why Random-Order?
- Amount of randomness

Summary

- What is Random-Order?
- Why Random-Order?
- Amount of randomness
- The Secretary Problem from multiple angles

Summary

- What is Random-Order?
- Why Random-Order?
- Amount of randomness
- The Secretary Problem from multiple angles
- Max Weight Forests

Summary

- What is Random-Order?
- Why Random-Order?
- Amount of randomness
- The Secretary Problem from multiple angles
- Max Weight Forests
- Example of a minimization problem - Bin Packing

Summary

Thank you for listening.