Topics in Algorithmic Game Theory and Economics

Pieter Kleer

Max Planck Institute for Informatics
Saarland Informatics Campus

January 6, 2020

Lecture 7
Online Selection Problems

(Offline) selection problems

Selection problem

Given is

Selection problem

Given is

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.

Selection problem

Given is

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.

Selection problem

Given is

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of (downward-closed) feasible subsets

$$\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}.$$

Selection problem

Given is

- Finite set of elements $E = \{e_1, \dots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of (downward-closed) feasible subsets

$$\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}.$$

Goal: Compute (in poly-time) feasible subset $X \subseteq E$ maximizing $w(X) := \sum_{e \in X} w(e)$.

Selection problem

Given is

- Finite set of elements $E = \{e_1, \dots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of (downward-closed) feasible subsets

$$\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}.$$

Goal: Compute (in poly-time) feasible subset $X \subseteq E$ maximizing $w(X) := \sum_{e \in X} w(e)$.

$$\begin{aligned} \max \quad & \sum_{e \in X} w(e) \\ \text{subject to} \quad & X \in \mathcal{F}. \end{aligned}$$

Selection problem

Given is

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of (downward-closed) feasible subsets

$$\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}.$$

Goal: Compute (in poly-time) feasible subset $X \subseteq E$ maximizing $w(X) := \sum_{e \in X} w(e)$.

$$\begin{array}{ll} \max & \sum_{e \in X} w(e) \\ \text{subject to} & X \in \mathcal{F}. \end{array}$$

Combinatorial examples of $\mathcal{F}$:

Selection problem

Given is

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of (downward-closed) feasible subsets

$$\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}.$$

Goal: Compute (in poly-time) feasible subset $X \subseteq E$ maximizing $w(X) := \sum_{e \in X} w(e)$.

$$\begin{array}{ll} \max & \sum_{e \in X} w(e) \\ \text{subject to} & X \in \mathcal{F}. \end{array}$$

Combinatorial examples of $\mathcal{F}$:

- Spanning trees in given graph;

Selection problem

Given is

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of (downward-closed) feasible subsets

$$\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}.$$

Goal: Compute (in poly-time) feasible subset $X \subseteq E$ maximizing $w(X) := \sum_{e \in X} w(e)$.

$$\begin{aligned} \max \quad & \sum_{e \in X} w(e) \\ \text{subject to} \quad & X \in \mathcal{F}. \end{aligned}$$

Combinatorial examples of $\mathcal{F}$:

- Spanning trees in given graph;
- Bases of a matroid;

Selection problem

Given is

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of (downward-closed) feasible subsets

$$\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}.$$

Goal: Compute (in poly-time) feasible subset $X \subseteq E$ maximizing $w(X) := \sum_{e \in X} w(e)$.

$$\begin{array}{ll} \max & \sum_{e \in X} w(e) \\ \text{subject to} & X \in \mathcal{F}. \end{array}$$

Combinatorial examples of $\mathcal{F}$:

- Spanning trees in given graph;
- Bases of a matroid;
- Matchings in given (bipartite) graph.

Some examples

Maximum weight spanning tree

Maximum weight spanning tree

Example (Spanning tree)

Given is undirected graph $G = (V, E)$ with edge-weight function $w : E \rightarrow \mathbb{R}_{\geq 0}$.

Maximum weight spanning tree

Example (Spanning tree)

Given is undirected graph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}_{\geq 0}$. Feasible sets in $\mathcal{F}$ are spanning trees of G.

Maximum weight spanning tree

Example (Spanning tree)

Given is undirected graph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}_{\geq 0}$. Feasible sets in $\mathcal{F}$ are spanning trees of G.

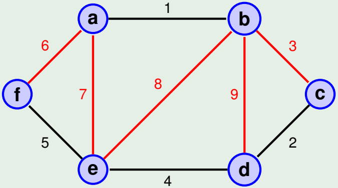
- Subgraph connecting all nodes and not having cycles.

Maximum weight spanning tree

Example (Spanning tree)

Given is undirected graph $G = (V, E)$ with edge-weight function $w : E \rightarrow \mathbb{R}_{\geq 0}$. Feasible sets in $\mathcal{F}$ are spanning trees of G.

- Subgraph connecting all nodes and not having cycles.

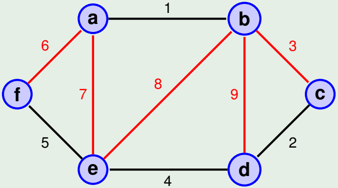


Maximum weight spanning tree

Example (Spanning tree)

Given is undirected graph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}_{\geq 0}$. Feasible sets in $\mathcal{F}$ are spanning trees of G.

- Subgraph connecting all nodes and not having cycles.



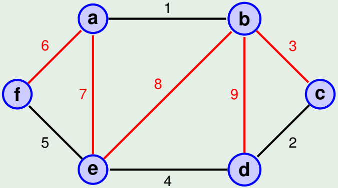
More general, compute maximum weight base of matroid.

Maximum weight spanning tree

Example (Spanning tree)

Given is undirected graph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}_{\geq 0}$. Feasible sets in $\mathcal{F}$ are spanning trees of G.

- Subgraph connecting all nodes and not having cycles.



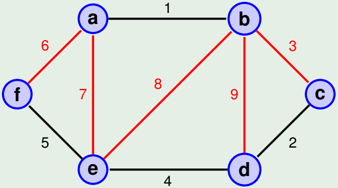
More general, compute maximum weight base of matroid.

Remember that $\mathcal{M} = (E, \mathcal{F})$ is matroid if:

Example (Spanning tree)

Given is undirected graph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}_{\geq 0}$. Feasible sets in $\mathcal{F}$ are spanning trees of G.

- Subgraph connecting all nodes and not having cycles.



More general, compute maximum weight base of matroid.

Remember that $\mathcal{M} = (E, \mathcal{F})$ is matroid if:

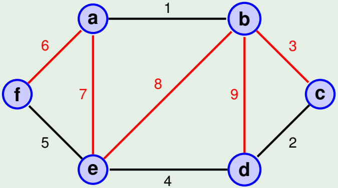
- *Downward-closed*: $A \in \mathcal{F}$ and $B \subseteq A \Rightarrow B \in \mathcal{F}$,

Maximum weight spanning tree

Example (Spanning tree)

Given is undirected graph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}_{\geq 0}$. Feasible sets in $\mathcal{F}$ are spanning trees of G.

- Subgraph connecting all nodes and not having cycles.



More general, compute maximum weight base of matroid.

Remember that $\mathcal{M} = (E, \mathcal{F})$ is matroid if:

- *Downward-closed*: $A \in \mathcal{F}$ and $B \subseteq A \Rightarrow B \in \mathcal{F}$,
- *Augmentation property*:

  $A, C \in \mathcal{F}$ and $|C| > |A| \Rightarrow \exists e \in C \setminus A$ such that $A \cup \{e\} \in \mathcal{F}$.

Greedy algorithm

Rename edges such that $w_1 \geq w_2 \geq \cdots \geq w_m \geq 0$.

Greedy algorithm

Rename edges such that $w_1 \geq w_2 \geq \cdots \geq w_m \geq 0$.

Greedy algorithm (for spanning trees)

Set $X = \emptyset$.

Greedy algorithm

Rename edges such that $w_1 \geq w_2 \geq \cdots \geq w_m \geq 0$.

Greedy algorithm (for spanning trees)

Set $X = \emptyset$. For $i = 1, \ldots, m$:

Greedy algorithm

Rename edges such that $w_1 \geq w_2 \geq \cdots \geq w_m \geq 0$.

Greedy algorithm (for spanning trees)

Set $X = \emptyset$. For $i = 1, \ldots, m$:

- If $X + e_i$ does not contain a cycle, set $X = X + e_i$.

Greedy algorithm

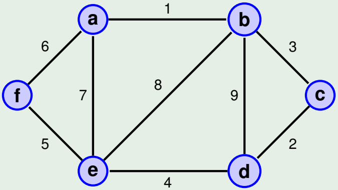
Rename edges such that $w_1 \geq w_2 \geq \cdots \geq w_m \geq 0$.

Greedy algorithm (for spanning trees)

Set $X = \emptyset$. For $i = 1, \ldots, m$:

- If $X + e_i$ does not contain a cycle, set $X = X + e_i$.

Example (cont'd)



Greedy algorithm

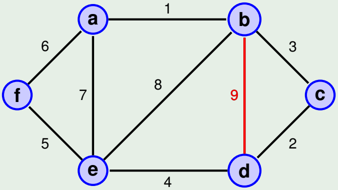
Rename edges such that $w_1 \geq w_2 \geq \cdots \geq w_m \geq 0$.

Greedy algorithm (for spanning trees)

Set $X = \emptyset$. For $i = 1, \ldots, m$:

- If $X + e_i$ does not contain a cycle, set $X = X + e_i$.

Example (cont'd)



Greedy algorithm

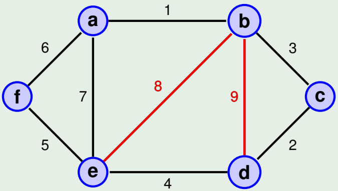
Rename edges such that $w_1 \geq w_2 \geq \cdots \geq w_m \geq 0$.

Greedy algorithm (for spanning trees)

Set $X = \emptyset$. For $i = 1, \ldots, m$:

- If $X + e_i$ does not contain a cycle, set $X = X + e_i$.

Example (cont'd)



Greedy algorithm

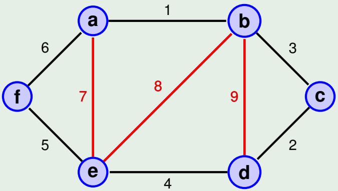
Rename edges such that $w_1 \geq w_2 \geq \cdots \geq w_m \geq 0$.

Greedy algorithm (for spanning trees)

Set $X = \emptyset$. For $i = 1, \ldots, m$:

- If $X + e_i$ does not contain a cycle, set $X = X + e_i$.

Example (cont'd)



Greedy algorithm

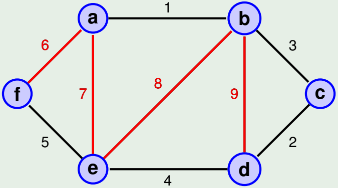
Rename edges such that $w_1 \geq w_2 \geq \cdots \geq w_m \geq 0$.

Greedy algorithm (for spanning trees)

Set $X = \emptyset$. For $i = 1, \ldots, m$:

- If $X + e_i$ does not contain a cycle, set $X = X + e_i$.

Example (cont'd)



Greedy algorithm

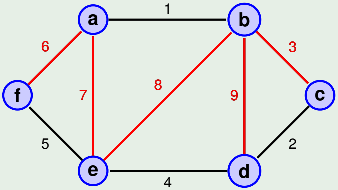
Rename edges such that $w_1 \geq w_2 \geq \cdots \geq w_m \geq 0$.

Greedy algorithm (for spanning trees)

Set $X = \emptyset$. For $i = 1, \ldots, m$:

- If $X + e_i$ does not contain a cycle, set $X = X + e_i$.

Example (cont'd)



Greedy algorithm

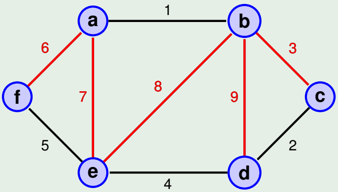
Rename edges such that $w_1 \geq w_2 \geq \cdots \geq w_m \geq 0$.

Greedy algorithm (for spanning trees)

Set $X = \emptyset$. For $i = 1, \ldots, m$:

- If $X + e_i$ does not contain a cycle, set $X = X + e_i$.

Example (cont'd)

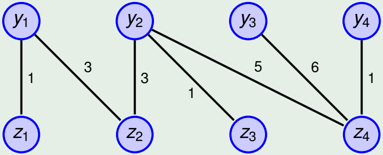


As the weights are non-negative, the output of the greedy algorithm is always a maximum weight spanning tree.

Maximum weight bipartite matching

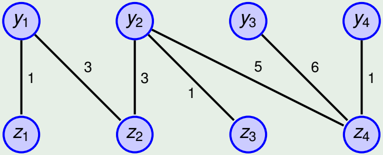
Maximum weight bipartite matching

Example



Maximum weight bipartite matching

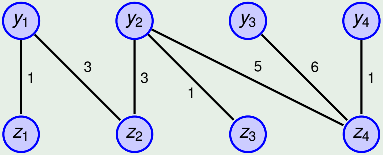
Example



Given bipartite graph $B = (Y \cup Z, E)$ with $E = \{\{y, z\} : y \in Y, z \in Z\}$.

Maximum weight bipartite matching

Example

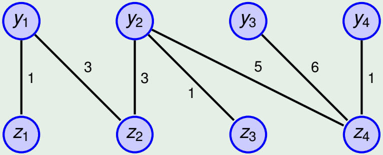


Given bipartite graph $B = (Y \cup Z, E)$ with $E = \{\{y, z\} : y \in Y, z \in Z\}$.

- Edge weight function $w : E \to \mathbb{R}_{\geq 0}$.

Maximum weight bipartite matching

Example

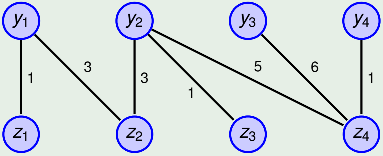


Given bipartite graph $B = (Y \cup Z, E)$ with $E = \{\{y, z\} : y \in Y, z \in Z\}$.

- Edge weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Feasible sets are (bipartite) matchings of B.

Maximum weight bipartite matching

Example

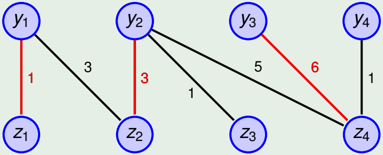


Given bipartite graph $B = (Y \cup Z, E)$ with $E = \{\{y, z\} : y \in Y, z \in Z\}$.

- Edge weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Feasible sets are (bipartite) matchings of B.
  - Matching $M \subseteq E$ is set of edges where every node is incident to at most one edge from M:

Maximum weight bipartite matching

Example

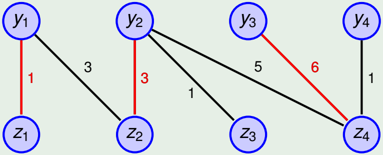


Given bipartite graph $B = (Y \cup Z, E)$ with $E = \{\{y, z\} : y \in Y, z \in Z\}$.

- Edge weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Feasible sets are (bipartite) matchings of B.
  - Matching $M \subseteq E$ is set of edges where every node is incident to at most one edge from M: $|\{e \in M : e \cap \{v\}\}| \leq 1 \; \forall v \in Y \cup Z$.

Maximum weight bipartite matching

Example

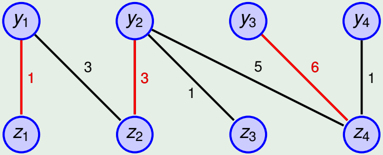


Given bipartite graph $B = (Y \cup Z, E)$ with $E = \{\{y, z\} : y \in Y, z \in Z\}$.

- Edge weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Feasible sets are (bipartite) matchings of B.
  - Matching $M \subseteq E$ is set of edges where every node is incident to at most one edge from M: $|\{e \in M : e \cap \{v\}\}| \leq 1 \; \forall v \in Y \cup Z$.

Maximum weight bipartite matching

Example



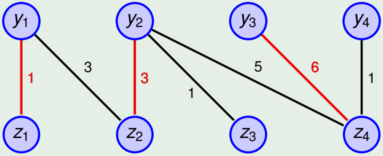
Given bipartite graph $B = (Y \cup Z, E)$ with $E = \{\{y, z\} : y \in Y, z \in Z\}$.

- Edge weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Feasible sets are (bipartite) matchings of B.
  - Matching $M \subseteq E$ is set of edges where every node is incident to at most one edge from M: $|\{e \in M : e \cap \{v\}\}| \leq 1 \; \forall v \in Y \cup Z$.

Poly-time algorithms known for solving maximum matching problem.

Maximum weight bipartite matching

Example



Given bipartite graph $B = (Y \cup Z, E)$ with $E = \{\{y, z\} : y \in Y, z \in Z\}$.

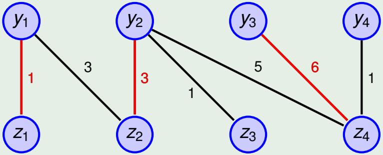
- Edge weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Feasible sets are (bipartite) matchings of B.
  - Matching $M \subseteq E$ is set of edges where every node is incident to at most one edge from M: $|\{e \in M : e \cap \{v\}\}| \leq 1 \; \forall v \in Y \cup Z$.

Poly-time algorithms known for solving maximum matching problem.

- Linear programming can be used.

Maximum weight bipartite matching

Example



Given bipartite graph $B = (Y \cup Z, E)$ with $E = \{\{y, z\} : y \in Y, z \in Z\}$.

- Edge weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Feasible sets are (bipartite) matchings of B.
  - Matching $M \subseteq E$ is set of edges where every node is incident to at most one edge from M: $|\{e \in M : e \cap \{v\}\}| \leq 1 \ \forall v \in Y \cup Z$.

Poly-time algorithms known for solving maximum matching problem.

- Linear programming can be used.
- Well-known combinatorial algorithm: Hungarian method.

Approximation algorithms

Sometimes solving the program

$$\begin{aligned} \mathrm{OPT} = \quad & \max & & \sum_{e \in X} w(e) \\ & \text{subject to} & & X \in \mathcal{F}. \end{aligned}$$

is NP-complete.

Approximation algorithms

Sometimes solving the program

$$\begin{aligned} \mathrm{OPT} = \quad \max \quad & \sum_{e \in X} w(e) \\ \text{subject to} \quad & X \in \mathcal{F}. \end{aligned}$$

is NP-complete.

- For example, Traveling Salesman Problem (TSP).

Approximation algorithms

Sometimes solving the program

$$\begin{aligned} \mathrm{OPT} = \quad & \max & & \sum_{e \in X} w(e) \\ & \text{subject to} & & X \in \mathcal{F}. \end{aligned}$$

is NP-complete.

- For example, Traveling Salesman Problem (TSP).

Therefore, we also study (constant-factor) α-approximation algorithms:

Approximation algorithms

Sometimes solving the program

$$\begin{aligned} \mathrm{OPT} = \quad & \max & & \sum_{e \in X} w(e) \\ & \text{subject to} & & X \in \mathcal{F}. \end{aligned}$$

is NP-complete.

- For example, Traveling Salesman Problem (TSP).

Therefore, we also study (constant-factor) α-approximation algorithms:

> **Goal:** Compute (in poly-time) $X \in \mathcal{F}$ such that
>
> $$w(X) := \sum_{e \in X} w(e) \geq \alpha \cdot \mathrm{OPT}.$$

Computational aspects

Given is

- Finite set of elements $E = \{e_1, \ldots, e_m\}$,
- Weight function $w : E \to \mathbb{R}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

$$\begin{array}{ll} \text{max} & \sum_{e \in X} w(e) \\ \text{subject to} & X \in \mathcal{F}. \end{array}$$

Computational aspects

Given is

- Finite set of elements $E = \{e_1, \ldots, e_m\}$,
- Weight function $w : E \to \mathbb{R}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

$$\begin{array}{ll} \max & \sum_{e \in X} w(e) \\ \text{subject to} & X \in \mathcal{F}. \end{array}$$

Remark

In general, we assume to have a feasibility oracle for $\mathcal{F}$:

Computational aspects

Given is

- Finite set of elements $E = \{e_1, \ldots, e_m\}$,
- Weight function $w : E \to \mathbb{R}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

$$\begin{array}{ll} \max & \sum_{e \in X} w(e) \\ \text{subject to} & X \in \mathcal{F}. \end{array}$$

Remark

In general, we assume to have a feasibility oracle for $\mathcal{F}$: Given $X \subset E$, oracle tells us whether $X \in \mathcal{F}$ or not, i.e., whether X is feasible.

Computational aspects

Given is

- Finite set of elements $E = \{e_1, \ldots, e_m\}$,
- Weight function $w : E \to \mathbb{R}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

$$\begin{array}{ll} \max & \sum_{e \in X} w(e) \\ \text{subject to} & X \in \mathcal{F}. \end{array}$$

Remark

In general, we assume to have a feasibility oracle for $\mathcal{F}$: Given $X \subset E$, oracle tells us whether $X \in \mathcal{F}$ or not, i.e., whether X is feasible.

- Computational complexity measured in terms of m, representation size of weights $w(e)$, and number of oracle calls.

Online selection problems

Online selection problem

Consider

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

Online selection problem

Consider

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

Online selection problem

Online selection problem

Consider

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \ldots, \sigma(m))$.

Online selection problem

Consider

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \ldots, \sigma(m))$.

- *Permutation of elements, e.g.,* (e_4, e_2, e_1, e_3).

Online selection problem

Consider

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \ldots, \sigma(m))$.

- *Permutation of elements, e.g.,* (e_4, e_2, e_1, e_3).

For $i = 1, \ldots, m$, upon arrival of element $\sigma(i)$:

Online selection problem

Consider

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \ldots, \sigma(m))$.

- *Permutation of elements, e.g., (e_4, e_2, e_1, e_3).*

For $i = 1, \ldots, m$, upon arrival of element $\sigma(i)$:

- Weight $w_{\sigma(i)}$ is revealed.

Online selection problem

Consider

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \ldots, \sigma(m))$.

- *Permutation of elements, e.g., (e_4, e_2, e_1, e_3).*

For $i = 1, \ldots, m$, upon arrival of element $\sigma(i)$:

- Weight $w_{\sigma(i)}$ is revealed.
- Decide (irrevocably) whether to select or reject $\sigma(i)$.

Online selection problem

Consider

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \ldots, \sigma(m))$.

- *Permutation of elements, e.g.,* (e_4, e_2, e_1, e_3).

For $i = 1, \ldots, m$, upon arrival of element $\sigma(i)$:

- Weight $w_{\sigma(i)}$ is revealed.
- Decide (irrevocably) whether to select or reject $\sigma(i)$.

Goal: *Select feasible subset* $X \in \mathcal{F}$ *maximizing* $w(X) = \sum_{e \in X} w(e)$.

Online selection problem

Consider

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \ldots, \sigma(m))$.

- *Permutation of elements, e.g., (e_4, e_2, e_1, e_3).*

For $i = 1, \ldots, m$, upon arrival of element $\sigma(i)$:

- Weight $w_{\sigma(i)}$ is revealed.
- Decide (irrevocably) whether to select or reject $\sigma(i)$.

Goal: *Select feasible subset $X \in \mathcal{F}$ maximizing $w(X) = \sum_{e \in X} w(e)$.*

- Algorithm knows $\mathcal{F}$ up front (or has oracle access)

Online selection problem

Consider

- Finite set of elements $E = \{e_1, \ldots, e_m\}$.
- Weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Collection of feasible subsets $\mathcal{F} \subseteq 2^E = \{X : X \subseteq E\}$.

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \ldots, \sigma(m))$.

- *Permutation of elements, e.g., (e_4, e_2, e_1, e_3).*

For $i = 1, \ldots, m$, upon arrival of element $\sigma(i)$:

- Weight $w_{\sigma(i)}$ is revealed.
- Decide (irrevocably) whether to select or reject $\sigma(i)$.

Goal: *Select feasible subset $X \in \mathcal{F}$ maximizing $w(X) = \sum_{e \in X} w(e)$.*

- Algorithm knows $\mathcal{F}$ up front (or has oracle access)
- It has to base decisions only on elements and weights seen so far!

Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

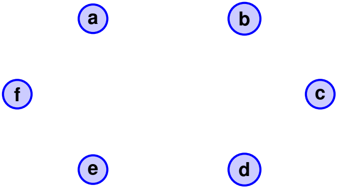
Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

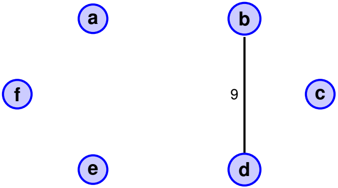


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

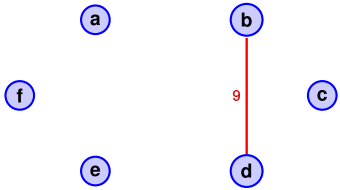


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

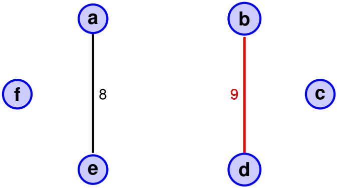


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

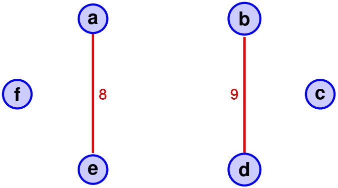


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

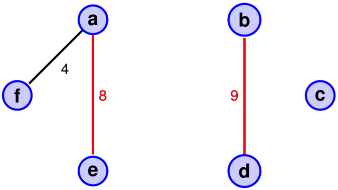


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

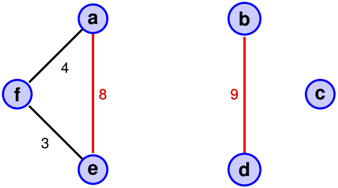


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

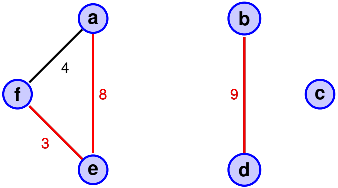


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

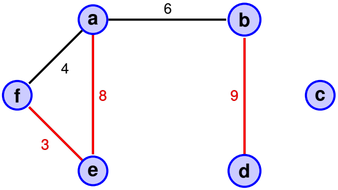


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

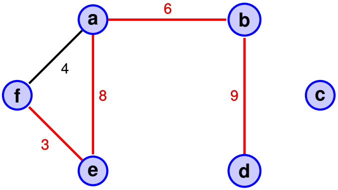


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

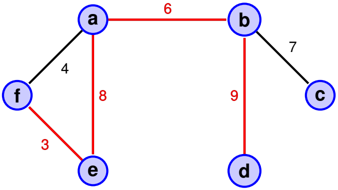


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

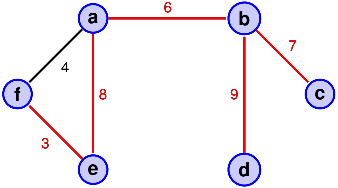


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

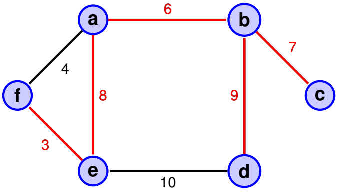


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

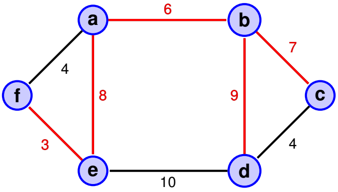


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

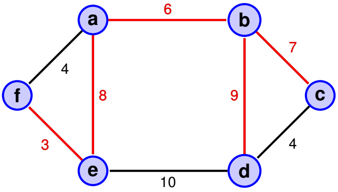


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).

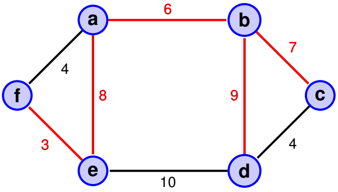


Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).



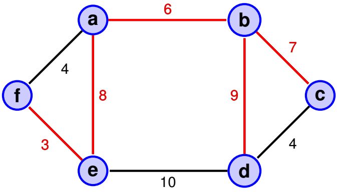
- In general, output might not be spanning tree!

Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).



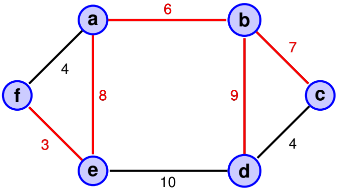
- In general, output might not be spanning tree!
  - We might reject too many edges.

Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).



- In general, output might not be spanning tree!
  - We might reject too many edges.

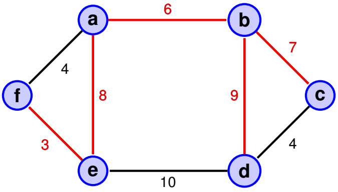
Without any prior knowledge of arrival order or element weights (that still have to arrive),

Example: Online spanning tree selection

Given is undirected graph $G = (V, E)$.

- Edges in E arrive one by one in unknown order.

Want to select collection of edges of maximum weight that does not contain a cycle (i.e., a spanning tree).



- In general, output might not be spanning tree!
  - We might reject too many edges.

Without any prior knowledge of arrival order or element weights (that still have to arrive), not much is possible algorithmically.

Online selection problems

Uniform random arrivals

Uniform random arrival order

Elements arrive in uniform random order σ.

Uniform random arrival order

Elements arrive in uniform random order σ.

- There are $m! = m(m-1)\cdots 1$ orderings.

Uniform random arrival order

Elements arrive in uniform random order σ.

- There are $m! = m(m-1)\cdots 1$ orderings. Probability that elements arrive in order σ is $\frac{1}{m!}$ for every σ.

Uniform random arrival order

Elements arrive in uniform random order σ.

- There are $m! = m(m-1)\cdots 1$ orderings. Probability that elements arrive in order σ is $\frac{1}{m!}$ for every σ.

Example

Suppose we have $E = \{a, b, c\}$,

Uniform random arrival order

Elements arrive in uniform random order σ.

- There are $m! = m(m-1)\cdots 1$ orderings. Probability that elements arrive in order σ is $\frac{1}{m!}$ for every σ.

Example

Suppose we have $E = \{a, b, c\}$, then possible arrival orderings are

$$\sigma \in \{(a, b, c), (a, c, b), (b, a, c), (b, c, a), (c, a, b), (c, b, a)\}.$$

Uniform random arrival order

Elements arrive in uniform random order σ.

- There are $m! = m(m-1)\cdots 1$ orderings. Probability that elements arrive in order σ is $\frac{1}{m!}$ for every σ.

Example

Suppose we have $E = \{a, b, c\}$, then possible arrival orderings are

$$\sigma \in \{(a, b, c), (a, c, b), (b, a, c), (b, c, a), (c, a, b), (c, b, a)\}.$$

Probability of seeing given ordering σ is $1/6$.

Uniform random arrival order

Elements arrive in uniform random order σ.

- There are $m! = m(m-1)\cdots 1$ orderings. Probability that elements arrive in order σ is $\frac{1}{m!}$ for every σ.

Example

Suppose we have $E = \{a, b, c\}$, then possible arrival orderings are

$$\sigma \in \{(a, b, c), (a, c, b), (b, a, c), (b, c, a), (c, a, b), (c, b, a)\}.$$

Probability of seeing given ordering σ is $1/6$.

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_\sigma[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \text{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\text{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum.
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

Uniform random arrival order

Elements arrive in uniform random order σ.

- There are $m! = m(m-1)\cdots 1$ orderings. Probability that elements arrive in order σ is $\frac{1}{m!}$ for every σ.

Example

Suppose we have $E = \{a, b, c\}$, then possible arrival orderings are

$$\sigma \in \{(a, b, c), (a, c, b), (b, a, c), (b, c, a), (c, a, b), (c, b, a)\}.$$

Probability of seeing given ordering σ is $1/6$.

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_\sigma[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \text{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\text{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum.
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

- Polynomial running time is also desired, although not required.

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_\sigma[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \mathrm{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\mathrm{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_\sigma[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \mathrm{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\mathrm{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

- Polynomial running time is also desired, although not required.

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_\sigma[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \text{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\text{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

- Polynomial running time is also desired, although not required.

For offline problem:

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_\sigma[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \text{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\text{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

- Polynomial running time is also desired, although not required.

For offline problem:

- There is trivial $O(2^m)$ time algorithm for computing OPT.

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_\sigma[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \text{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\text{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

- Polynomial running time is also desired, although not required.

For offline problem:

- There is trivial $O(2^m)$ time algorithm for computing OPT.
  - Simply check weight of every $X \in \mathcal{F} = \{X : X \subseteq E\}$.

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_{\sigma}[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \text{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\text{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

- Polynomial running time is also desired, although not required.

For offline problem:

- There is trivial $O(2^m)$ time algorithm for computing OPT.
  - Simply check weight of every $X \in \mathcal{F} = \{X : X \subseteq E\}$.
  - Remember that $|E| = m$.

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_\sigma[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \mathrm{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\mathrm{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

- Polynomial running time is also desired, although not required.

For offline problem:

- There is trivial $O(2^m)$ time algorithm for computing OPT.
  - Simply check weight of every $X \in \mathcal{F} = \{X : X \subseteq E\}$.
  - Remember that $|E| = m$.
- Therefore, goal is to find poly-time algorithm for OPT.

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_\sigma[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \text{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\text{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

- Polynomial running time is also desired, although not required.

For offline problem:

- There is trivial $O(2^m)$ time algorithm for computing OPT.
  - Simply check weight of every $X \in \mathcal{F} = \{X : X \subseteq E\}$.
  - Remember that $|E| = m$.
- Therefore, goal is to find poly-time algorithm for OPT.

For online problem:

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_\sigma[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \text{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\text{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

- Polynomial running time is also desired, although not required.

For offline problem:

- There is trivial $O(2^m)$ time algorithm for computing OPT.
  - Simply check weight of every $X \in \mathcal{F} = \{X : X \subseteq E\}$.
  - Remember that $|E| = m$.
- Therefore, goal is to find poly-time algorithm for OPT.

For online problem:

- (Exponential time) algorithm choosing, for every ordering σ, an X with maximum weight is impossible.

Goal: Give a (possibly randomized) α-approximation $\mathcal{A}$, i.e.,

$$\mathbb{E}_\sigma[w(\mathcal{A}(\sigma))] \geq \alpha \cdot \text{OPT}$$

Ideally, $0 < \alpha < 1$ is constant.

- $\text{OPT} = \max_{X \in \mathcal{F}} w(X)$ is the offline optimum
- $w(\mathcal{A}(\sigma))$ is (expected) weight of set outputted by $\mathcal{A}(\sigma)$.

- Polynomial running time is also desired, although not required.

For offline problem:

- There is trivial $O(2^m)$ time algorithm for computing OPT.
  - Simply check weight of every $X \in \mathcal{F} = \{X : X \subseteq E\}$.
  - Remember that $|E| = m$.
- Therefore, goal is to find poly-time algorithm for OPT.

For online problem:

- (Exponential time) algorithm choosing, for every ordering σ, an X with maximum weight is impossible.
- Even finding (exponential time) α-approximation is non-trivial.

Formal requirements for (online) algorithm $\mathcal{A}$:

Formal requirements for (online) algorithm $\mathcal{A}$:

Takes as input deterministic ordering $(x_1, \ldots, x_m)$ and weights $w_{x_1}, \ldots, w_{x_m}$.

Formal requirements for (online) algorithm $\mathcal{A}$:

Takes as input deterministic ordering $(x_1, \ldots, x_m)$ and weights $w_{x_1}, \ldots, w_{x_m}$.

- Specifies for every $i = 1, \ldots, m$ whether or not it selects x_i.

Formal requirements for (online) algorithm $\mathcal{A}$:

Takes as input deterministic ordering $(x_1, \ldots, x_m)$ and weights $w_{x_1}, \ldots, w_{x_m}$.

- Specifies for every $i = 1, \ldots, m$ whether or not it selects x_i.
- For given i, this YES-NO decision is a (randomized) function of:

Formal requirements for (online) algorithm $\mathcal{A}$:

Takes as input deterministic ordering $(x_1, \ldots, x_m)$ and weights $w_{x_1}, \ldots, w_{x_m}$.

- Specifies for every $i = 1, \ldots, m$ whether or not it selects x_i.
- For given i, this YES-NO decision is a (randomized) function of:
  - Total number of elements m.

Formal requirements for (online) algorithm $\mathcal{A}$:

Takes as input deterministic ordering $(x_1, \ldots, x_m)$ and weights $w_{x_1}, \ldots, w_{x_m}$.

- Specifies for every $i = 1, \ldots, m$ whether or not it selects x_i.
- For given i, this YES-NO decision is a (randomized) function of:
  - Total number of elements m.
  - Elements $x_1, \ldots, x_{i-1}$.

Formal requirements for (online) algorithm $\mathcal{A}$:

Takes as input deterministic ordering $(x_1, \ldots, x_m)$ and weights $w_{x_1}, \ldots, w_{x_m}$.

- Specifies for every $i = 1, \ldots, m$ whether or not it selects x_i.
- For given i, this YES-NO decision is a (randomized) function of:
  - Total number of elements m.
  - Elements $x_1, \ldots, x_{i-1}$.
  - Weights $w_{x_1}, \ldots, w_{x_{i-1}}$.

Formal requirements for (online) algorithm $\mathcal{A}$:

Takes as input deterministic ordering $(x_1, \ldots, x_m)$ and weights $w_{x_1}, \ldots, w_{x_m}$.

- Specifies for every $i = 1, \ldots, m$ whether or not it selects x_i.
- For given i, this YES-NO decision is a (randomized) function of:
  - Total number of elements m.
  - Elements $x_1, \ldots, x_{i-1}$.
  - Weights $w_{x_1}, \ldots, w_{x_{i-1}}$.
  - The order $(x_1, \ldots, x_i)$.

Formal requirements for (online) algorithm $\mathcal{A}$:

Takes as input deterministic ordering $(x_1, \ldots, x_m)$ and weights $w_{x_1}, \ldots, w_{x_m}$.

- Specifies for every $i = 1, \ldots, m$ whether or not it selects x_i.
- For given i, this YES-NO decision is a (randomized) function of:
  - Total number of elements m.
  - Elements $x_1, \ldots, x_{i-1}$.
  - Weights $w_{x_1}, \ldots, w_{x_{i-1}}$.
  - The order $(x_1, \ldots, x_i)$.
    - Last aspect is usually irrelevant.

Formal requirements for (online) algorithm $\mathcal{A}$:

Takes as input deterministic ordering $(x_1, \ldots, x_m)$ and weights $w_{x_1}, \ldots, w_{x_m}$.

- Specifies for every $i = 1, \ldots, m$ whether or not it selects x_i.
- For given i, this YES-NO decision is a (randomized) function of:
  - Total number of elements m.
  - Elements $x_1, \ldots, x_{i-1}$.
  - Weights $w_{x_1}, \ldots, w_{x_{i-1}}$.
  - The order $(x_1, \ldots, x_i)$.
    - Last aspect is usually irrelevant.

Worst-case ordering

Remark (Worst-case ordering)

If we would (again) drop the assumption of a uniform random arrival order,

Worst-case ordering

Remark (Worst-case ordering)

If we would (again) drop the assumption of a uniform random arrival order, the performance guarantee of $\mathcal{A}$ can be defined w.r.t. worst-case ordering.

Worst-case ordering

Remark (Worst-case ordering)

If we would (again) drop the assumption of a uniform random arrival order, the performance guarantee of $\mathcal{A}$ can be defined w.r.t. worst-case ordering. Then $\mathcal{A}$ is called α-approximation if

Worst-case ordering

Remark (Worst-case ordering)

If we would (again) drop the assumption of a uniform random arrival order, the performance guarantee of $\mathcal{A}$ can be defined w.r.t. worst-case ordering. Then $\mathcal{A}$ is called α-approximation if

$$\min_\sigma w(\mathcal{A}(\sigma)) \geq \alpha \cdot \mathrm{OPT},$$

Worst-case ordering

Remark (Worst-case ordering)

If we would (again) drop the assumption of a uniform random arrival order, the performance guarantee of $\mathcal{A}$ can be defined w.r.t. worst-case ordering. Then $\mathcal{A}$ is called α-approximation if

$$\min_\sigma w(\mathcal{A}(\sigma)) \geq \alpha \cdot \text{OPT},$$

again with $\text{OPT} = \max_{X \in \mathcal{F}} w(X)$ the offline optimum.

Worst-case ordering

Remark (Worst-case ordering)

If we would (again) drop the assumption of a uniform random arrival order, the performance guarantee of $\mathcal{A}$ can be defined w.r.t. worst-case ordering. Then $\mathcal{A}$ is called α-approximation if

$$\min_\sigma w(\mathcal{A}(\sigma)) \geq \alpha \cdot \mathrm{OPT},$$

again with $\mathrm{OPT} = \max_{X \in \mathcal{F}} w(X)$ the offline optimum.

- In worst-case arrival setting, no constant-factor algorithm exists.

Online selection problems

Uniform random arrivals: Secretary problem

Secretary problem (selecting best element)

- Elements (secretaries) $E = \{e_1, \ldots, e_n\}$ arrive one by one.

Secretary problem (selecting best element)

- Elements (secretaries) $E = \{e_1, \ldots, e_n\}$ arrive one by one.
  - Uniform random arrival order $\sigma = (\sigma(1), \ldots, \sigma(m))$

Secretary problem (selecting best element)

- Elements (secretaries) $E = \{e_1, \ldots, e_n\}$ arrive one by one.
  - Uniform random arrival order $\sigma = (\sigma(1), \ldots, \sigma(m))$
  - σ is permutation of elements $\{e_1, \ldots, e_n\}$.

Secretary problem (selecting best element)

- Elements (secretaries) $E = \{e_1, \dots, e_n\}$ arrive one by one.
  - Uniform random arrival order $\sigma = (\sigma(1), \dots, \sigma(m))$
  - σ is permutation of elements $\{e_1, \dots, e_n\}$.
- Weight $w_{\sigma(i)}$ revealed upon arrival of $\sigma(i)$.

Secretary problem (selecting best element)

- Elements (secretaries) $E = \{e_1, \dots, e_n\}$ arrive one by one.
  - Uniform random arrival order $\sigma = (\sigma(1), \dots, \sigma(m))$
  - σ is permutation of elements $\{e_1, \dots, e_n\}$.
- Weight $w_{\sigma(i)}$ revealed upon arrival of $\sigma(i)$.
- Irrevocably decide whether to select or reject $\sigma(i)$.

Secretary problem (selecting best element)

- Elements (secretaries) $E = \{e_1, \ldots, e_n\}$ arrive one by one.
  - Uniform random arrival order $\sigma = (\sigma(1), \ldots, \sigma(m))$
  - σ is permutation of elements $\{e_1, \ldots, e_n\}$.
- Weight $w_{\sigma(i)}$ revealed upon arrival of $\sigma(i)$.
- Irrevocably decide whether to select or reject $\sigma(i)$.

Goal: Select a secretary with maximum weight $w^* = \max_i w_i$.

Secretary problem (selecting best element)

- Elements (secretaries) $E = \{e_1, \dots, e_n\}$ arrive one by one.
  - Uniform random arrival order $\sigma = (\sigma(1), \dots, \sigma(m))$
  - σ is permutation of elements $\{e_1, \dots, e_n\}$.
- Weight $w_{\sigma(i)}$ revealed upon arrival of $\sigma(i)$.
- Irrevocably decide whether to select or reject $\sigma(i)$.

Goal: Select a secretary with maximum weight $w^* = \max_i w_i$.

- *Formally speaking, we have $\mathcal{F} = \{\{e_1\}, \{e_2\}, \dots, \{e_m\}\}$.*

Secretary problem (selecting best element)

- Elements (secretaries) $E = \{e_1, \ldots, e_n\}$ arrive one by one.
  - Uniform random arrival order $\sigma = (\sigma(1), \ldots, \sigma(m))$
  - σ is permutation of elements $\{e_1, \ldots, e_n\}$.
- Weight $w_{\sigma(i)}$ revealed upon arrival of $\sigma(i)$.
- Irrevocably decide whether to select or reject $\sigma(i)$.

Goal: Select a secretary with maximum weight $w^* = \max_i w_i$.

- *Formally speaking, we have $\mathcal{F} = \{\{e_1\}, \{e_2\}, \ldots, \{e_m\}\}$.*

Theorem (Lindley (1961) and Dynkin (1963))

There is a $(\frac{1}{e} - \frac{1}{m})$-approximation algorithm for the (weight maximization) secretary problem.

Secretary problem (selecting best element)

- Elements (secretaries) $E = \{e_1, \dots, e_n\}$ arrive one by one.
  - Uniform random arrival order $\sigma = (\sigma(1), \dots, \sigma(m))$
  - σ is permutation of elements $\{e_1, \dots, e_n\}$.
- Weight $w_{\sigma(i)}$ revealed upon arrival of $\sigma(i)$.
- Irrevocably decide whether to select or reject $\sigma(i)$.

Goal: Select a secretary with maximum weight $w^* = \max_i w_i$.

- *Formally speaking, we have $\mathcal{F} = \{\{e_1\}, \{e_2\}, \dots, \{e_m\}\}$.*

Theorem (Lindley (1961) and Dynkin (1963))

There is a $(\frac{1}{e} - \frac{1}{m})$-approximation algorithm for the (weight maximization) secretary problem.

- **Side note:** Original version of secretary problem asks for maximizing probability with which best element is selected.

Secretary problem (selecting best element)

- Elements (secretaries) $E = \{e_1, \ldots, e_n\}$ arrive one by one.
  - Uniform random arrival order $\sigma = (\sigma(1), \ldots, \sigma(m))$
  - σ is permutation of elements $\{e_1, \ldots, e_n\}$.
- Weight $w_{\sigma(i)}$ revealed upon arrival of $\sigma(i)$.
- Irrevocably decide whether to select or reject $\sigma(i)$.

Goal: Select a secretary with maximum weight $w^* = \max_i w_i$.

- *Formally speaking, we have $\mathcal{F} = \{\{e_1\}, \{e_2\}, \ldots, \{e_m\}\}$.*

Theorem (Lindley (1961) and Dynkin (1963))

There is a $(\frac{1}{e} - \frac{1}{m})$-approximation algorithm for the (weight maximization) secretary problem.

- **Side note:** Original version of secretary problem asks for maximizing probability with which best element is selected.
  - If one picks maximum weight element with prob. $\geq \frac{1}{e}$,

Secretary problem (selecting best element)

- Elements (secretaries) $E = \{e_1, \ldots, e_n\}$ arrive one by one.
  - Uniform random arrival order $\sigma = (\sigma(1), \ldots, \sigma(m))$
  - σ is permutation of elements $\{e_1, \ldots, e_n\}$.
- Weight $w_{\sigma(i)}$ revealed upon arrival of $\sigma(i)$.
- Irrevocably decide whether to select or reject $\sigma(i)$.

Goal: Select a secretary with maximum weight $w^* = \max_i w_i$.

- *Formally speaking, we have $\mathcal{F} = \{\{e_1\}, \{e_2\}, \ldots, \{e_m\}\}$.*

Theorem (Lindley (1961) and Dynkin (1963))

There is a $(\frac{1}{e} - \frac{1}{m})$-approximation algorithm for the (weight maximization) secretary problem.

- **Side note:** Original version of secretary problem asks for maximizing probability with which best element is selected.
  - If one picks maximum weight element with prob. $\geq \frac{1}{e}$, then expected weight of chosen element is $\geq \frac{1}{e} w^*$.

Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

Phase I (Observation):

Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.

Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.

Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

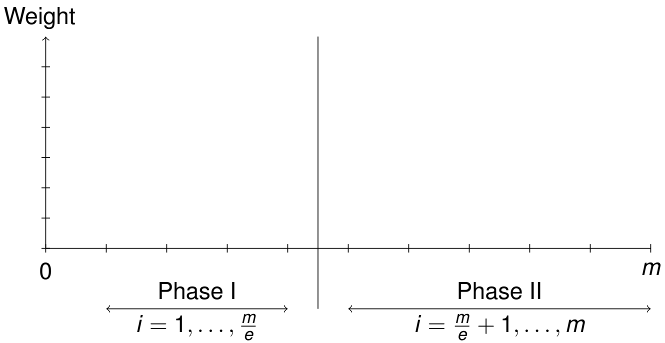
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

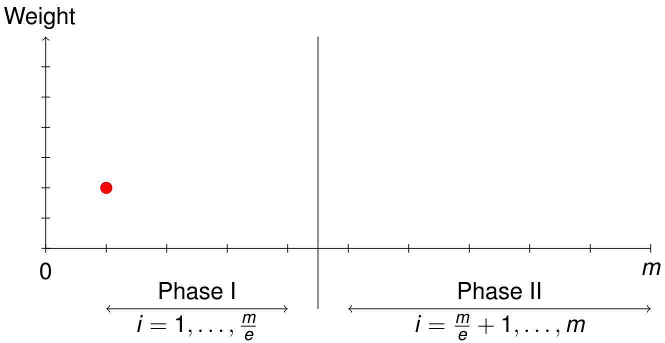
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

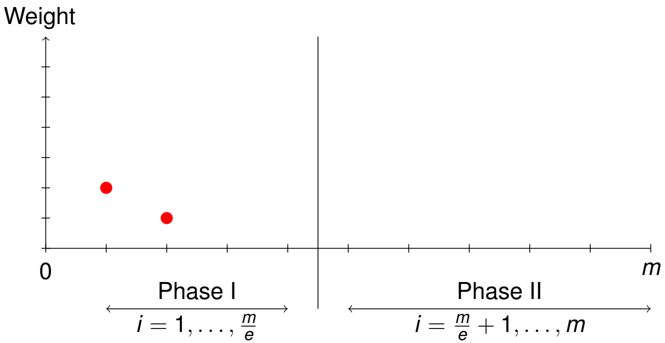
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

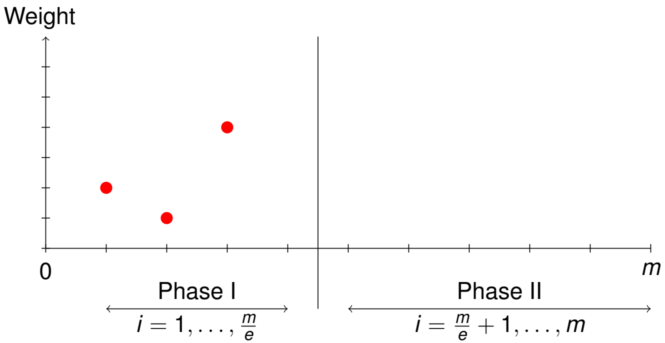
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

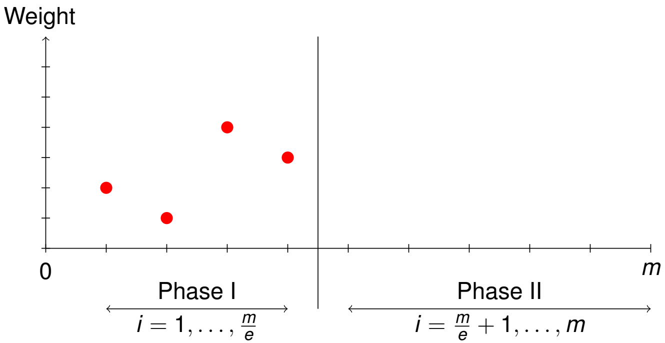
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

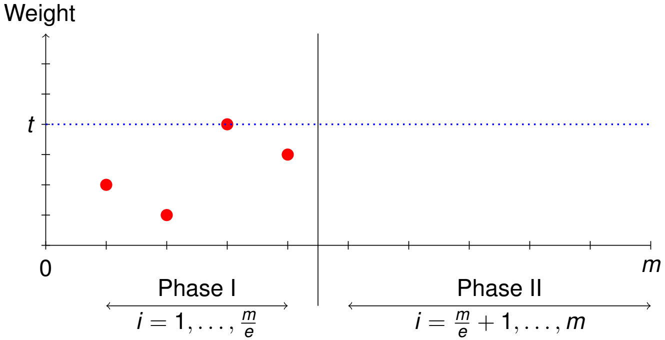
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

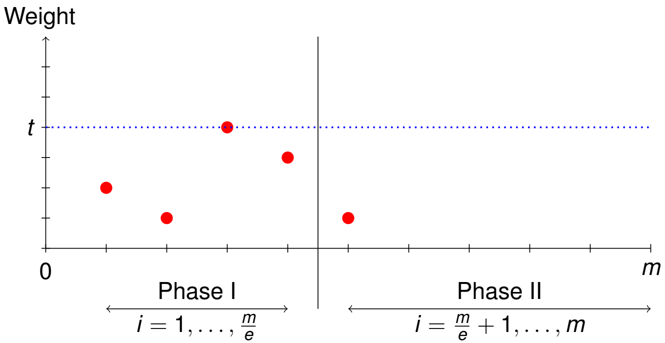
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

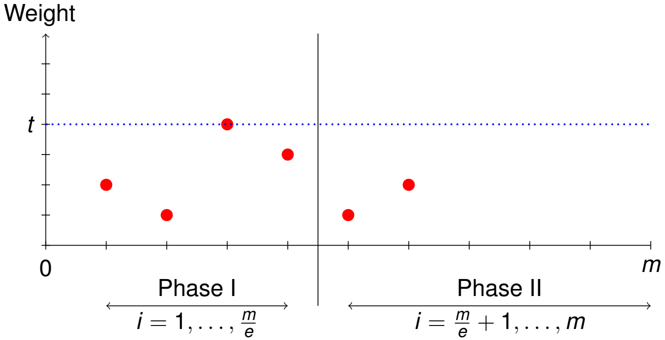
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

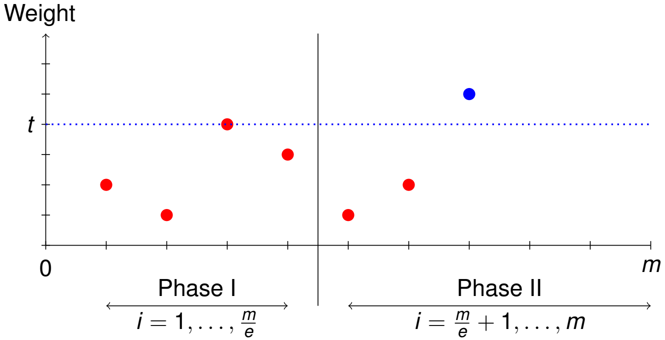
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

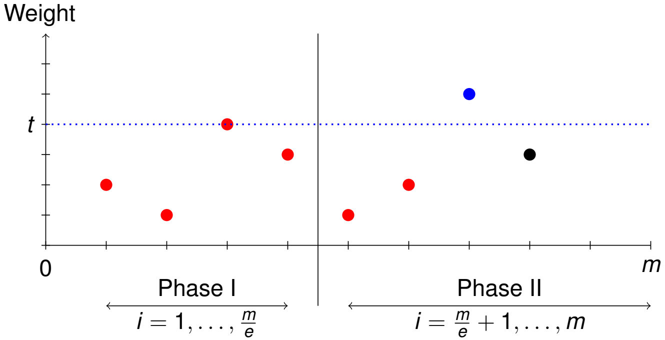
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

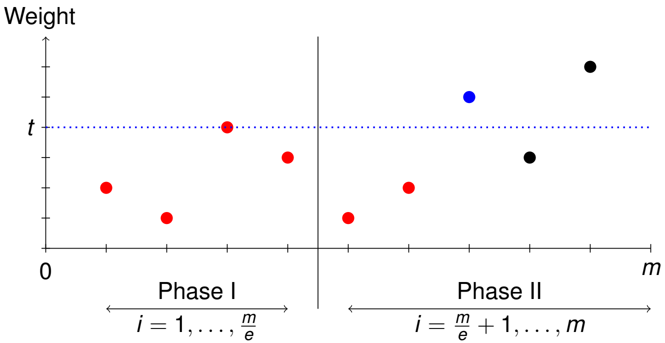
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Secretary algorithm for ordering $(\sigma(1), \ldots, \sigma(m))$

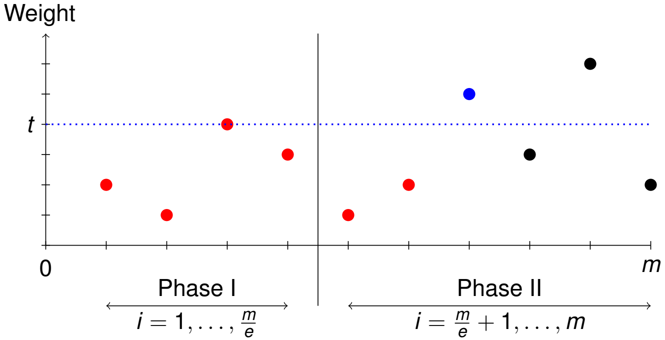
Phase I (Observation):

- For $i = 1, \ldots, \lfloor \frac{m}{e} \rfloor$: Do not select $\sigma(i)$.

Phase II (Selection):

- Set threshold $t = \max_{j=1,\ldots,\lfloor \frac{m}{e} \rfloor} w_{\sigma(j)}$.
- For $i = \lfloor \frac{m}{e} \rfloor + 1, \ldots, n$: If $w_{\sigma(i)} \geq t$, select $\sigma(i)$ and STOP.

If in future statements we write $\frac{m}{e}$, we mean $\lfloor \frac{m}{e} \rfloor$.



Analysis (sketch)

Consider $w^* = \max_i w_i$ (assume w.l.o.g. that weights are distinct).

Analysis (sketch)

Consider $w^* = \max_i w_i$ (assume w.l.o.g. that weights are distinct).

- For readability, we assume (wrongfully) that $r = m/e$ is integer.

Analysis (sketch)

Consider $w^* = \max_i w_i$ (assume w.l.o.g. that weights are distinct).

- For readability, we assume (wrongfully) that $r = m/e$ is integer.
  - Simple adjustments to analysis suffice to deal with this issue.

Analysis (sketch)

Consider $w^* = \max_i w_i$ (assume w.l.o.g. that weights are distinct).

- For readability, we assume (wrongfully) that $r = m/e$ is integer.
  - Simple adjustments to analysis suffice to deal with this issue.

Claim: Secretary algorithm is $\approx \frac{1}{e}$-approximation for w^*.

Analysis (sketch)

Consider $w^* = \max_i w_i$ (assume w.l.o.g. that weights are distinct).

- For readability, we assume (wrongfully) that $r = m/e$ is integer.
  - Simple adjustments to analysis suffice to deal with this issue.

Claim: Secretary algorithm is $\approx \frac{1}{e}$-approximation for w^*.

- We show that with prob. $\frac{1}{e}$, element with weight w^* is selected.

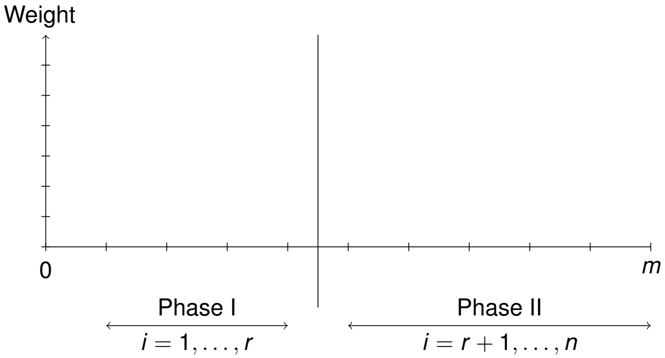
Analysis (sketch)

Consider $w^* = \max_i w_i$ (assume w.l.o.g. that weights are distinct).

- For readability, we assume (wrongfully) that $r = m/e$ is integer.
  - Simple adjustments to analysis suffice to deal with this issue.

Claim: Secretary algorithm is $\approx \frac{1}{e}$-approximation for w^*.

- We show that with prob. $\frac{1}{e}$, element with weight w^* is selected.



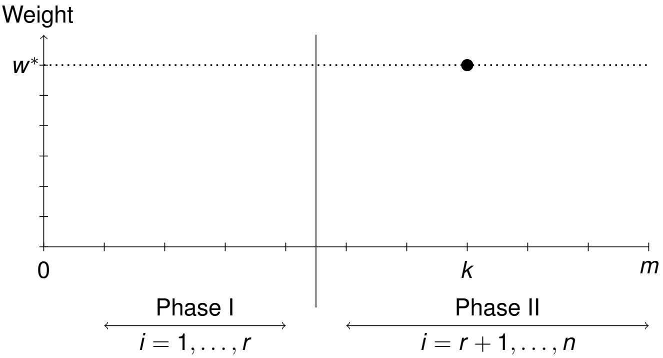
Analysis (sketch)

Consider $w^* = \max_i w_i$ (assume w.l.o.g. that weights are distinct).

- For readability, we assume (wrongfully) that $r = m/e$ is integer.
  - Simple adjustments to analysis suffice to deal with this issue.

Claim: Secretary algorithm is $\approx \frac{1}{e}$-approximation for w^*.

- We show that with prob. $\frac{1}{e}$, element with weight w^* is selected.



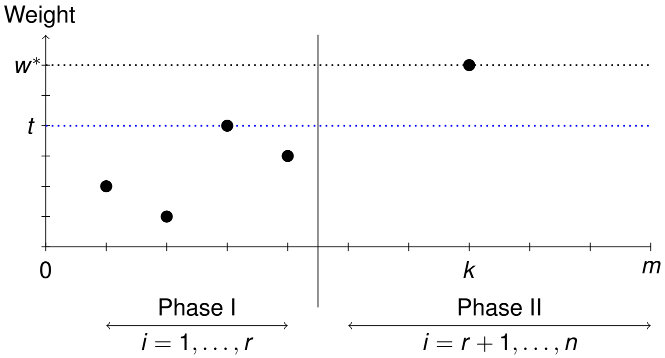
Analysis (sketch)

Consider $w^* = \max_i w_i$ (assume w.l.o.g. that weights are distinct).

- For readability, we assume (wrongfully) that $r = m/e$ is integer.
    - Simple adjustments to analysis suffice to deal with this issue.

Claim: Secretary algorithm is $\approx \frac{1}{e}$-approximation for w^*.

- We show that with prob. $\frac{1}{e}$, element with weight w^* is selected.



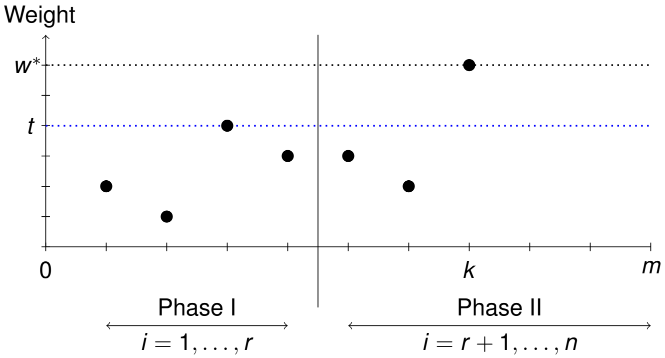
Analysis (sketch)

Consider $w^* = \max_i w_i$ (assume w.l.o.g. that weights are distinct).

- For readability, we assume (wrongfully) that $r = m/e$ is integer.
  - Simple adjustments to analysis suffice to deal with this issue.

Claim: Secretary algorithm is $\approx \frac{1}{e}$-approximation for w^*.

- We show that with prob. $\frac{1}{e}$, element with weight w^* is selected.



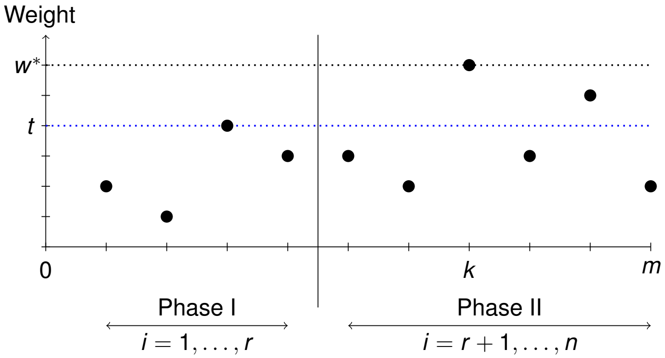
Analysis (sketch)

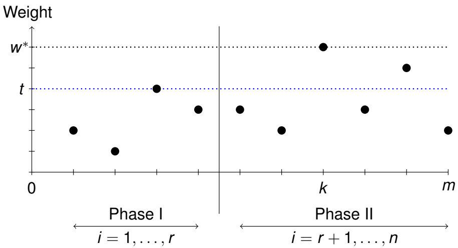
Consider $w^* = \max_i w_i$ (assume w.l.o.g. that weights are distinct).

- For readability, we assume (wrongfully) that $r = m/e$ is integer.
  - Simple adjustments to analysis suffice to deal with this issue.

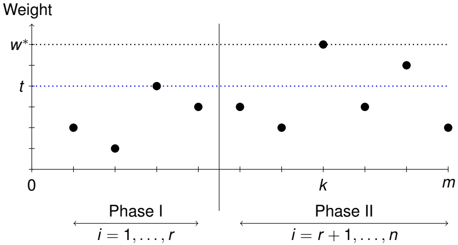
Claim: Secretary algorithm is $\approx \frac{1}{e}$-approximation for w^*.

- We show that with prob. $\frac{1}{e}$, element with weight w^* is selected.



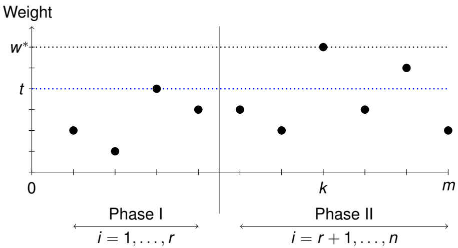


For fixed position $k \geq r + 1$, we select weight w^* at k if



For fixed position $k \geq r + 1$, we select weight w^* at k if

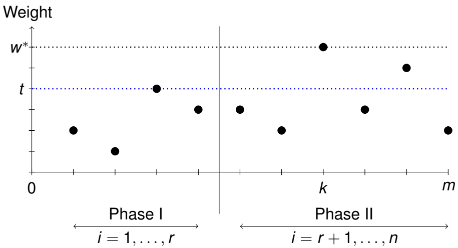
- Best secretary in $\{1, \dots, r\}$ is same as best in $\{1, \dots, k - 1\}$. (Z)



For fixed position $k \geq r + 1$, we select weight w^* at k if

- Best secretary in $\{1, \ldots, r\}$ is same as best in $\{1, \ldots, k - 1\}$. (Z)

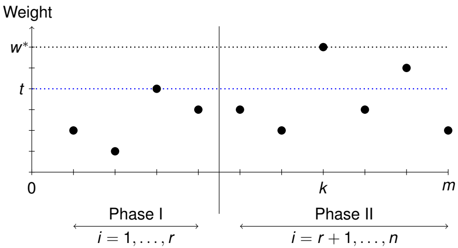
For given $k \geq r + 1$, using uniform random order assumption,



For fixed position $k \geq r + 1$, we select weight w^* at k if

- Best secretary in $\{1, \ldots, r\}$ is same as best in $\{1, \ldots, k-1\}$. (Z)

For given $k \geq r + 1$, using uniform random order assumption, this happens with probability

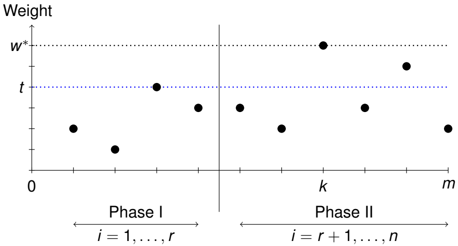


For fixed position $k \geq r+1$, we select weight w^* at k if

- Best secretary in $\{1, \ldots, r\}$ is same as best in $\{1, \ldots, k-1\}$. (Z)

For given $k \geq r+1$, using uniform random order assumption, this happens with probability

$\mathbb{P}(w^* \text{ appears at } k \text{ and } (Z) \text{ holds})$

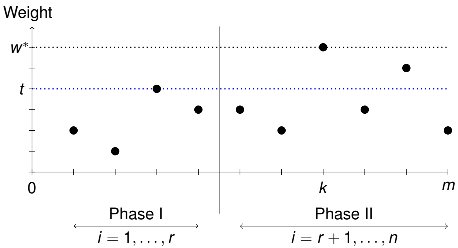


For fixed position $k \geq r + 1$, we select weight w^* at k if

- Best secretary in $\{1, \ldots, r\}$ is same as best in $\{1, \ldots, k-1\}$. (Z)

For given $k \geq r + 1$, using uniform random order assumption, this happens with probability

$$\mathbb{P}(w^* \text{ appears at } k \text{ and } (Z) \text{ holds}) = \mathbb{P}(w^* \text{ appears at } k)\mathbb{P}((Z) \text{ holds})$$

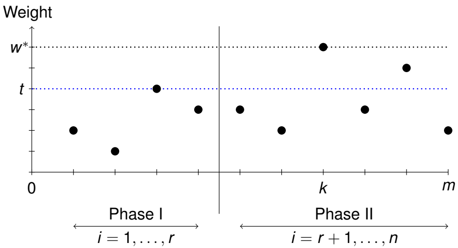


For fixed position $k \geq r + 1$, we select weight w^* at k if

- Best secretary in $\{1, \ldots, r\}$ is same as best in $\{1, \ldots, k-1\}$. (Z)

For given $k \geq r + 1$, using uniform random order assumption, this happens with probability

$$\begin{aligned} \mathbb{P}(w^* \text{ appears at } k \text{ and } (Z) \text{ holds}) &= \mathbb{P}(w^* \text{ appears at } k)\mathbb{P}((Z) \text{ holds}) \\ &= \frac{1}{m}\frac{r}{k-1}. \end{aligned}$$



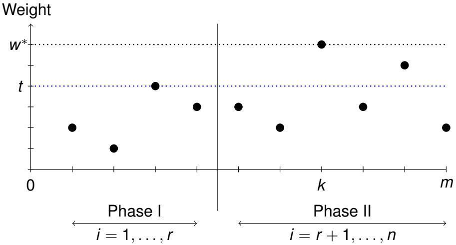
For fixed position $k \geq r + 1$, we select weight w^* at k if

- Best secretary in $\{1, \dots, r\}$ is same as best in $\{1, \dots, k-1\}$. (Z)

For given $k \geq r + 1$, using uniform random order assumption, this happens with probability

$$\begin{aligned}\mathbb{P}(w^* \text{ appears at } k \text{ and } (Z) \text{ holds}) &= \mathbb{P}(w^* \text{ appears at } k)\mathbb{P}((Z) \text{ holds}) \\ &= \frac{1}{m}\frac{r}{k-1}.\end{aligned}$$

Exercise: Convince yourself that these events are indeed independent!



For fixed position $k \geq r+1$, we select weight w^* at k if

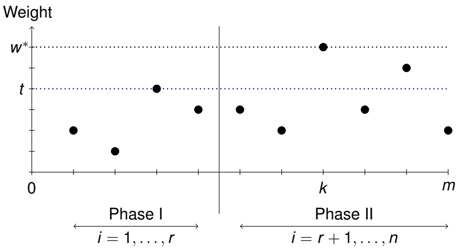
- Best secretary in $\{1, \dots, r\}$ is same as best in $\{1, \dots, k-1\}$. (Z)

For given $k \geq r+1$, using uniform random order assumption, this happens with probability

$$\begin{aligned} \mathbb{P}(w^* \text{ appears at } k \text{ and } (Z) \text{ holds}) &= \mathbb{P}(w^* \text{ appears at } k)\mathbb{P}((Z) \text{ holds}) \\ &= \frac{1}{m}\frac{r}{k-1}. \end{aligned}$$

Exercise: Convince yourself that these events are indeed independent!

It follows that



For fixed position $k \geq r+1$, we select weight w^* at k if

- Best secretary in $\{1, \ldots, r\}$ is same as best in $\{1, \ldots, k-1\}$. (Z)

For given $k \geq r+1$, using uniform random order assumption, this happens with probability

$$\begin{aligned} \mathbb{P}(w^* \text{ appears at } k \text{ and } (Z) \text{ holds}) &= \mathbb{P}(w^* \text{ appears at } k)\mathbb{P}((Z) \text{ holds}) \\ &= \frac{1}{m}\frac{r}{k-1}. \end{aligned}$$

Exercise: Convince yourself that these events are indeed independent!

It follows that

$$\mathbb{P}(w^* \text{ is selected}) = \sum_{k=r+1}^{m} \frac{r}{k-1}\frac{1}{m} = \frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k}.$$

$$\mathbb{P}(w^* \text{ is selected}) = \sum_{k=r+1}^{m} \frac{r}{k-1}\frac{1}{m} = \frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k}.$$

At this point, it becomes a matter of calculus.

$$\mathbb{P}(w^* \text{ is selected}) = \sum_{k=r+1}^{m} \frac{r}{k-1}\frac{1}{m} = \frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k}.$$

At this point, it becomes a matter of calculus. Roughly speaking, for m large,

$$\mathbb{P}(w^* \text{ is selected}) = \sum_{k=r+1}^{m} \frac{r}{k-1}\frac{1}{m} = \frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k}.$$

At this point, it becomes a matter of calculus. Roughly speaking, for m large,

$$\frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k} \approx -\frac{r}{m}\ln\left(\frac{r}{m}\right) \approx \frac{1}{e} \quad \text{for } r = \frac{m}{e}.$$

$$\mathbb{P}(w^* \text{ is selected}) = \sum_{k=r+1}^{m} \frac{r}{k-1}\frac{1}{m} = \frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k}.$$

At this point, it becomes a matter of calculus. Roughly speaking, for m large,

$$\frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k} \approx -\frac{r}{m}\ln\left(\frac{r}{m}\right) \approx \frac{1}{e} \quad \text{for } r = \frac{m}{e}.$$

This completes the analysis. □

$$\mathbb{P}(w^* \text{ is selected}) = \sum_{k=r+1}^{m} \frac{r}{k-1}\frac{1}{m} = \frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k}.$$

At this point, it becomes a matter of calculus. Roughly speaking, for m large,

$$\frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k} \approx -\frac{r}{m}\ln\left(\frac{r}{m}\right) \approx \frac{1}{e} \quad \text{for } r = \frac{m}{e}.$$

This completes the analysis. □

Theorem (Lindley (1961) and Dynkin (1963))

There is a $(\frac{1}{e} - \frac{1}{m})$-approximation algorithm for the (weight maximization) secretary problem.

$$\mathbb{P}(w^* \text{ is selected}) = \sum_{k=r+1}^{m} \frac{r}{k-1}\frac{1}{m} = \frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k}.$$

At this point, it becomes a matter of calculus. Roughly speaking, for m large,

$$\frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k} \approx -\frac{r}{m}\ln\left(\frac{r}{m}\right) \approx \frac{1}{e} \quad \text{for } r = \frac{m}{e}.$$

This completes the analysis. □

Theorem (Lindley (1961) and Dynkin (1963))

There is a $(\frac{1}{e} - \frac{1}{m})$-approximation algorithm for the (weight maximization) secretary problem.

- Factor $1/e$ is also best possible.

$$\mathbb{P}(w^* \text{ is selected}) = \sum_{k=r+1}^{m} \frac{r}{k-1}\frac{1}{m} = \frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k}.$$

At this point, it becomes a matter of calculus. Roughly speaking, for m large,

$$\frac{r}{m}\sum_{k=r}^{m-1}\frac{1}{k} \approx -\frac{r}{m}\ln\left(\frac{r}{m}\right) \approx \frac{1}{e} \quad \text{for } r = \frac{m}{e}.$$

This completes the analysis. □

Theorem (Lindley (1961) and Dynkin (1963))

There is a $(\frac{1}{e} - \frac{1}{m})$-approximation algorithm for the (weight maximization) secretary problem.

- Factor $1/e$ is also best possible.
- Secretary algorithm is polynomial time algorithm.

Online selection problems

Prophet Inequalities

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

In Bayesian setting, we have for every element i a probability distribution $X_i : \mathbb{R}_{\geq 0} \rightarrow [0, 1]$.

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

In Bayesian setting, we have for every element i a probability distribution $X_i : \mathbb{R}_{\geq 0} \rightarrow [0, 1]$.

- You may also think of X_i as discrete distribution $X_i : \mathbb{N} \rightarrow [0, 1]$.

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

In Bayesian setting, we have for every element i a probability distribution $X_i : \mathbb{R}_{\geq 0} \rightarrow [0, 1]$.

- You may also think of X_i as discrete distribution $X_i : \mathbb{N} \rightarrow [0, 1]$.
- Weight w_i of element e_i is sample from X_i.

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

In Bayesian setting, we have for every element i a probability distribution $X_i : \mathbb{R}_{\geq 0} \to [0, 1]$.

- You may also think of X_i as discrete distribution $X_i : \mathbb{N} \to [0, 1]$.
- Weight w_i of element e_i is sample from X_i.
    - That is, we have $w_j \sim X_j$

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

In Bayesian setting, we have for every element i a probability distribution $X_i : \mathbb{R}_{\geq 0} \to [0, 1]$.

- You may also think of X_i as discrete distribution $X_i : \mathbb{N} \to [0, 1]$.
- Weight w_i of element e_i is sample from X_i.
  - That is, we have $w_j \sim X_j$

Online selection problem

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

In Bayesian setting, we have for every element i a probability distribution $X_i : \mathbb{R}_{\geq 0} \rightarrow [0, 1]$.

- You may also think of X_i as discrete distribution $X_i : \mathbb{N} \rightarrow [0, 1]$.
- Weight w_i of element e_i is sample from X_i.
    - That is, we have $w_j \sim X_j$

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \ldots, \sigma(m))$.

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

In Bayesian setting, we have for every element i a probability distribution $X_i : \mathbb{R}_{\geq 0} \to [0, 1]$.

- You may also think of X_i as discrete distribution $X_i : \mathbb{N} \to [0, 1]$.
- Weight w_i of element e_i is sample from X_i.
  - That is, we have $w_j \sim X_j$

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \ldots, \sigma(m))$.

- *Permutation of elements e.g., (e_4, e_2, e_1, e_3).*

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

In Bayesian setting, we have for every element i a probability distribution $X_i : \mathbb{R}_{\geq 0} \rightarrow [0, 1]$.

- You may also think of X_i as discrete distribution $X_i : \mathbb{N} \rightarrow [0, 1]$.
- Weight w_i of element e_i is sample from X_i.
  - That is, we have $w_j \sim X_j$

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \dots, \sigma(m))$.

- *Permutation of elements e.g.,* (e_4, e_2, e_1, e_3).

Upon arrival of element $\sigma(i)$:

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

In Bayesian setting, we have for every element i a probability distribution $X_i : \mathbb{R}_{\geq 0} \to [0, 1]$.

- You may also think of X_i as discrete distribution $X_i : \mathbb{N} \to [0, 1]$.
- Weight w_i of element e_i is sample from X_i.
  - That is, we have $w_j \sim X_j$

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \dots, \sigma(m))$.

- *Permutation of elements e.g.,* (e_4, e_2, e_1, e_3).

Upon arrival of element $\sigma(i)$:

- Weight $w_{\sigma(i)}$ is revealed.

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

In Bayesian setting, we have for every element i a probability distribution $X_i : \mathbb{R}_{\geq 0} \to [0, 1]$.

- You may also think of X_i as discrete distribution $X_i : \mathbb{N} \to [0, 1]$.
- Weight w_i of element e_i is sample from X_i.
  - That is, we have $w_j \sim X_j$

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \dots, \sigma(m))$.

- *Permutation of elements e.g.,* (e_4, e_2, e_1, e_3).

Upon arrival of element $\sigma(i)$:

- Weight $w_{\sigma(i)}$ is revealed.
- Decide (irrevocably) whether to select or reject $\sigma(i)$.

Bayesian setting (for selecting one element)

Instead of making assumption on arrival order (uniform random), we make assumption on the (unknown) weights of the elements.

In Bayesian setting, we have for every element i a probability distribution $X_i : \mathbb{R}_{\geq 0} \rightarrow [0, 1]$.

- You may also think of X_i as discrete distribution $X_i : \mathbb{N} \rightarrow [0, 1]$.
- Weight w_i of element e_i is sample from X_i.
  - That is, we have $w_j \sim X_j$

Online selection problem

Elements arrive one by one in unknown order $\sigma = (\sigma(1), \dots, \sigma(m))$.

- *Permutation of elements e.g.,* (e_4, e_2, e_1, e_3).

Upon arrival of element $\sigma(i)$:

- Weight $w_{\sigma(i)}$ is revealed.
- Decide (irrevocably) whether to select or reject $\sigma(i)$.

Goal: *Select element with weight* $w^* = \max_{e \in E} w(e)$.

Formal procedure defining the problem:

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.
  - Adversary has seen all realizations.

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.
  - Adversary has seen all realizations.
- In step i, algorithm decides whether to select/reject $\sigma(i)$.

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.
    - Adversary has seen all realizations.
- In step i, algorithm decides whether to select/reject $\sigma(i)$.

Algorithm has access to same information as on Slide 17,

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.
    - Adversary has seen all realizations.
- In step i, algorithm decides whether to select/reject $\sigma(i)$.

Algorithm has access to same information as on Slide 17, and, in addition, it knows the distributions X_i.

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.
  - Adversary has seen all realizations.
- In step i, algorithm decides whether to select/reject $\sigma(i)$.

Algorithm has access to same information as on Slide 17, and, in addition, it knows the distributions X_i.

(But not realizations of not yet arrived elements.)

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.
  - Adversary has seen all realizations.
- In step i, algorithm decides whether to select/reject $\sigma(i)$.

Algorithm has access to same information as on Slide 17, and, in addition, it knows the distributions X_i.

(But not realizations of not yet arrived elements.)

About the adversary

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.
  - Adversary has seen all realizations.
- In step i, algorithm decides whether to select/reject $\sigma(i)$.

Algorithm has access to same information as on Slide 17, and, in addition, it knows the distributions X_i.

(But not realizations of not yet arrived elements.)

About the adversary

In general, we assume to have an all-knowing, adaptive adversary

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.
  - Adversary has seen all realizations.
- In step i, algorithm decides whether to select/reject $\sigma(i)$.

Algorithm has access to same information as on Slide 17, and, in addition, it knows the distributions X_i.

(But not realizations of not yet arrived elements.)

About the adversary

In general, we assume to have an all-knowing, adaptive adversary

- Can choose which element to present in step i, based on

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.
  - Adversary has seen all realizations.
- In step i, algorithm decides whether to select/reject $\sigma(i)$.

Algorithm has access to same information as on Slide 17, and, in addition, it knows the distributions X_i.

(But not realizations of not yet arrived elements.)

About the adversary

In general, we assume to have an all-knowing, adaptive adversary

- Can choose which element to present in step i, based on
  - Choices of online algorithm in steps $1, \ldots, i-1$.

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.
  - Adversary has seen all realizations.
- In step i, algorithm decides whether to select/reject $\sigma(i)$.

Algorithm has access to same information as on Slide 17, and, in addition, it knows the distributions X_i.

(But not realizations of not yet arrived elements.)

About the adversary

In general, we assume to have an all-knowing, adaptive adversary

- Can choose which element to present in step i, based on
  - Choices of online algorithm in steps $1, \ldots, i-1$.
  - Realizations of all elements (including those that have not

Formal procedure defining the problem:

- For every i, a realization $w_i \sim X_i$ is generated.
- Elements are presented by adversary in unknown order σ.
  - Adversary has seen all realizations.
- In step i, algorithm decides whether to select/reject $\sigma(i)$.

Algorithm has access to same information as on Slide 17, and, in addition, it knows the distributions X_i.

(But not realizations of not yet arrived elements.)

About the adversary

In general, we assume to have an all-knowing, adaptive adversary

- Can choose which element to present in step i, based on
  - Choices of online algorithm in steps $1, \ldots, i-1$.
  - Realizations of all elements (including those that have not arrived).

Adversary is non-adaptive if order is fixed after seeing all realizations.

Example

$E = \{e_1, e_2\}$ with following distributions.

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$.

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \tag{1}$$

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \tag{1}$$

$$w_2 \sim X_2 = \left\{ 1 + \delta \quad \text{with probability } 1. \right. \tag{2}$$

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \quad (1)$$

$$w_2 \sim X_2 = \left\{ 1 + \delta \quad \text{with probability } 1. \right. \quad (2)$$

Note that $\mathbb{E}[X_1] = \frac{1}{\epsilon} \times \epsilon + 0 \times (1 - \epsilon) = 1$ and $\mathbb{E}[X_2] = 1 + \delta$.

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \quad (1)$$

$$w_2 \sim X_2 = \{ 1 + \delta \quad \text{with probability } 1. \quad (2)$$

Note that $\mathbb{E}[X_1] = \frac{1}{\epsilon} \times \epsilon + 0 \times (1 - \epsilon) = 1$ and $\mathbb{E}[X_2] = 1 + \delta$.

- If arrival order would be (e_1, e_2), simply observe realization w_1.

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \quad (1)$$

$$w_2 \sim X_2 = \{\ 1 + \delta \quad \text{with probability } 1. \quad (2)$$

Note that $\mathbb{E}[X_1] = \frac{1}{\epsilon} \times \epsilon + 0 \times (1 - \epsilon) = 1$ and $\mathbb{E}[X_2] = 1 + \delta$.

- If arrival order would be (e_1, e_2), simply observe realization w_1.
  - If $w_1 = 1/\epsilon$, then select e_1 (as $\frac{1}{\epsilon} > 1 + \delta$).

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \tag{1}$$

$$w_2 \sim X_2 = \{\, 1 + \delta \quad \text{with probability } 1. \tag{2}$$

Note that $\mathbb{E}[X_1] = \frac{1}{\epsilon} \times \epsilon + 0 \times (1 - \epsilon) = 1$ and $\mathbb{E}[X_2] = 1 + \delta$.

- If arrival order would be (e_1, e_2), simply observe realization w_1.
  - If $w_1 = 1/\epsilon$, then select e_1 (as $\frac{1}{\epsilon} > 1 + \delta$).
  - If $w_1 = 0$, reject e_1 and select e_2.

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \tag{1}$$

$$w_2 \sim X_2 = \left\{ \, 1 + \delta \quad \text{with probability } 1. \right. \tag{2}$$

Note that $\mathbb{E}[X_1] = \frac{1}{\epsilon} \times \epsilon + 0 \times (1 - \epsilon) = 1$ and $\mathbb{E}[X_2] = 1 + \delta$.

- If arrival order would be (e_1, e_2), simply observe realization w_1.
  - If $w_1 = 1/\epsilon$, then select e_1 (as $\frac{1}{\epsilon} > 1 + \delta$).
  - If $w_1 = 0$, reject e_1 and select e_2.
- Worst-case arrival order is (e_2, e_1).

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \tag{1}$$

$$w_2 \sim X_2 = \left\{ 1 + \delta \quad \text{with probability } 1. \right. \tag{2}$$

Note that $\mathbb{E}[X_1] = \frac{1}{\epsilon} \times \epsilon + 0 \times (1 - \epsilon) = 1$ and $\mathbb{E}[X_2] = 1 + \delta$.

- If arrival order would be (e_1, e_2), simply observe realization w_1.
  - If $w_1 = 1/\epsilon$, then select e_1 (as $\frac{1}{\epsilon} > 1 + \delta$).
  - If $w_1 = 0$, reject e_1 and select e_2.
- Worst-case arrival order is (e_2, e_1).
  - We don't know realization w_1, when deciding on element e_2.

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \tag{1}$$

$$w_2 \sim X_2 = \left\{ 1 + \delta \quad \text{with probability } 1. \right. \tag{2}$$

Note that $\mathbb{E}[X_1] = \frac{1}{\epsilon} \times \epsilon + 0 \times (1 - \epsilon) = 1$ and $\mathbb{E}[X_2] = 1 + \delta$.

- If arrival order would be (e_1, e_2), simply observe realization w_1.
  - If $w_1 = 1/\epsilon$, then select e_1 (as $\frac{1}{\epsilon} > 1 + \delta$).
  - If $w_1 = 0$, reject e_1 and select e_2.
- Worst-case arrival order is (e_2, e_1).
  - We don't know realization w_1, when deciding on element e_2.
  - Nevertheless, it is (intuitively) optimal to select e_2.

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \tag{1}$$

$$w_2 \sim X_2 = \{ 1 + \delta \quad \text{with probability } 1. \tag{2}$$

Note that $\mathbb{E}[X_1] = \frac{1}{\epsilon} \times \epsilon + 0 \times (1 - \epsilon) = 1$ and $\mathbb{E}[X_2] = 1 + \delta$.

- If arrival order would be (e_1, e_2), simply observe realization w_1.
  - If $w_1 = 1/\epsilon$, then select e_1 (as $\frac{1}{\epsilon} > 1 + \delta$).
  - If $w_1 = 0$, reject e_1 and select e_2.
- Worst-case arrival order is (e_2, e_1).
  - We don't know realization w_1, when deciding on element e_2.
  - Nevertheless, it is (intuitively) optimal to select e_2.
  - Why?

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \tag{1}$$

$$w_2 \sim X_2 = \left\{ 1 + \delta \quad \text{with probability } 1. \right. \tag{2}$$

Note that $\mathbb{E}[X_1] = \frac{1}{\epsilon} \times \epsilon + 0 \times (1 - \epsilon) = 1$ and $\mathbb{E}[X_2] = 1 + \delta$.

- If arrival order would be (e_1, e_2), simply observe realization w_1.
  - If $w_1 = 1/\epsilon$, then select e_1 (as $\frac{1}{\epsilon} > 1 + \delta$).
  - If $w_1 = 0$, reject e_1 and select e_2.
- Worst-case arrival order is (e_2, e_1).
  - We don't know realization w_1, when deciding on element e_2.
  - Nevertheless, it is (intuitively) optimal to select e_2.
  - Why? Deterministic value $w_2 = 1 + \delta > \mathbb{E}[X_1]$.

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \quad (1)$$

$$w_2 \sim X_2 = \{ 1 + \delta \quad \text{with probability } 1. \quad (2)$$

Note that $\mathbb{E}[X_1] = \frac{1}{\epsilon} \times \epsilon + 0 \times (1 - \epsilon) = 1$ and $\mathbb{E}[X_2] = 1 + \delta$.

- If arrival order would be (e_1, e_2), simply observe realization w_1.
  - If $w_1 = 1/\epsilon$, then select e_1 (as $\frac{1}{\epsilon} > 1 + \delta$).
  - If $w_1 = 0$, reject e_1 and select e_2.
- Worst-case arrival order is (e_2, e_1).
  - We don't know realization w_1, when deciding on element e_2.
  - Nevertheless, it is (intuitively) optimal to select e_2.
  - Why? Deterministic value $w_2 = 1 + \delta > \mathbb{E}[X_1]$.
    - In expectation (of X_1), we cannot do better if we reject e_2.

Example

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \quad (1)$$

$$w_2 \sim X_2 = \{ \; 1 + \delta \quad \text{with probability } 1. \quad (2)$$

Note that $\mathbb{E}[X_1] = \frac{1}{\epsilon} \times \epsilon + 0 \times (1 - \epsilon) = 1$ and $\mathbb{E}[X_2] = 1 + \delta$.

- If arrival order would be (e_1, e_2), simply observe realization w_1.
  - If $w_1 = 1/\epsilon$, then select e_1 (as $\frac{1}{\epsilon} > 1 + \delta$).
  - If $w_1 = 0$, reject e_1 and select e_2.
- Worst-case arrival order is (e_2, e_1).
  - We don't know realization w_1, when deciding on element e_2.
  - Nevertheless, it is (intuitively) optimal to select e_2.
  - Why? Deterministic value $w_2 = 1 + \delta > \mathbb{E}[X_1]$.
    - In expectation (of X_1), we cannot do better if we reject e_2.

- Performance objective is formalized next.

Performance of online algorithm

Performance is measured against that of the prophet.

Performance of online algorithm

Performance is measured against that of the prophet.

- Prophet gets to see all realizations $w_i \sim X_i$ after they are sampled.

Performance of online algorithm

Performance is measured against that of the prophet.

- Prophet gets to see all realizations $w_i \sim X_i$ after they are sampled.
- Independent of ordering, she simply selects $w^* = \max_i w_i$.

Performance of online algorithm

Performance is measured against that of the prophet.

- Prophet gets to see all realizations $w_i \sim X_i$ after they are sampled.
- Independent of ordering, she simply selects $w^* = \max_i w_i$.

Expected weight of chosen element for prophet is

Performance of online algorithm

Performance is measured against that of the prophet.

- Prophet gets to see all realizations $w_i \sim X_i$ after they are sampled.
- Independent of ordering, she simply selects $w^* = \max_i w_i$.

Expected weight of chosen element for prophet is

$$\mathrm{OPT} = \mathbb{E}_{(y_1,\ldots,y_m)\sim X_1\times\cdots\times X_m}\left[\max_i y_i\right].$$

Performance of online algorithm

Performance is measured against that of the prophet.

- Prophet gets to see all realizations $w_i \sim X_i$ after they are sampled.
- Independent of ordering, she simply selects $w^* = \max_i w_i$.

Expected weight of chosen element for prophet is

$$\mathrm{OPT} = \mathbb{E}_{(y_1,\ldots,y_m)\sim X_1\times\cdots\times X_m}\left[\max_i y_i\right].$$

Expected weight of algorithm $\mathcal{A}$ (under worst-case arrival order) is

Performance of online algorithm

Performance is measured against that of the prophet.

- Prophet gets to see all realizations $w_i \sim X_i$ after they are sampled.
- Independent of ordering, she simply selects $w^* = \max_i w_i$.

Expected weight of chosen element for prophet is

$$\mathrm{OPT} = \mathbb{E}_{(y_1,\ldots,y_m)\sim X_1\times\cdots\times X_m}\left[\max_i y_i\right].$$

Expected weight of algorithm $\mathcal{A}$ (under worst-case arrival order) is

$$\mathrm{ALG} = \mathbb{E}_{(y_1,\ldots,y_m)\sim X_1\times\cdots\times X_m}\left[\min_\sigma w(\mathcal{A}(\sigma, y_1, \ldots, y_m))\right].$$

Performance of online algorithm

Performance is measured against that of the prophet.

- Prophet gets to see all realizations $w_i \sim X_i$ after they are sampled.
- Independent of ordering, she simply selects $w^* = \max_i w_i$.

Expected weight of chosen element for prophet is

$$\text{OPT} = \mathbb{E}_{(y_1,\ldots,y_m)\sim X_1\times\cdots\times X_m}\left[\max_i y_i\right].$$

Expected weight of algorithm $\mathcal{A}$ (under worst-case arrival order) is

$$\text{ALG} = \mathbb{E}_{(y_1,\ldots,y_m)\sim X_1\times\cdots\times X_m}\left[\min_\sigma w(\mathcal{A}(\sigma, y_1, \ldots, y_m))\right].$$

- With $w(\mathcal{A}(\sigma, y_1, \ldots, y_m))$ (expected) weight of set outputted by $\mathcal{A}$.

Performance of online algorithm

Performance is measured against that of the prophet.

- Prophet gets to see all realizations $w_i \sim X_i$ after they are sampled.
- Independent of ordering, she simply selects $w^* = \max_i w_i$.

Expected weight of chosen element for prophet is

$$\text{OPT} = \mathbb{E}_{(y_1,\dots,y_m)\sim X_1\times\cdots\times X_m}\left[\max_i y_i\right].$$

Expected weight of algorithm $\mathcal{A}$ (under worst-case arrival order) is

$$\text{ALG} = \mathbb{E}_{(y_1,\dots,y_m)\sim X_1\times\cdots\times X_m}\left[\min_\sigma w(\mathcal{A}(\sigma, y_1,\dots,y_m))\right].$$

- With $w(\mathcal{A}(\sigma, y_1, \dots, y_m))$ (expected) weight of set outputted by $\mathcal{A}$.

For $0 < \alpha < 1$, algorithm $\mathcal{A}$ is α-approximation if

Performance of online algorithm

Performance is measured against that of the prophet.

- Prophet gets to see all realizations $w_i \sim X_i$ after they are sampled.
- Independent of ordering, she simply selects $w^* = \max_i w_i$.

Expected weight of chosen element for prophet is

$$\text{OPT} = \mathbb{E}_{(y_1,\dots,y_m)\sim X_1\times\cdots\times X_m}\left[\max_i y_i\right].$$

Expected weight of algorithm $\mathcal{A}$ (under worst-case arrival order) is

$$\text{ALG} = \mathbb{E}_{(y_1,\dots,y_m)\sim X_1\times\cdots\times X_m}\left[\min_\sigma w(\mathcal{A}(\sigma, y_1, \dots, y_m))\right].$$

- With $w(\mathcal{A}(\sigma, y_1, \dots, y_m))$ (expected) weight of set outputted by $\mathcal{A}$.

For $0 < \alpha < 1$, algorithm $\mathcal{A}$ is α-approximation if

$$\text{ALG} \geq \alpha\text{OPT}$$

Performance of online algorithm

Performance is measured against that of the prophet.

- Prophet gets to see all realizations $w_i \sim X_i$ after they are sampled.
- Independent of ordering, she simply selects $w^* = \max_i w_i$.

Expected weight of chosen element for prophet is

$$\text{OPT} = \mathbb{E}_{(y_1,\dots,y_m)\sim X_1\times\dots\times X_m}\left[\max_i y_i\right].$$

Expected weight of algorithm $\mathcal{A}$ (under worst-case arrival order) is

$$\text{ALG} = \mathbb{E}_{(y_1,\dots,y_m)\sim X_1\times\dots\times X_m}\left[\min_\sigma w(\mathcal{A}(\sigma, y_1, \dots, y_m))\right].$$

- With $w(\mathcal{A}(\sigma, y_1, \dots, y_m))$ (expected) weight of set outputted by $\mathcal{A}$.

For $0 < \alpha < 1$, algorithm $\mathcal{A}$ is α-approximation if

$$\text{ALG} \geq \alpha\text{OPT}$$

- This is called a prophet inequality.

Example (Cont'd)

$E = \{e_1, e_2\}$ with following distributions.

Example (Cont'd)

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$.

Example (Cont'd)

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \tag{3}$$

Example (Cont'd)

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \tag{3}$$

$$w_2 \sim X_2 = \begin{cases} 1 + \delta & \text{with probability } 1. \end{cases} \tag{4}$$

Example (Cont'd)

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \quad (3)$$

$$w_2 \sim X_2 = \left\{ 1 + \delta \quad \text{with probability } 1. \right. \quad (4)$$

What can prophet get?

Example (Cont'd)

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \quad (3)$$

$$w_2 \sim X_2 = \begin{cases} 1 + \delta & \text{with probability } 1. \end{cases} \quad (4)$$

What can prophet get?

$$\max\{w_1, w_2\} = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 1 + \delta & \text{with probability } 1 - \epsilon \end{cases} .$$

Example (Cont'd)

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1-\epsilon \end{cases} \quad (3)$$

$$w_2 \sim X_2 = \begin{cases} 1+\delta & \text{with probability } 1. \end{cases} \quad (4)$$

What can prophet get?

$$\max\{w_1, w_2\} = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 1+\delta & \text{with probability } 1-\epsilon \end{cases}.$$

Then

$$\mathbb{E}_{(x_1,x_2)}[\max_i x_i] = \frac{1}{\epsilon} \times \epsilon + (1+\delta) \times (1-\epsilon) \to 2 \quad \text{as} \quad \epsilon, \delta \to 0.$$

Example (Cont'd)

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1 - \epsilon \end{cases} \tag{3}$$

$$w_2 \sim X_2 = \{ \ 1 + \delta \quad \text{with probability } 1. \tag{4}$$

What can prophet get?

$$\max\{w_1, w_2\} = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 1 + \delta & \text{with probability } 1 - \epsilon \end{cases}.$$

Then

$$\mathbb{E}_{(x_1, x_2)}[\max_i x_i] = \frac{1}{\epsilon} \times \epsilon + (1 + \delta) \times (1 - \epsilon) \to 2 \quad \text{as} \quad \epsilon, \delta \to 0.$$

Optimal algorithm $\mathcal{A}$ is to select e_2 (again, think about it).

Example (Cont'd)

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1-\epsilon \end{cases} \tag{3}$$

$$w_2 \sim X_2 = \{\ 1+\delta \quad \text{with probability } 1. \tag{4}$$

What can prophet get?

$$\max\{w_1, w_2\} = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 1+\delta & \text{with probability } 1-\epsilon \end{cases}.$$

Then

$$\mathbb{E}_{(x_1,x_2)}[\max_i x_i] = \frac{1}{\epsilon} \times \epsilon + (1+\delta) \times (1-\epsilon) \to 2 \quad \text{as} \quad \epsilon, \delta \to 0.$$

Optimal algorithm $\mathcal{A}$ is to select e_2 (again, think about it).

- In worst-case order (e_2, e_1).

Example (Cont'd)

$E = \{e_1, e_2\}$ with following distributions. Let $1 > \epsilon, \delta > 0$, and assume that $\frac{1}{\epsilon} > 1 + \delta$. Let

$$w_1 \sim X_1 = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 0 & \text{with probability } 1-\epsilon \end{cases} \quad (3)$$

$$w_2 \sim X_2 = \begin{cases} 1+\delta & \text{with probability 1.} \end{cases} \quad (4)$$

What can prophet get?

$$\max\{w_1, w_2\} = \begin{cases} \frac{1}{\epsilon} & \text{with probability } \epsilon \\ 1+\delta & \text{with probability } 1-\epsilon \end{cases}.$$

Then

$$\mathbb{E}_{(x_1,x_2)}[\max_i x_i] = \frac{1}{\epsilon} \times \epsilon + (1+\delta) \times (1-\epsilon) \to 2 \quad \text{as} \quad \epsilon, \delta \to 0.$$

Optimal algorithm $\mathcal{A}$ is to select e_2 (again, think about it).

- In worst-case order (e_2, e_1).

Then

$$\mathbb{E}_{(x_1,x_2)}[w(\mathcal{A}(x_1, x_2))] = 1.$$

- I.e., optimal algorithm only half as bad as prophet ($\alpha = \frac{1}{2}$).

Outline for remaining lectures

Rough outline of upcoming lectures

Rough outline of upcoming lectures

Related to this lecture, we will see:

Rough outline of upcoming lectures

Related to this lecture, we will see:

- Algorithms for combinatorial settings where more than one element can be selected (in uniform random arrival model).

Rough outline of upcoming lectures

Related to this lecture, we will see:

- Algorithms for combinatorial settings where more than one element can be selected (in uniform random arrival model).
  - Online bipartite matching.

Rough outline of upcoming lectures

Related to this lecture, we will see:

- Algorithms for combinatorial settings where more than one element can be selected (in uniform random arrival model).
  - Online bipartite matching.
    - $\frac{1}{e}$-approximation (significant generalization of secretary problem).

Rough outline of upcoming lectures

Related to this lecture, we will see:

- Algorithms for combinatorial settings where more than one element can be selected (in uniform random arrival model).
  - Online bipartite matching.
    - $\frac{1}{e}$-approximation (significant generalization of secretary problem).
  - Matroid secretary problems.

Rough outline of upcoming lectures

Related to this lecture, we will see:

- Algorithms for combinatorial settings where more than one element can be selected (in uniform random arrival model).
  - Online bipartite matching.
    - $\frac{1}{e}$-approximation (significant generalization of secretary problem).
  - Matroid secretary problems.
    - Still mayor open problem to find constant-factor approximation.

Rough outline of upcoming lectures

Related to this lecture, we will see:

- Algorithms for combinatorial settings where more than one element can be selected (in uniform random arrival model).
  - Online bipartite matching.
    - $\frac{1}{e}$-approximation (significant generalization of secretary problem).
  - Matroid secretary problems.
    - Still mayor open problem to find constant-factor approximation.
- Algorithm for single element prophet inequality (with $\alpha = \frac{1}{2}$).

Rough outline of upcoming lectures

Related to this lecture, we will see:

- Algorithms for combinatorial settings where more than one element can be selected (in uniform random arrival model).
  - Online bipartite matching.
    - $\frac{1}{e}$-approximation (significant generalization of secretary problem).
  - Matroid secretary problems.
    - Still mayor open problem to find constant-factor approximation.
- Algorithm for single element prophet inequality (with $\alpha = \frac{1}{2}$).

Problems can also be turned into *(online) auction* problems.

Rough outline of upcoming lectures

Related to this lecture, we will see:

- Algorithms for combinatorial settings where more than one element can be selected (in uniform random arrival model).
  - Online bipartite matching.
    - $\frac{1}{e}$-approximation (significant generalization of secretary problem).
  - Matroid secretary problems.
    - Still mayor open problem to find constant-factor approximation.
- Algorithm for single element prophet inequality (with $\alpha = \frac{1}{2}$).

Problems can also be turned into *(online) auction* problems.

- Will see some (offline) mechanism design basics next week.