Sequential Decision Making: Prophets and Secretaries II – Secretary Problems

Matt Weinberg

Princeton University

Online Selection Problems: Secretary Problems

Offline:

- Every secretary i has a weight w_i (chosen by adversary, unknown to you).
- Secretaries permuted randomly.

Online:

- Secretaries revealed one at a time. You learn their weight.
- Immediately and irrevocably decide to hire or not.
- May only hire one secretary!

Goal: Maximize probability of selecting max-weight element.

w = 6 4 7 8 9

Online Selection Problems: Secretary Problems

Offline:

- Every secretary i has a weight w_i (chosen by adversary, unknown to you).
- Secretaries permuted randomly.

Online:

- Secretaries revealed one at a time. You learn their weight.
- Immediately and irrevocably decide to hire or not.
- May only hire one secretary!

Goal: Maximize probability of selecting max-weight element.

t = 1 2 3

w = 7 6 8

Secretary Problem: Some Observations

Observation 1: optimal policy w.l.o.g. only accepts elements that have the largest weight seen so far.

Proof: Any element that isn't the largest so far clearly isn't the largest. So don't lose anything by rejecting it. Might gain something by accepting something else in future.

Secretary Problem: Some Observations

Observation 1: optimal policy w.l.o.g. only accepts elements that have the largest weight seen so far.

Observation 2: can find optimal policy via dynamic programming.

Proof:

- If we make it to step n, see the largest element so far, clearly accept it.
- If we make it to step n-1, see the largest element so far, we can:
  - Accept it, get the largest element w.p. 1-1/n (as last element isn't true largest).
  - Reject, get largest element w.p. 1/n.
  - So accept.
- If we make it to step i, see largest element so far, we can:
  - Accept it, get the largest element w.p. i/n (as long as largest element is in first i steps).
  - Reject, get largest element w.p. f(i,n) (computed by dynamic program).
    - f(i,n) = Pr[optimal policy selects largest element, conditioned on reaching step i+1].
  - So accept iff i/n > f(i,n).

Secretary Problems: Some Observations

Observation 1: optimal policy w.l.o.g. only accepts elements that have the largest weight seen so far.

Observation 2: can find optimal policy via dynamic programming.

- If we make it to step i, see largest element so far, we can:
  - Accept it, get the largest element w.p. i/n (as long as largest element is in first i steps).
  - Reject, get largest element w.p. f(i,n) (computed by dynamic program).
    - f(i,n) = Pr[optimal policy selects largest element, conditioned on reaching step i+1].
  - So accept iff i/n > f(i,n).

Observation 3: if i < j, then f(i,n) $\geq$ f(j,n).

Proof:

One policy starting from i+1: reject everything until reach step j+1. Then run optimal policy starting from j+1. Succeeds w.p. exactly f(j,n).

Secretary Problems: Some Observations

Observation 1: optimal policy w.l.o.g. only accepts elements that have the largest weight seen so far.

Observation 2: can find optimal policy via dynamic programming.

- If we make it to step i, see largest element so far, we can:
  - Accept it, get the largest element w.p. i/n (as long as largest element is in first i steps).
  - Reject, get largest element w.p. $f(i,n)$ (computed by dynamic program).
    - $f(i,n)$ = Pr[optimal policy selects largest element, conditioned on reaching step i+1].
  - So accept iff $i/n > f(i,n)$.

Observation 3: if $i < j$, then $f(i,n) \geq f(j,n)$.

Observation 4: Optimal policy sets cutoff time T. Rejects everything before (or during) time T. Accepts any element after T iff highest so far.

Proof: Optimal policy accepts i iff $i/n > f(i,n)$ (and i highest so far).

If $i < j$, and optimal policy would accept i, then: $j/n > i/n > f(i,n) \geq f(j,n)$.

So optimal policy would accept j (if j was highest so far) too.

Secretary Problem: A Competitive Policy

Observation: Optimal policy sets cutoff time T. Rejects everything before time T. Accepts any element after T iff highest so far.

So just need to find optimal T.

(suboptimal) Proposition: Optimal policy selects highest element w.p. $\geq 1/4$.

Proof: (randomly) set T $\leftarrow$ Binom(n,1/2).

- (every element comes before T w.p. exactly 1/2 independently of the others).

If highest element comes after T and 2^{nd} highest comes before T, definitely select highest element.

Above occurs w.p. 1/4.

Secretary Problem: Optimal Policy

Observation: Optimal policy sets cutoff time T. Rejects everything before time T. Accepts any element after T iff highest so far.

(suboptimal) Proposition: Optimal policy selects highest element w.p. $\geq 1/4$.

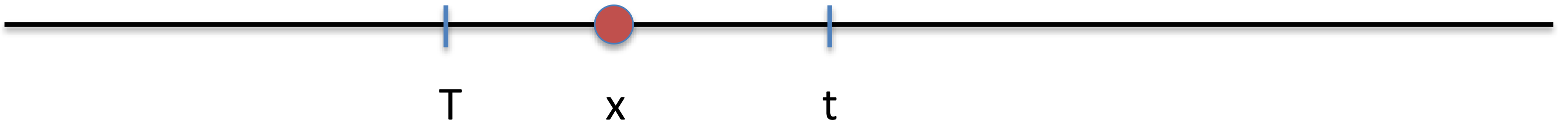
Theorem [Dynkin 63]: Optimal policy selects highest element w.p. $\approx 1/e$.

Lemma: For any cutoff time T, Pr[reach time t+1] = $\min\{T/t, 1\}$.

Proof: If $t < T$, then clearly we will reach time t.

If $t \geq T$, then we reach time t+1 iff no element between T and t is the highest so far.

This happens iff the highest element from the first t arrives in the first T steps.



Secretary Problem: Optimal Policy

Observation: Optimal policy sets cutoff time T. Rejects everything before time T. Accepts any element after T iff highest so far.

(suboptimal) Proposition: Optimal policy selects highest element w.p. $\geq 1/4$.

Theorem [Dynkin 63]: Optimal policy selects highest element w.p. $\approx 1/e$.

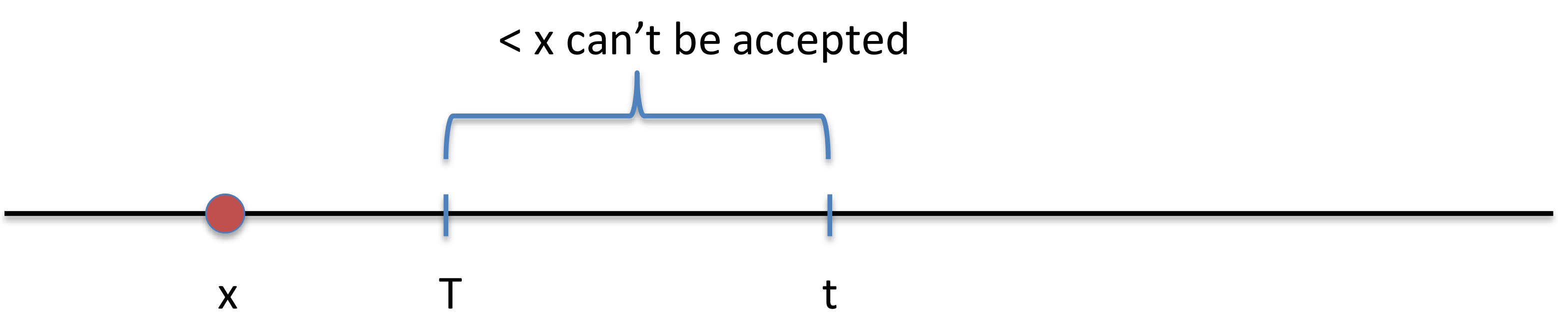
Lemma: For any cutoff time T, Pr[reach time t+1] = $\min\{T/t, 1\}$.

Proof: If $t < T$, then clearly we will reach time t.

If $t \geq T$, then we reach time t+1 iff no element between T and t is the highest so far.

This happens iff the highest element from the first t arrives in the first T steps.

→ Reach time t+1 w.p. exactly T/t.



Secretary Problem: Optimal Policy

Observation: Optimal policy sets cutoff time T. Rejects everything before time T. Accepts any element after T iff highest so far.

(suboptimal) Proposition: Optimal policy selects highest element w.p. $\geq$ 1/4.

Theorem [Dynkin 63]: Optimal policy selects highest element w.p. $\approx$ 1/e.

Lemma: For any cutoff time T, Pr[reach time t+1] = min{T/t, 1}.

Corollary: For any cutoff time T, Pr[select highest] = $\sum_{t=T+1}^{n} T/tn$.

Proof: Pr[select highest] = $\sum_t \text{Pr[highest arrives at t AND is selected]}$.

If t $\leq$ T, and the highest arrives at time t, then it isn't selected.

If t > T, and the highest arrives at time t, then it is selected iff we reach time t.

Pr[highest arrives at t] = 1/n for all t.

So Pr[select highest] = $\sum_{t=T+1}^{n} T/tn$.

Secretary Problem: Optimal Policy

Observation: Optimal policy sets cutoff time T. Rejects everything before time T. Accepts any element after T iff highest so far.

(suboptimal) Proposition: Optimal policy selects highest element w.p. $\geq$ 1/4.

Theorem [Dynkin 63]: Optimal policy selects highest element w.p. $\approx$ 1/e.

Lemma: For any cutoff time T, Pr[reach time t+1] = min{T/t, 1}.

Corollary: For any cutoff time T, Pr[select highest] = $\sum_{t=T+1}^{n} T/tn$.

Proof of Theorem: For large n, $\sum_{t=T+1}^{n} T/tn \approx \int_{T}^{n} T/tn \, dt = (T/n)\ln(n/T)$.

$\underset{T\leq n}{\operatorname{argmax}}\{(T/n)\ln(n/T)\} = n/e.\ \ln(e)\,/e = 1/e.$

Secretary Problem: Optimal Policy

Theorem [Dynkin 63]: Optimal policy selects highest element w.p. ≈ 1/e.

- Also guarantees E[Gambler] $\geq \max_i \{w_i\} / e$.

Compare to Prophet Inequalities:

- Both: simple policies get constant (and optimal) competitive ratios.
- Secretary problem: simple policy is actually optimal on all instances.
- Prophet inequalities: simple policy gets optimal competitive ratio, but is not necessarily optimal on every instance (versus dynamic program).

Multiple Choice Secretary Problems

Rest of talk: What if multiple choices?

Offline:

- Secretary i has a weight w_i (chosen by adversary).
- Adversary chooses **feasibility constraints**: which secretaries can simultaneously hire? (known to you).
- Secretaries permuted randomly.

Online:

- Secretaries revealed one at a time. You learn their weight.
- Immediately and irrevocably decide to hire or not.
- H = all hired secretaries. Must maintain H feasible **at all times**.

Goal: Maximize $\mathrm{E}[\sum_{i \in H} w_i]$ - expected weight of hires.

- Compete with $\max_{\text{feasible } H} \{\sum_{i \in H} w_i\}$.

State-of-the-art (non-exhaustive)

Feasibility	Approximation Guarantee (most omitted lower bounds are e)
k-Uniform	$1+\Theta(1/\sqrt{k})$ [Kleinberg 05].
Matroids	O(loglog(rank)) [Lachish 14, Feldman-Svensson-Zenklusen 15].
Graphic Matroids	2e [Korula-Pal 09],
Transversal Matroids	e [Kesselheim-Radke-Tonnis-Vocking 13].
Laminar Matroids	9.6 [Ma-Tang-Wang 13].
Regular Matroids	9e [Dinitz-Kortsarz 14].
Knapsack Contstraints	10e [Babaioff-Immorlica-Kempe-Kleinberg 07]. (each element has cost c_i, set is feasible iff total cost at most B).
Downwards Closed	O(log n log r) [Rubinstein 16]. n = #elements, r = largest feasible set.

Transversal Matroid: Elements = left-hand nodes of bipartite graph. Set of nodes S is feasible iff exists matching of size |S| from S to right-hand nodes.

Regular Matroid: For all fields F, exists vector space V over F, elements can be mapped to vectors in V such that S feasible iff corresponding vectors linearly independent.

One Slide Primer on Matroids

Matroid: Feasibility constraints such that S, T feasible with |S|>|T| → exists i ∈ S such that $\mathrm{T} \cup \{i\}$ feasible. Downwards closed.

Rank: Rank(S) = $\max_{T \subseteq S, T \text{ is feasible}} \{|T|\}$.

- Think of vector spaces.
- Feasible sets have |S| = rank(S).
- Sometimes call a subset B of S with |B|=rank(B)=rank(S) a **basis** of S.

Span: Span(S) = $\{i | \mathrm{rank}(S \cup \{i\}) = \mathrm{rank(S)}\}$.

- Think of vector spaces.

Theorem [Edmonds 70]: The greedy algorithm finds the max-weight feasible set in all matroids.

- Sort in decreasing order of weight. Accept any feasible element.
- Feasible to accept i iff i not in span of earlier elements.
- Implies i in max-weight basis iff i not spanned by heavier elements.

Rest of Talk – Some Examples

Goal: Get a taste for different techniques via:

- e-approximation for k-uniform [**Babaioff-Immorlica-Kempe-Kleinberg 07**].
- 2e-approximation for graphic matroids [**Korula-Pal 09**].
- 4-approximation for matroids in “Free-order model” [**Jaillet-Soto-Zenklusen 13**].

Note: Won’t see applications of deep matroid theory in this talk.

- [**Soto 11, Dinitz-Kortsarz 14, Huynh-Nelson 16**] use decomposition theorems and matroid minor theory [e.g. **Seymour 80**].

K-Uniform Matroids

Theorem [Babaioff-Immorlica-Kempe-Kleinberg 07]: e-approximation for k-uniform.

Algorithm: When the element at time step t is processed:

- If t < n/e, reject.
- If the current element is not in the top k so far, reject.
- If the previous k^{th} highest element arrived **between n/e and t**, reject.
- Else, accept.

Examples: k=2, n=5. Pretend 2 < 5/e < 3.

w = 1 2 3 4 5

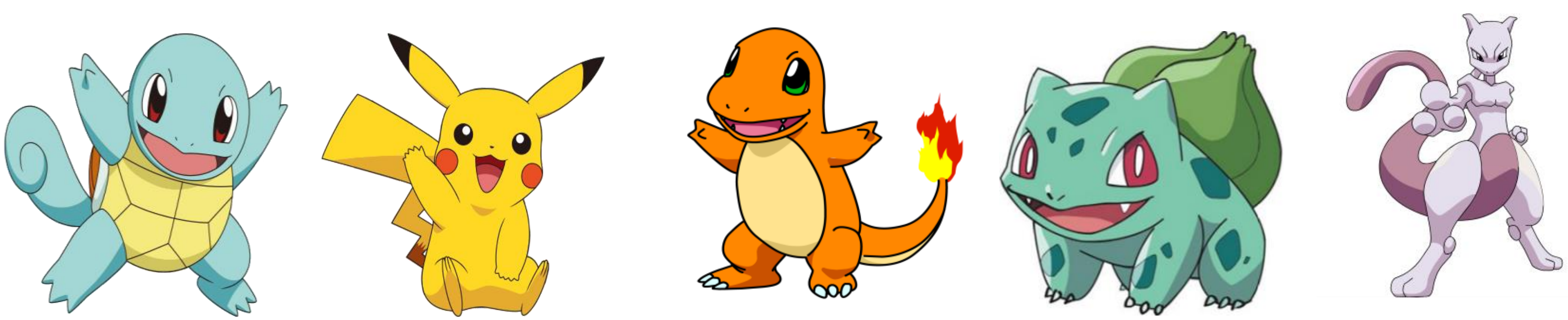
K-Uniform Matroids

Theorem [Babaioff-Immorlica-Kempe-Kleinberg 07]: e-approximation for k-uniform.

Algorithm: When the element at time step t is processed:

- If t < n/e, reject.
- If the current element is not in the top k so far, reject.
- If the previous k^{th} highest element arrived **between n/e and t**, reject.
- Else, accept.

Examples: k=2, n=5. Pretend 2 < 5/e < 3.



w = 2 4 3 1 5

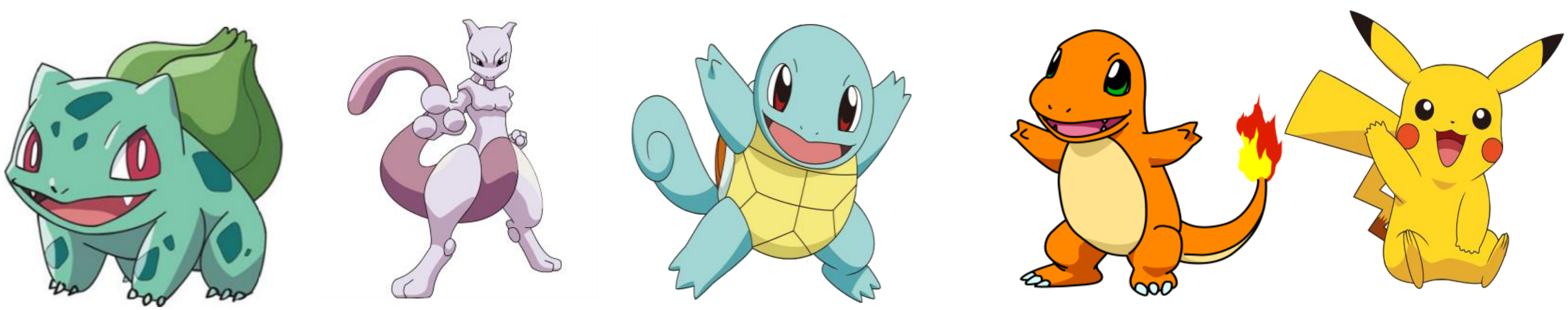
K-Uniform Matroids

Theorem [Babaioff-Immorlica-Kempe-Kleinberg 07]: e-approximation for k-uniform.

Algorithm: When the element at time step t is processed:

- If t < n/e, reject.
- If the current element is not in the top k so far, reject.
- If the previous k^{th} highest element arrived **between n/e and t**, reject.
- Else, accept.

Examples: k=2, n=5. Pretend 2 < 5/e < 3.



w = 1 5 2 3 4

K-Uniform Matroids

Theorem [Babaioff-Immorlica-Kempe-Kleinberg 07]: e-approximation for k-uniform.

Algorithm: When the element at time step t is processed:

- If t < n/e, reject.
- If the current element is not in the top k so far, reject.
- If the previous k^{th} highest element arrived **between n/e and t**, reject.
- Else, accept.

Observation: Algorithm never accepts more than k elements.

Proof: Let S = top k elements arriving before n/e.

Every accepted element “kicks out” an element of S from the top k.

K-Uniform Matroids

Theorem [Babaioff-Immorlica-Kempe-Kleinberg 07]: e-approximation for k-uniform.

Algorithm: When the element at time step t is processed:

- If t < n/e, reject.
- If the current element is not in the top k so far, reject.
- If the previous k^{th} highest element arrived **between n/e and t**, reject.
- Else, accept.

Observation: Algorithm never accepts more than k elements.

Proposition: Every element in the true top k is accepted w. p. $\geq 1/e$.

Proof: If i is in the true top k, then i is always in the top k so far when it arrives.

If i arrives at t > n/e, it is accepted iff the k^{th} highest element from steps {1,…,t-1} arrived before n/e. Pr[this occurs] = (n/e)/t.

So $\mathrm{Pr}[\mathrm{i\ accepted}|\mathrm{i\ in\ true\ top\ k}] = \sum_{\lceil n/e \rceil}^{n} 1/(et) \approx \int_{n/e}^{n} 1/(et)\, dt = 1/e$.

K-Uniform Matroids

Theorem [Babaioff-Immorlica-Kempe-Kleinberg 07]: e-approximation for k-uniform.

Algorithm: When the element at time step t is processed:

- If t < n/e, reject.
- If the current element is not in the top k so far, reject.
- If the previous $\mathrm{k^{th}}$ highest element arrived **between n/e and t**, reject.
- Else, accept.

Observation: Algorithm never accepts more than k elements.

Proposition: Every element in the true top k is accepted w. p. $\geq 1/e$.

Proof of Theorem: Observation → algorithm is feasible. Proposition → gets e-approximation.

K-Uniform Matroids

Theorem [Babaioff-Immorlica-Kempe-Kleinberg 07]: e-approximation for k-uniform.

Algorithm: When the element at time step t is processed:

- If t < n/e, reject.
- If the current element is not in the top k so far, reject.
- If the previous k^{th} highest element arrived **between n/e and t**, reject.
- Else, accept.

Underlying Technique: "samples" = max-weight feasible set of elements before n/e.

Whenever you accept an element, charge it to a sample. No sample charged twice.

Charge samples consistently to maintain feasibility of accepted elements.

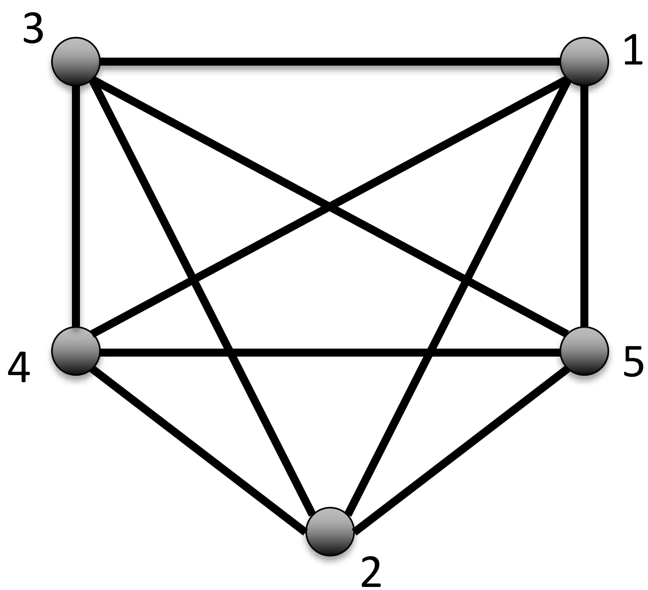
Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v} \text{ comes before w in ordering above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Example:



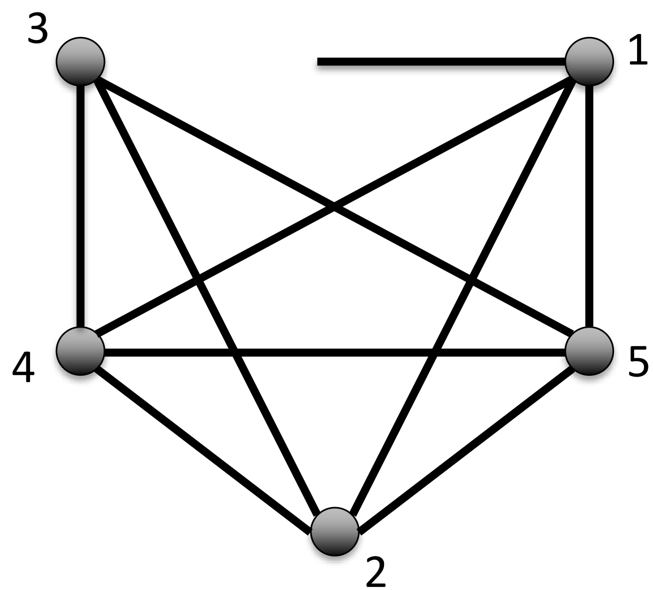
Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v} \text{ comes before w in ordering above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Example:



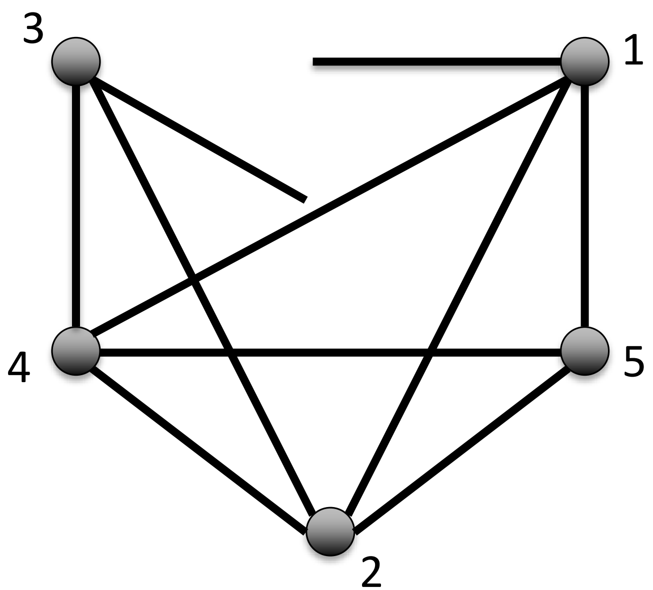
Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v} \text{ comes before w in ordering above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Example:



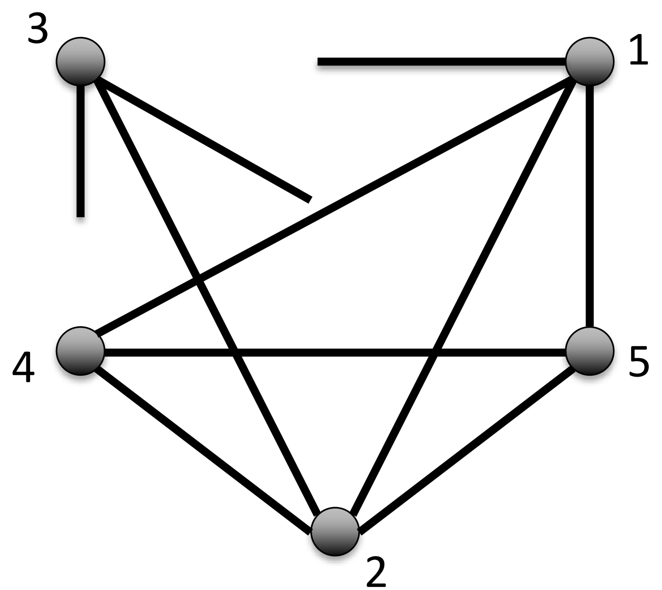
Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v}\text{ comes before w in ordering above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Example:



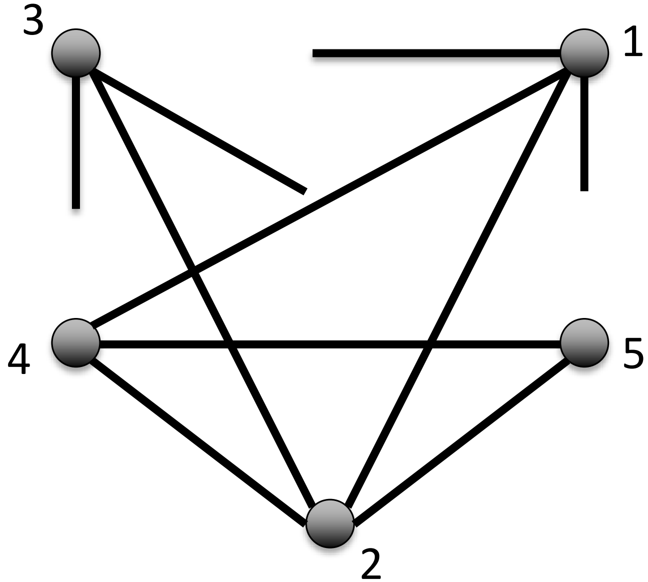
Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v} \text{ comes before w in ordering above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Example:



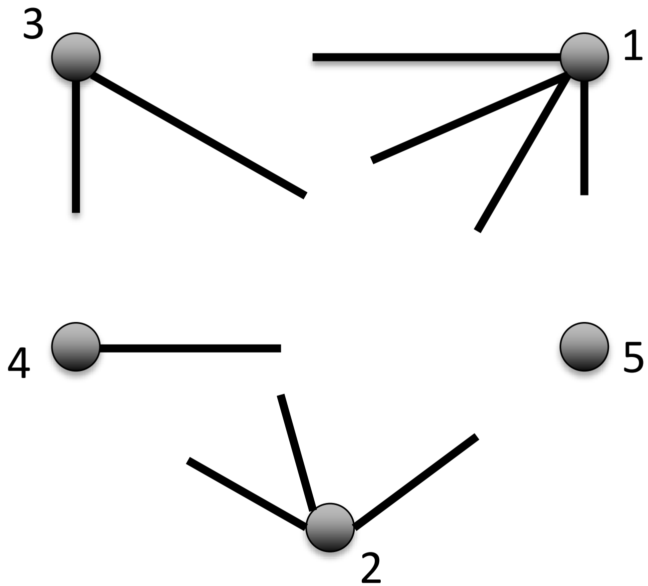
Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v}\text{ comes before w in ordering above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Example:



Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v}\text{ comes before w in ordering above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Observation: Algorithm always accepts an acyclic subgraph.

Proof: Assume for contradiction that algorithm accepts a cycle.

Let v be earliest node according to offline ordering in that cycle.

Then two edges were accepted from E_v. Contradiction.

Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

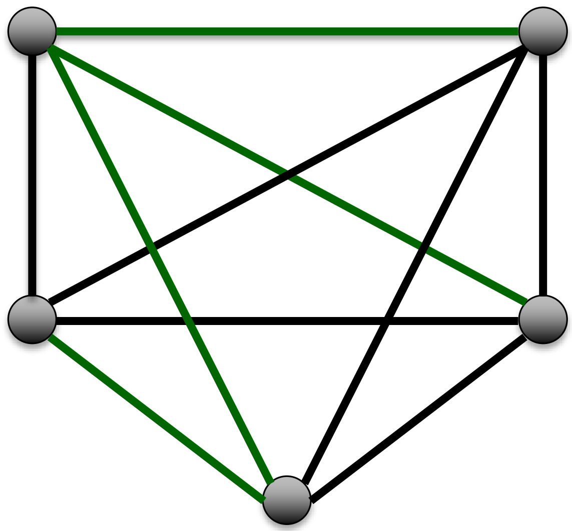
Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v\ comes\ before\ w\ in\ ordering\ above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Observation: Algorithm always accepts an acyclic subgraph.

Intuition for Algorithm:

If we get lucky and choose an ordering that recursively selects leaves of max-weight spanning tree, get each edge w.p. 1/e.



Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

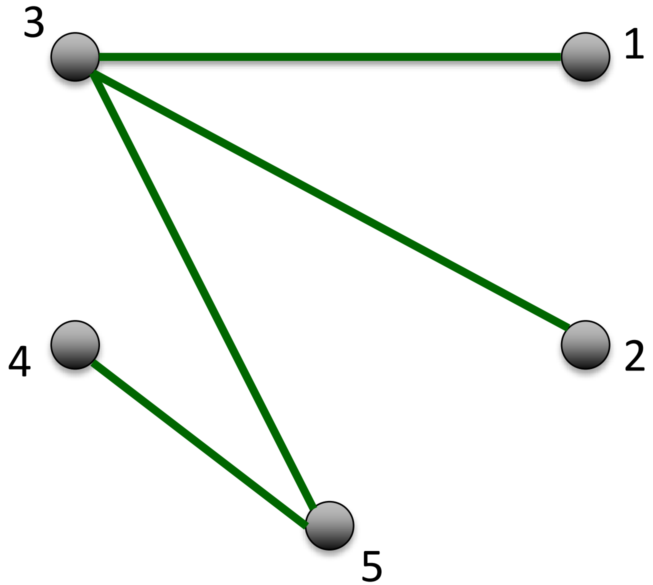
Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v} \text{ comes before w in ordering above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Observation: Algorithm always accepts an acyclic subgraph.

Intuition for Algorithm:

If we get lucky and choose an ordering that recursively selects leaves of max-weight spanning tree, get each edge w.p. 1/e.



Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

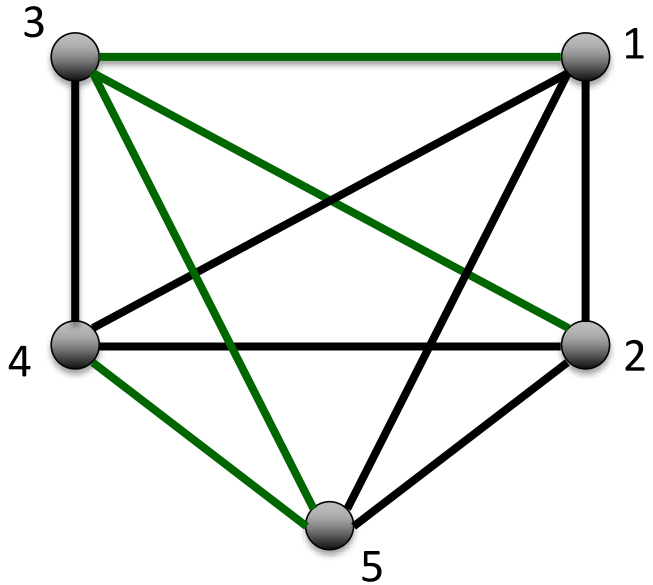
Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v} \text{ comes before } \mathrm{w} \text{ in ordering above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Observation: Algorithm always accepts an acyclic subgraph.

Intuition for Algorithm:

If we get lucky and choose an ordering that recursively selects leaves of max-weight spanning tree, get each edge w.p. 1/e.



Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

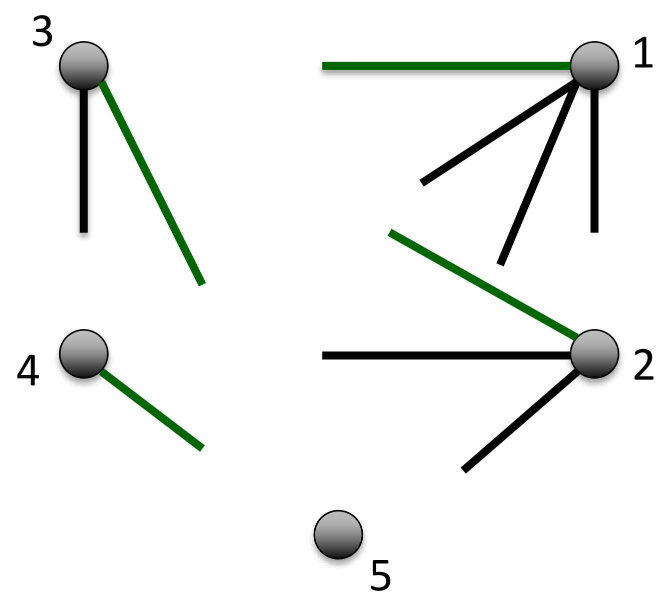
Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v} \text{ comes before w in ordering above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Observation: Algorithm always accepts an acyclic subgraph.

Intuition for Algorithm:

If we get lucky and choose an ordering that recursively selects leaves of max-weight spanning tree, get each edge w.p. 1/e.



Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v}\text{ comes before w in ordering above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Observation: Algorithm always accepts an acyclic subgraph.

Proof of Theorem: E[ALG] = $\sum_v E[\max_{i \in E_v}\{w_i\} / e]$.

Let v = leaf in MST, e = MST edge adjacent to v. Then $E\left[\max_{i \in E_v}\{w_i\}\right] \geq w_e/2$.

Let v become leaf in MST if all leaves are removed, e = MST edge adjacent to v (once leaves are removed). Then $E\left[\max_{i \in E_v}\{w_i\}\right] \geq w_e/2$.

Repeat above reasoning for all nodes. Eventually all edges in MST covered by some v.

Implies $\sum_v E[\max_{i \in E_v}\{w_i\}] \geq OPT/2$.

Graphic Matroids

Theorem [Korula-Pal 09]: 2e-approximation for graphic matroids.

Algorithm:

- Pick a random ordering of nodes (offline).
- Let $E_v = \{\mathrm{e} = (\mathrm{v}, \mathrm{w}), \mathrm{v\ comes\ before\ w\ in\ ordering\ above}\}$ (offline).
- Run Dynkin's 1-uniform algorithm on each E_v (online).

Underlying Technique: (randomly) restrict feasible sets to something simpler.

Disjoint union of many smaller, simpler subproblems.

Any union of feasible solutions to simpler problems is feasible.

- In this case, each subproblem = 1-uniform matroid.
- [**Soto 11, Lachish 14, Dinitz-Kortsarz 14, Feldman-Svensson-Zenklusen 15, Huynh-Nelson 16**] use similar high-level decomposition approach, but decompose differently. Subproblems more complex, still solvable.

Free-Order Model

Offline:

- Secretary i has a weight w_i (chosen by adversary).
- Adversary chooses **feasibility constraints**: which secretaries can simultaneously hire? (known to you).

Online:

- **You choose** which secretary to interview one at a time. You learn their weight.
- Immediately and irrevocably decide to hire or not.
- H = all hired secretaries. Must maintain H feasible **at all times**.

Goal: Maximize $\mathrm{E}[\sum_{i \in H} w_i]$ - expected weight of hires.

- Compete with $\max_{\text{feasible } H} \{\sum_{i \in H} w_i\}$.

Note: Restricted to 1-uniform matroids, still get original secretary problem.

- Because all elements identical.

Free-Order Model

Definition: Let $S = \{i_1, \dots, i_k\}$ be any set with $w_{i_1} > w_{i_2} > \cdots > w_{i_k}$. Let j = $\min_j\{j \mid i \in \mathrm{span}(\{i_1, \dots, i_j\})\}$. Then **Price**(i,S) = w_{i_j}.

- Span(V) = $\{i \mid \mathrm{rank}(V \cup \{i\}) = \mathrm{rank}(V)\}$.

Examples:

- k-uniform: Price(i,S) = $\mathrm{k^{th}}$ highest element of S.
- Graphic: Price(i,S) = min-weight edge on cycle formed by adding i to MST(S).
- Observation: i in max-weight basis of $S \cup \{i\}$ iff w_i > Price(i,S).

Free-Order Model

Definition: Let $S = \{i_1, \ldots, i_k\}$ be any set with $w_{i_1} > w_{i_2} > \cdots > w_{i_k}$. Let j = $\min_j\{j \mid i \in \text{span}(\{i_1, \ldots, i_j\})\}$. Then **Price**(i,S) = w_{i_j}.

- Span(V) = $\{i \mid \text{rank}(V \cup \{i\}) = \text{rank}(V)\}$.

Theorem [Jaillet-Soto-Zenklusen 13]: 4-approximation for matroids (free-order).

Algorithm: Let S = random Binom(n, 1/2) elements.

- Process all elements in S first, reject everything.
- Process remaining elements in decreasing order of Price(i,S).
- Accept i iff i in max-weight basis so far (and feasible to accept i).

Examples: k-uniform - once S is chosen:

- Span($\{i_1\}$) = ∅.
- Span(V) = ∅ if |V| < k.
- Span(V) = entire ground set if |V| = k.
- So process elements in arbitrary order.

Free-Order Model

Definition: Let $S = \{i_1, \ldots, i_k\}$ be any set with $w_{i_1} > w_{i_2} > \cdots > w_{i_k}$. Let j = $\min_j\{j \mid i \in \mathrm{span}(\{i_1, \ldots, i_j\})\}$. Then **Price**(i,S) = w_{i_j}.

- Span(V) = $\{i \mid \mathrm{rank}(V \cup \{i\}) = \mathrm{rank}(V)\}$.

Theorem [Jaillet-Soto-Zenklusen 13]: 4-approximation for matroids (free-order).

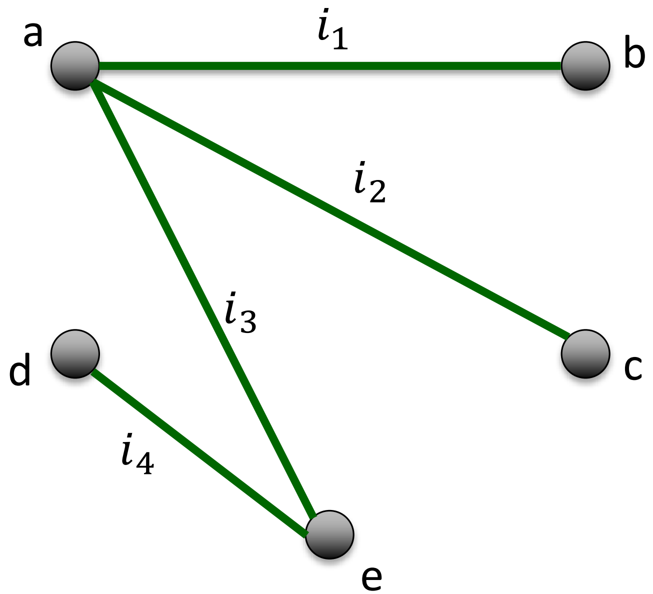
Algorithm: Let S = random Binom(n, 1/2) elements.

- Process all elements in S first, reject everything.
- Process remaining elements in decreasing order of Price(i,S).
- Accept i iff i in max-weight basis so far (and feasible to accept i).

Examples: graphic matroids - once S is chosen:

Process edges in the following order:

- Any copies of (a,b).
- Any edge with both endpoints in {a,b,c}.
- Any edge with both endpoints in {a,b,c,e}.
- All remaining edges.



Free-Order Model

Definition: Let $S = \{i_1, \ldots, i_k\}$ be any set with $w_{i_1} > w_{i_2} > \cdots > w_{i_k}$. Let j = $\min_j\{j \mid i \in \text{span}(\{i_1, \ldots, i_j\})\}$. Then **Price**(i,S) = w_{i_j}.

- Span(V) = $\{i \mid \text{rank}(V \cup \{i\}) = \text{rank}(V)\}$.

Theorem [Jaillet-Soto-Zenklusen 13]: 4-approximation for matroids (free-order).

Algorithm: Let S = random Binom(n, 1/2) elements.

- Process all elements in S first, reject everything.
- Process remaining elements in decreasing order of Price(i,S).
- Accept i iff i in max-weight basis so far (and feasible to accept i).

Observation: Element i is **only** accepted if w_i > Price(i,S).

Proof: Otherwise, i isn't in max-weight basis so far.

Free-Order Model

Definition: Let $S = \{i_1, \dots, i_k\}$ be any set with $w_{i_1} > w_{i_2} > \cdots > w_{i_k}$. Let j = $\min_j\{j \mid i \in \text{span}(\{i_1, \dots, i_j\})\}$. Then **Price**(i,S) = w_{i_j}.

- Span(V) = $\{i \mid \text{rank}(V \cup \{i\}) = \text{rank}(V)\}$.

Theorem [Jaillet-Soto-Zenklusen 13]: 4-approximation for matroids (free-order).

Algorithm: Let S = random Binom(n, 1/2) elements.

- Process all elements in S first, reject everything.
- Process remaining elements in decreasing order of Price(i,S).
- Accept i iff i in max-weight basis so far (and feasible to accept i).

Observation: Element i is **only** accepted if w_i > Price(i,S).

Lemma: If Price(i,S) > Price(i,$\bar{S}$-{i}), then feasible to accept i.

Proof: By Algorithm, all j processed before i have Price(j,S)≥Price(i,S).

In order for any j to be accepted, must have w_j>Price(j,S).

Therefore, all elements accepted before i arrives have w_j>Price(i,S).

If infeasible to accept i, then such elements span i, and Price(i,$\bar{S}$-{i}) > Price(i,S).

Free-Order Model

Definition: Let $S = \{i_1, \ldots, i_k\}$ be any set with $w_{i_1} > w_{i_2} > \cdots > w_{i_k}$. Let j = $\min_j\{j \mid i \in \text{span}(\{i_1, \ldots, i_j\})\}$. Then **Price**(i,S) = w_{i_j}.

- Span(V) = $\{i \mid \text{rank}(V \cup \{i\}) = \text{rank}(V)\}$.

Theorem [Jaillet-Soto-Zenklusen 13]: 4-approximation for matroids (free-order).

Algorithm: Let S = random Binom(n, 1/2) elements.

- Process all elements in S first, reject everything.
- Process remaining elements in decreasing order of Price(i,S).
- Accept i iff i in max-weight basis so far (and feasible to accept i).

Observation: Element i is **only** accepted if w_i > Price(i,S).

Lemma: If Price(i,S) > Price(i,$\bar{S}$-{i}), then feasible to accept i.

Proof of Theorem: i∈ $\bar{S}$ w.p. 1/2. Independently, Price(i,S)>Price(i,$\bar{S}$-{i}) w.p. 1/2.

For any i in true max-weight basis, will accept i whenever both occur.

Free-Order Model

Definition: Let $S = \{i_1, \ldots, i_k\}$ be any set with $w_{i_1} > w_{i_2} > \cdots > w_{i_k}$. Let j = $\min_j\{j \mid i \in \text{span}(\{i_1, \ldots, i_j\})\}$. Then **Price**(i,S) = w_{i_j}.

- Span(V) = $\{i \mid \text{rank}(V \cup \{i\}) = \text{rank}(V)\}$.

Theorem [Jaillet-Soto-Zenklusen 13]: 4-approximation for matroids (free-order).

Algorithm: Let S = random Binom(n, 1/2) elements.

- Process all elements in S first, reject everything.
- Process remaining elements in decreasing order of Price(i,S).
- Accept i iff i in max-weight basis so far (and feasible to accept i).

Underlying Technique: Couple "good" samples with "bad" samples – if S is bad, then $\bar{S}$ is good.

Recap

What we saw:

- e-approximation for k-uniform matroids.
  - Main idea: charge different element in max-weight basis of “samples” whenever accept.
- 2e-approximation for graphic matroids.
  - Main idea: stricter feasibility constraints that decompose into smaller subproblems.
- 4-approximation for matroids in free-order model.
  - Main idea: couple “good” samples with “bad” samples.

Again, not exhaustive list of high level techniques, but pretty good sample.

Related Results/Problems

What if objective function isn't linear in weights?

- Maybe submodular instead?
- Lots of works show constant factor approximations [**Gupta-Roth-Schoenebeck-Talwar 10, Bateni-Hajiaghgayi-Zadimoghaddam 10, Feldman-Naor-Schwartz 11, Ma-Tang-Wang 16, Kesselheim-Tonnis 16**] (non-exhaustive list).
- [**Feldman-Zenklusen 15**]: reduction from submodular to linear objective.

What if weights randomly assigned to elements?

- Harder for adversary to generate hard instances.
- O(1) approximation for all matroids, even with adversarial order [**Soto 11, Oveis Gharan-Vondrak 11**].

What if ordering not completely random?

- $\Theta(\log\log(n))$ entropy: O(1) approximation [**Kesselheim-Kleinberg-Niazadeh 15**].
- Note uniform random order has $\Theta(n\log n)$ entroy.

Some Current Directions

O(1) approximation for all matroids in standard model?

- Interesting special case: representable matroids (vector spaces).
  - i.e. pick your favorite field F. O(1) approximation for all vector spaces over F?

Tight e-approximation in any special cases?

- Currently known for transversal matroids [**Kesselheim-Radke-Tonnis-Vocking 13**].
- What about graphic? Laminar? Free-order model?

Any lower bounds?

- > e for classes of simple algorithms?

Thanks for listening!