Submodular Functions, Optimization, and Applications to Machine Learning

— Spring Quarter, Lecture 13 —

http://www.ee.washington.edu/people/faculty/bilmes/classes/ee563_spring_2018/

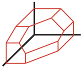
Prof. Jeff Bilmes

University of Washington, Seattle
Department of Electrical Engineering
http://melodi.ee.washington.edu/~bilmes

May 9th, 2018

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$$

$= f(A_r) + 2f(C) + f(B_r)$ $\quad = f(A_r) + f(C) + f(B_r)$ $\quad = f(A \cap B)$



Cumulative Outstanding Reading

- Read chapter 1 from Fujishige's book.
- Read chapter 2 from Fujishige's book.
- Read chapter 3 from Fujishige's book.
- Read chapter 4 from Fujishige's book.

Announcements, Assignments, and Reminders

- Next homework is posted on canvas. Due Thursday 5/10, 11:59pm.
- As always, if you have any questions about anything, please ask then via our discussion board (`https://canvas.uw.edu/courses/1216339/discussion_topics`). Can meet at odd hours via zoom (send message on canvas to schedule time to chat).

Class Road Map - EE563

- L1(3/26): Motivation, Applications, & Basic Definitions,
- L2(3/28): Machine Learning Apps (diversity, complexity, parameter, learning target, surrogate).
- L3(4/2): Info theory exs, more apps, definitions, graph/combinatorial examples
- L4(4/4): Graph and Combinatorial Examples, Matrix Rank, Examples and Properties, visualizations
- L5(4/9): More Examples/Properties/ Other Submodular Defs., Independence,
- L6(4/11): Matroids, Matroid Examples, Matroid Rank, Partition/Laminar Matroids
- L7(4/16): Laminar Matroids, System of Distinct Reps, Transversals, Transversal Matroid, Matroid Representation, Dual Matroids
- L8(4/18): Dual Matroids, Other Matroid Properties, Combinatorial Geometries, Matroids and Greedy.
- L9(4/23): Polyhedra, Matroid Polytopes, Matroids $\rightarrow$ Polymatroids
- L10(4/29): Matroids $\rightarrow$ Polymatroids, Polymatroids, Polymatroids and Greedy,
- L11(4/30): Polymatroids, Polymatroids and Greedy
- L12(5/2): Polymatroids and Greedy, Extreme Points, Cardinality Constrained Maximization
- L13(5/7): Constrained Submodular Maximization
- L14(5/9):
- L15(5/14):
- L16(5/16):
- L17(5/21):
- L18(5/23):
- L–(5/28): Memorial Day (holiday)
- L19(5/30):
- L21(6/4): Final Presentations maximization.

Last day of instruction, June 1st. Finals Week: June 2-8, 2018.

Multiple Polytopes associated with arbitrary f

- Given an arbitrary submodular function $f : 2^V \to R$ (not necessarily a polymatroid function, so it need not be positive, monotone, etc.).
- If $f(\emptyset) \neq 0$, can set $f'(A) = f(A) - f(\emptyset)$ without destroying submodularity. This does not change any minima, (i.e., $\text{argmin}_A f(A) = \text{argmin}_{A'} f'(A)$) so we often assume all functions are normalized $f(\emptyset) = 0$.
- We can define several polytopes:

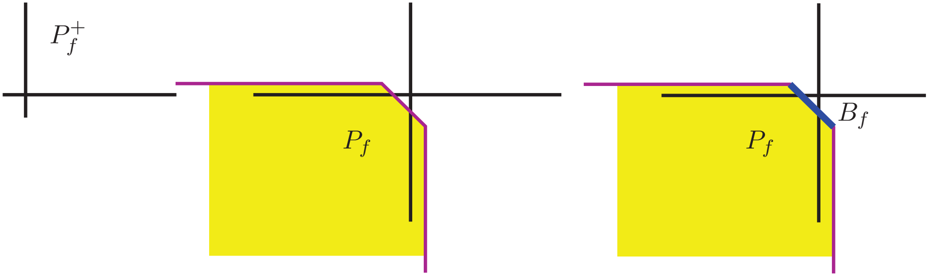
$$P_f = \left\{x \in \mathbb{R}^E : x(S) \leq f(S), \forall S \subseteq E\right\} \tag{13.1}$$

$$P_f^+ = P_f \cap \left\{x \in \mathbb{R}^E : x \geq 0\right\} \tag{13.2}$$

$$B_f = P_f \cap \left\{x \in \mathbb{R}^E : x(E) = f(E)\right\} \tag{13.3}$$

- P_f is what is sometimes called the extended polytope (sometimes notated as EP_f.
- P_f^+ is P_f restricted to the positive orthant.
- B_f is called the base polytope, analogous to the base in matroid.

Multiple Polytopes in 2D associated with f



$$P_f^+ = P_f \cap \left\{x \in \mathbb{R}^E : x \geq 0\right\} \tag{13.1}$$

$$P_f = \left\{x \in \mathbb{R}^E : x(S) \leq f(S), \forall S \subseteq E\right\} \tag{13.2}$$

$$B_f = P_f \cap \left\{x \in \mathbb{R}^E : x(E) = f(E)\right\} \tag{13.3}$$

A polymatroid function's polyhedron is a polymatroid.

Theorem 13.2.1

Let f be a submodular function defined on subsets of E. For any $x \in \mathbb{R}^E$, we have:

$$\textit{rank}(x) = \max\left(y(E) : y \leq x, y \in P_f\right) = \min\left(x(A) + f(E \setminus A) : A \subseteq E\right) \tag{13.1}$$

Essentially the same theorem as Theorem **??**, but note P_f rather than P_f^+. Taking $x = 0$ we get:

Corollary 13.2.2

Let f be a submodular function defined on subsets of E. We have:

$$\textit{rank}(0) = \max\left(y(E) : y \leq 0, y \in P_f\right) = \min\left(f(A) : A \subseteq E\right) \tag{13.2}$$

Polymatroid extreme points

Theorem 13.2.1

For a given ordering $E = (e_1, \dots, e_m)$ of E and a given $E_i = (e_1, \dots, e_i)$ and x generated by E_i using the greedy procedure ($x(e_i) = f(e_i|E_{i-1})$), then x is an extreme point of P_f when f is submodular.

Proof.

- We already saw that $x \in P_f$ (Theorem ??).
- To show that x is an extreme point of P_f, note that it is the unique solution of the following system of equations

$$x(E_j) = f(E_j) \text{ for } 1 \leq j \leq i \leq m \tag{13.4}$$

$$x(e) = 0 \text{ for } e \in E \setminus E_i \tag{13.5}$$

There are $i \leq m$ equations and $i \leq m$ unknowns, and simple Gaussian elimination gives us back the x constructed via the Greedy algorithm!!

Polymatroid extreme points

- Moreover, we have (and will ultimately prove)

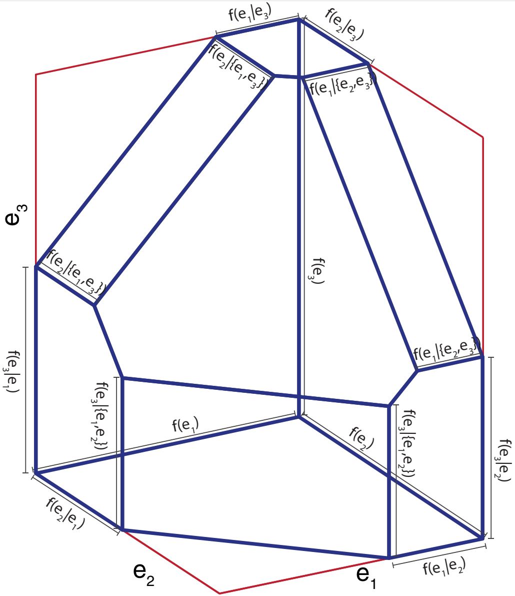
Corollary 13.2.2

If x is an extreme point of P_f and $B \subseteq E$ is given such that $\text{supp}(x) = \{e \in E : x(e) \neq 0\} \subseteq B \subseteq \cup(A : x(A) = f(A)) = \text{sat}(x)$, then x is generated using greedy by some ordering of B.

- Note, $\text{sat}(x) = \text{cl}(x) = \cup(A : x(A) = f(A))$ is also called the closure of x (recall that sets A such that $x(A) = f(A)$ are called tight, and such sets are closed under union and intersection, as seen in Lecture 10, Theorem ??)
- Thus, $\text{cl}(x)$ is a tight set.
- Also, $\text{supp}(x) = \{e \in E : x(e) \neq 0\}$ is called the support of x.
- For arbitrary x, $\text{supp}(x)$ is not necessarily tight, but for an extreme point, $\text{supp}(x)$ is.

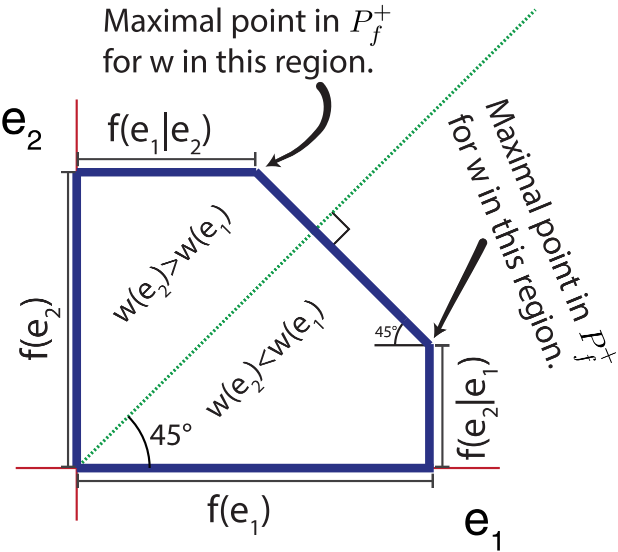
Polymatroid with labeled edge lengths

- Recall $f(e|A) = f(A+e) - f(A)$
- Notice how submodularity, $f(e|B) \leq f(e|A)$ for $A \subseteq B$, defines the shape of the polytope.
- In fact, we have strictness here $f(e|B) < f(e|A)$ for $A \subset B$.
- Also, consider how the greedy algorithm proceeds along the edges of the polytope.



Intuition: why greedy works with polymatroids

- Given w, the goal is to find $x = (x(e_1), x(e_2))$ that maximizes $x^\intercal w = x(e_1)w(e_1) + x(e_2)w(e_2)$.
- If $w(e_2) > w(e_1)$ the upper extreme point indicated maximizes $x^\intercal w$ over $x \in P_f^+$.
- If $w(e_2) < w(e_1)$ the lower extreme point indicated maximizes $x^\intercal w$ over $x \in P_f^+$.



The Greedy Algorithm for Submodular Max

A bit more precisely:

Algorithm 1: The Greedy Algorithm

Set $S_0 \leftarrow \emptyset$;
for $i \leftarrow 0 \ldots |E| - 1$ **do**
    Choose v_i as follows:
    $v_i \in \operatorname{argmax}_{v \in V \setminus S_i} f(\{v\}|S_i) = \operatorname{argmax}_{v \in V \setminus S_i} f(S_i \cup \{v\})$;
    Set $S_{i+1} \leftarrow S_i \cup \{v_i\}$;

Greedy Algorithm for Card. Constrained Submodular Max

- This algorithm has a guarantee

Theorem 13.2.1

Given a polymatroid function f , the above greedy algorithm returns sets S_i such that for each i we have $f(S_i) \geq (1 - 1/e) \max_{|S| \leq i} f(S)$.

- To approximately find $A^* \in \operatorname{argmax} \{f(A) : |A| \leq k\}$, we repeat the greedy step until $k = i + 1$:
- Again, since this generalizes max k-cover, Feige (1998) showed that this can't be improved. Unless $P = NP$, no polynomial time algorithm can do better than $(1 - 1/e + \epsilon)$ for any $\epsilon > 0$.

The Greedy Algorithm: $1 - 1/e$ intuition.

- At step $i < k$, greedy chooses v_i to maximize $f(v|S_i)$.

The Greedy Algorithm: $1 - 1/e$ intuition.

- At step $i < k$, greedy chooses v_i to maximize $f(v|S_i)$.
- Let S^* be optimal solution (of size k) and $\text{OPT} = f(S^*)$.

The Greedy Algorithm: $1 - 1/e$ intuition.

- At step $i < k$, greedy chooses v_i to maximize $f(v|S_i)$.
- Let S^* be optimal solution (of size k) and $\text{OPT} = f(S^*)$. By submodularity, we will show:

$$\exists v \in V \setminus S_i : f(v|S_i) = f(S_i + v|S_i) \geq \frac{1}{k}(\text{OPT} - f(S_i)) \tag{13.1}$$

The Greedy Algorithm: $1 - 1/e$ intuition.

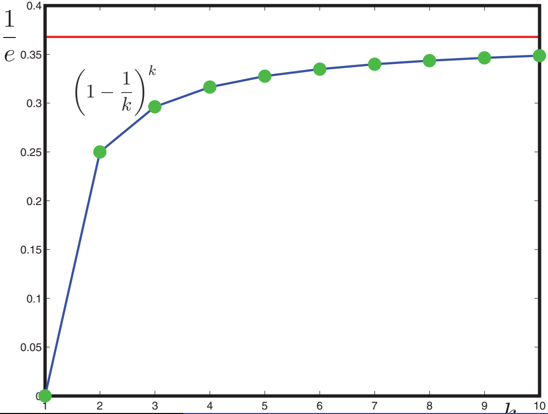
- At step $i < k$, greedy chooses v_i to maximize $f(v|S_i)$.
- Let S^* be optimal solution (of size k) and OPT $= f(S^*)$. By submodularity, we will show:

$$\exists v \in V \setminus S_i : f(v|S_i) = f(S_i + v|S_i) \geq \frac{1}{k}(\text{OPT} - f(S_i)) \tag{13.1}$$

The Greedy Algorithm: $1 - 1/e$ intuition.

- At step $i < k$, greedy chooses v_i to maximize $f(v|S_i)$.
- Let S^* be optimal solution (of size k) and $\mathrm{OPT} = f(S^*)$. By submodularity, we will show:

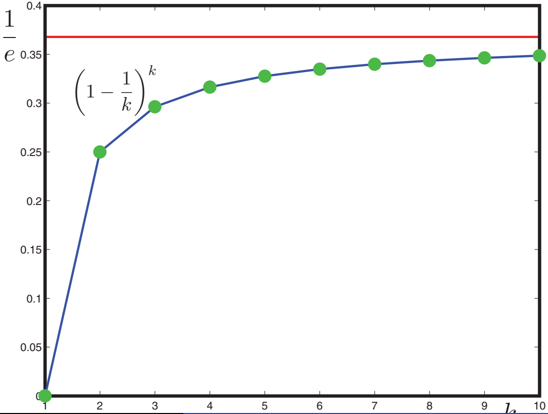
$$\exists v \in V \setminus S_i : f(v|S_i) = f(S_i + v|S_i) \geq \frac{1}{k}(\mathrm{OPT} - f(S_i)) \tag{13.1}$$



The Greedy Algorithm: $1 - 1/e$ intuition.

- At step $i < k$, greedy chooses v_i to maximize $f(v|S_i)$.
- Let S^* be optimal solution (of size k) and OPT $= f(S^*)$. By submodularity, we will show:

$$\exists v \in V \setminus S_i : f(v|S_i) = f(S_i + v|S_i) \geq \frac{1}{k}(\text{OPT} - f(S_i)) \tag{13.1}$$



Equation (13.10) will show that Equation (13.1) $\Rightarrow$:

$$\text{OPT} - f(S_{i+1}) \leq (1 - 1/k)(\text{OPT} - f(S_i))$$

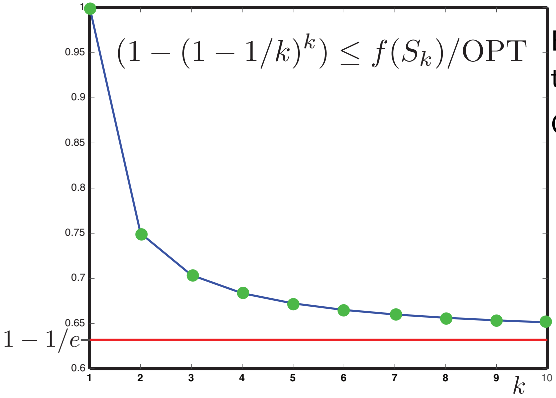
$$\Rightarrow \text{OPT} - f(S_k) \leq (1 - 1/k)^k \text{OPT} \leq 1/e\,\text{OPT}$$

$$\Rightarrow \text{OPT}(1 - 1/e) \leq f(S_k)$$

The Greedy Algorithm: $1 - 1/e$ intuition.

- At step $i < k$, greedy chooses v_i to maximize $f(v|S_i)$.
- Let S^* be optimal solution (of size k) and $\text{OPT} = f(S^*)$. By submodularity, we will show:

$$\exists v \in V \setminus S_i : f(v|S_i) = f(S_i + v|S_i) \geq \frac{1}{k}(\text{OPT} - f(S_i)) \tag{13.1}$$



Equation (13.10) will show that Equation (13.1) $\Rightarrow$:

$$\text{OPT} - f(S_{i+1}) \leq (1 - 1/k)(\text{OPT} - f(S_i))$$

$$\Rightarrow \text{OPT} - f(S_k) \leq (1 - 1/k)^k \text{OPT} \leq 1/e\text{OPT}$$

$$\Rightarrow \text{OPT}(1 - 1/e) \leq f(S_k)$$

Cardinality Constrained Polymatroid Max Theorem

Theorem 13.3.1 (Nemhauser et al. 1978)

Given non-negative monotone submodular function $f : 2^V \to \mathbb{R}_+$, define $\{S_i\}_{i \geq 0}$ to be the chain formed by the greedy algorithm (Eqn. (??)). Then for all $k, \ell \in \mathbb{Z}_{++}$, we have:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) \max_{S:|S| \leq k} f(S) \tag{13.2}$$

and in particular, for $\ell = k$, we have $f(S_k) \geq (1 - 1/e) \max_{S:|S| \leq k} f(S)$.

Cardinality Constrained Polymatroid Max Theorem

Theorem 13.3.1 (Nemhauser et al. 1978)

Given non-negative monotone submodular function $f : 2^V \to \mathbb{R}_+$, define $\{S_i\}_{i \geq 0}$ to be the chain formed by the greedy algorithm (Eqn. (??)). Then for all $k, \ell \in \mathbb{Z}_{++}$, we have:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) \max_{S:|S| \leq k} f(S) \tag{13.2}$$

and in particular, for $\ell = k$, we have $f(S_k) \geq (1 - 1/e) \max_{S:|S| \leq k} f(S)$.

- k is size of optimal set, i.e., OPT $= f(S^*)$ with $|S^*| = k$

Cardinality Constrained Polymatroid Max Theorem

Theorem 13.3.1 (Nemhauser et al. 1978)

Given non-negative monotone submodular function $f : 2^V \to \mathbb{R}_+$, define $\{S_i\}_{i \geq 0}$ to be the chain formed by the greedy algorithm (Eqn. (??)). Then for all $k, \ell \in \mathbb{Z}_{++}$, we have:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) \max_{S:|S| \leq k} f(S) \tag{13.2}$$

and in particular, for $\ell = k$, we have $f(S_k) \geq (1 - 1/e) \max_{S:|S| \leq k} f(S)$.

- k is size of optimal set, i.e., OPT $= f(S^*)$ with $|S^*| = k$
- ℓ is size of set we are choosing (i.e., we choose S_ℓ from greedy chain).

Cardinality Constrained Polymatroid Max Theorem

Theorem 13.3.1 (Nemhauser et al. 1978)

Given non-negative monotone submodular function $f : 2^V \to \mathbb{R}_+$, define $\{S_i\}_{i \geq 0}$ to be the chain formed by the greedy algorithm (Eqn. (??)). Then for all $k, \ell \in \mathbb{Z}_{++}$, we have:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) \max_{S:|S| \leq k} f(S) \tag{13.2}$$

and in particular, for $\ell = k$, we have $f(S_k) \geq (1 - 1/e) \max_{S:|S| \leq k} f(S)$.

- k is size of optimal set, i.e., OPT $= f(S^*)$ with $|S^*| = k$
- ℓ is size of set we are choosing (i.e., we choose S_ℓ from greedy chain).
- Bound is how well does S_ℓ (of size ℓ) do relative to S^*, the optimal set of size k.

Cardinality Constrained Polymatroid Max Theorem

Theorem 13.3.1 (Nemhauser et al. 1978)

Given non-negative monotone submodular function $f : 2^V \to \mathbb{R}_+$, define $\{S_i\}_{i \geq 0}$ to be the chain formed by the greedy algorithm (Eqn. (??)). Then for all $k, \ell \in \mathbb{Z}_{++}$, we have:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) \max_{S:|S| \leq k} f(S) \tag{13.2}$$

and in particular, for $\ell = k$, we have $f(S_k) \geq (1 - 1/e) \max_{S:|S| \leq k} f(S)$.

- k is size of optimal set, i.e., OPT $= f(S^*)$ with $|S^*| = k$
- ℓ is size of set we are choosing (i.e., we choose S_ℓ from greedy chain).
- Bound is how well does S_ℓ (of size ℓ) do relative to S^*, the optimal set of size k.
- Intuitively, bound should get worse when $\ell < k$ and get better when $\ell > k$.

Cardinality Constrained Polymatroid Max Theorem

Proof of Theorem 13.3.1.

...

Cardinality Constrained Polymatroid Max Theorem

Proof of Theorem 13.3.1.

- Fix ℓ (number of items greedy will chose) and k (size of optimal set to compare against).

...

Cardinality Constrained Polymatroid Max Theorem

Proof of Theorem 13.3.1.

- Fix ℓ (number of items greedy will chose) and k (size of optimal set to compare against).
- Set $S^* \in \operatorname{argmax} \{f(S) : |S| \leq k\}$

...

Cardinality Constrained Polymatroid Max Theorem

Proof of Theorem 13.3.1.

- Fix ℓ (number of items greedy will chose) and k (size of optimal set to compare against).
- Set $S^* \in \operatorname{argmax} \{f(S) : |S| \leq k\}$
- w.l.o.g. assume $|S^*| = k$.

...

Cardinality Constrained Polymatroid Max Theorem

Proof of Theorem 13.3.1.

- Fix ℓ (number of items greedy will chose) and k (size of optimal set to compare against).
- Set $S^* \in \operatorname{argmax}\{f(S) : |S| \leq k\}$
- w.l.o.g. assume $|S^*| = k$.
- Order $S^* = (v_1^*, v_2^*, \dots, v_k^*)$ arbitrarily.

...

Cardinality Constrained Polymatroid Max Theorem

Proof of Theorem 13.3.1.

- Fix ℓ (number of items greedy will chose) and k (size of optimal set to compare against).
- Set $S^* \in \operatorname{argmax} \{f(S) : |S| \leq k\}$
- w.l.o.g. assume $|S^*| = k$.
- Order $S^* = (v_1^*, v_2^*, \ldots, v_k^*)$ arbitrarily.
- Let $S_i = (v_1, v_2, \ldots, v_i)$ be the greedy order chain chosen by the algorithm, for $i \in \{1, 2, \ldots, \ell\}$.

...

Cardinality Constrained Polymatroid Max Theorem

Proof of Theorem 13.3.1.

- Fix ℓ (number of items greedy will chose) and k (size of optimal set to compare against).
- Set $S^* \in \operatorname{argmax}\{f(S) : |S| \leq k\}$
- w.l.o.g. assume $|S^*| = k$.
- Order $S^* = (v_1^*, v_2^*, \ldots, v_k^*)$ arbitrarily.
- Let $S_i = (v_1, v_2, \ldots, v_i)$ be the greedy order chain chosen by the algorithm, for $i \in \{1, 2, \ldots, \ell\}$.
- Then the following inequalities (on the next slide) follow:

...

Cardinality Constrained Polymatroid Max Theorem

... proof of Theorem 13.3.1 cont.

...

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- For all $i < \ell$, we have

$$f(S^*)$$

. . .

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- For all $i < \ell$, we have

$$f(S^*) \leq f(S^* \cup S_i)$$

. . .

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- For all $i < \ell$, we have

$$f(S^*) \leq f(S^* \cup S_i) = f(S_i) + f(S^*|S_i) \tag{13.3}$$

. . .

Cardinality Constrained Polymatroid Max Theorem

...proof of Theorem 13.3.1 cont.

- For all $i < \ell$, we have

$$f(S^*) \leq f(S^* \cup S_i) = f(S_i) + f(S^*|S_i) \tag{13.3}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^*|S_i \cup \{v_1^*, v_2^*, \ldots, v_{j-1}^*\}) \tag{13.4}$$

...

Cardinality Constrained Polymatroid Max Theorem

... proof of Theorem 13.3.1 cont.

- For all $i < \ell$, we have

$$f(S^*) \leq f(S^* \cup S_i) = f(S_i) + f(S^*|S_i) \tag{13.3}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^*|S_i \cup \{v_1^*, v_2^*, \ldots, v_{j-1}^*\}) \tag{13.4}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v|S_i) \tag{13.5}$$

...

Cardinality Constrained Polymatroid Max Theorem

... proof of Theorem 13.3.1 cont.

- For all $i < \ell$, we have

$$f(S^*) \leq f(S^* \cup S_i) = f(S_i) + f(S^*|S_i) \tag{13.3}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^*|S_i \cup \{v_1^*, v_2^*, \dots, v_{j-1}^*\}) \tag{13.4}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v|S_i) \tag{13.5}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v_{i+1}|S_i)$$

...

Cardinality Constrained Polymatroid Max Theorem

... proof of Theorem 13.3.1 cont.

- For all $i < \ell$, we have

$$f(S^*) \leq f(S^* \cup S_i) = f(S_i) + f(S^*|S_i) \tag{13.3}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^*|S_i \cup \{v_1^*, v_2^*, \dots, v_{j-1}^*\}) \tag{13.4}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v|S_i) \tag{13.5}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v_{i+1}|S_i) = f(S_i) + \sum_{v \in S^*} f(S_{i+1}|S_i) \tag{13.6}$$

...

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- For all $i < \ell$, we have

$$f(S^*) \leq f(S^* \cup S_i) = f(S_i) + f(S^*|S_i) \tag{13.3}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^*|S_i \cup \{v_1^*, v_2^*, \ldots, v_{j-1}^*\}) \tag{13.4}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v|S_i) \tag{13.5}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v_{i+1}|S_i) = f(S_i) + \sum_{v \in S^*} f(S_{i+1}|S_i) \tag{13.6}$$

$$= f(S_i) + kf(S_{i+1}|S_i) \tag{13.7}$$

. . .

Cardinality Constrained Polymatroid Max Theorem

... proof of Theorem 13.3.1 cont.

- For all $i < \ell$, we have

$$f(S^*) \leq f(S^* \cup S_i) = f(S_i) + f(S^*|S_i) \tag{13.3}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^*|S_i \cup \{v_1^*, v_2^*, \ldots, v_{j-1}^*\}) \tag{13.4}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v|S_i) \tag{13.5}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v_{i+1}|S_i) = f(S_i) + \sum_{v \in S^*} f(S_{i+1}|S_i) \tag{13.6}$$

$$= f(S_i) + k f(S_{i+1}|S_i) \tag{13.7}$$

- Therefore, we have Equation 13.1, i.e.,:

$$f(S^*) - f(S_i) \leq k f(S_{i+1}|S_i) = k(f(S_{i+1}) - f(S_i)) \tag{13.8}$$

...

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- Define gap $\delta_i \triangleq f(S^*) - f(S_i)$, so $\delta_i - \delta_{i+1} = f(S_{i+1}) - f(S_i)$,

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- Define gap $\delta_i \triangleq f(S^*) - f(S_i)$, so $\delta_i - \delta_{i+1} = f(S_{i+1}) - f(S_i)$, giving
$$\delta_i \leq k(\delta_i - \delta_{i+1}) \tag{13.9}$$
or

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- Define gap $\delta_i \triangleq f(S^*) - f(S_i)$, so $\delta_i - \delta_{i+1} = f(S_{i+1}) - f(S_i)$, giving

$$\delta_i \leq k(\delta_i - \delta_{i+1}) \tag{13.9}$$

or

$$\delta_{i+1} \leq (1 - \frac{1}{k})\delta_i \tag{13.10}$$

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- Define gap $\delta_i \triangleq f(S^*) - f(S_i)$, so $\delta_i - \delta_{i+1} = f(S_{i+1}) - f(S_i)$, giving

$$\delta_i \leq k(\delta_i - \delta_{i+1}) \tag{13.9}$$

or

$$\delta_{i+1} \leq (1 - \frac{1}{k})\delta_i \tag{13.10}$$

- The relationship between δ_0 and δ_ℓ is then

$$\delta_l \leq (1 - \frac{1}{k})^\ell \delta_0 \tag{13.11}$$

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- Define gap $\delta_i \triangleq f(S^*) - f(S_i)$, so $\delta_i - \delta_{i+1} = f(S_{i+1}) - f(S_i)$, giving

$$\delta_i \leq k(\delta_i - \delta_{i+1}) \tag{13.9}$$

or

$$\delta_{i+1} \leq (1 - \frac{1}{k})\delta_i \tag{13.10}$$

- The relationship between δ_0 and δ_ℓ is then

$$\delta_l \leq (1 - \frac{1}{k})^\ell \delta_0 \tag{13.11}$$

- Now, $\delta_0 = f(S^*) - f(\emptyset) \leq f(S^*)$ since $f \geq 0$.

Cardinality Constrained Polymatroid Max Theorem

...proof of Theorem 13.3.1 cont.

- Define gap $\delta_i \triangleq f(S^*) - f(S_i)$, so $\delta_i - \delta_{i+1} = f(S_{i+1}) - f(S_i)$, giving
$$\delta_i \leq k(\delta_i - \delta_{i+1}) \tag{13.9}$$
or
$$\delta_{i+1} \leq (1 - \frac{1}{k})\delta_i \tag{13.10}$$
- The relationship between δ_0 and δ_ℓ is then
$$\delta_l \leq (1 - \frac{1}{k})^\ell \delta_0 \tag{13.11}$$
- Now, $\delta_0 = f(S^*) - f(\emptyset) \leq f(S^*)$ since $f \geq 0$.
- Also, by variational bound $1 - x \leq e^{-x}$ for $x \in \mathbb{R}$, we have
$$\delta_\ell \leq (1 - \frac{1}{k})^\ell \delta_0 \leq e^{-\ell/k} f(S^*) \tag{13.12}$$

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

$\square$

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- When we identify $\delta_l = f(S^*) - f(S_\ell)$, a bit of rearranging then gives:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \tag{13.13}$$

$\square$

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- When we identify $\delta_l = f(S^*) - f(S_\ell)$, a bit of rearranging then gives:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \tag{13.13}$$

□

- With $\ell = k$, when picking k items, greedy gets $(1 - 1/e) \approx 0.6321$ bound. This means that if S_k is greedy solution of size k, and S^* is an optimal solution of size k, $f(S_k) \geq (1 - 1/e) f(S^*) \approx 0.6321 f(S^*)$.

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- When we identify $\delta_l = f(S^*) - f(S_\ell)$, a bit of rearranging then gives:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \tag{13.13}$$

$\square$

- With $\ell = k$, when picking k items, greedy gets $(1 - 1/e) \approx 0.6321$ bound. This means that if S_k is greedy solution of size k, and S^* is an optimal solution of size k, $f(S_k) \geq (1 - 1/e) f(S^*) \approx 0.6321 f(S^*)$.
- What if we want to guarantee a solution no worse than $.95 f(S^*)$ where $|S^*| = k$?

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- When we identify $\delta_l = f(S^*) - f(S_\ell)$, a bit of rearranging then gives:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \tag{13.13}$$

$\square$

- With $\ell = k$, when picking k items, greedy gets $(1 - 1/e) \approx 0.6321$ bound. This means that if S_k is greedy solution of size k, and S^* is an optimal solution of size k, $f(S_k) \geq (1 - 1/e) f(S^*) \approx 0.6321 f(S^*)$.
- What if we want to guarantee a solution no worse than $.95 f(S^*)$ where $|S^*| = k$? Set $0.95 = (1 - e^{-\ell/k})$, which gives $\ell = \lceil -k \ln(1 - 0.95) \rceil = 4k$.

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- When we identify $\delta_l = f(S^*) - f(S_\ell)$, a bit of rearranging then gives:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \tag{13.13}$$

$\square$

- With $\ell = k$, when picking k items, greedy gets $(1 - 1/e) \approx 0.6321$ bound. This means that if S_k is greedy solution of size k, and S^* is an optimal solution of size k, $f(S_k) \geq (1 - 1/e) f(S^*) \approx 0.6321 f(S^*)$.
- What if we want to guarantee a solution no worse than $.95 f(S^*)$ where $|S^*| = k$? Set $0.95 = (1 - e^{-\ell/k})$, which gives $\ell = \lceil -k \ln(1 - 0.95) \rceil = 4k$. And $\lceil -\ln(1 - 0.999) \rceil = 7$.

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 13.3.1 cont.

- When we identify $\delta_l = f(S^*) - f(S_\ell)$, a bit of rearranging then gives:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \tag{13.13}$$

$\square$

- With $\ell = k$, when picking k items, greedy gets $(1 - 1/e) \approx 0.6321$ bound. This means that if S_k is greedy solution of size k, and S^* is an optimal solution of size k, $f(S_k) \geq (1 - 1/e) f(S^*) \approx 0.6321 f(S^*)$.
- What if we want to guarantee a solution no worse than $.95 f(S^*)$ where $|S^*| = k$? Set $0.95 = (1 - e^{-\ell/k})$, which gives $\ell = \lceil -k \ln(1 - 0.95) \rceil = 4k$. And $\lceil -\ln(1 - 0.999) \rceil = 7$.
- So solution, in the worst case, quickly gets very good. Typical/practical case is much better.

Greedy running time

- Greedy computes a new maximum $n = |V|$ times, and each maximum computation requires $O(n)$ comparisons, leading to $O(n^2)$ computation for greedy.

Greedy running time

- Greedy computes a new maximum $n = |V|$ times, and each maximum computation requires $O(n)$ comparisons, leading to $O(n^2)$ computation for greedy.
- This is the best we can do for arbitrary functions, but $O(n^2)$ is not practical to some.

Greedy running time

- Greedy computes a new maximum $n = |V|$ times, and each maximum computation requires $O(n)$ comparisons, leading to $O(n^2)$ computation for greedy.
- This is the best we can do for arbitrary functions, but $O(n^2)$ is not practical to some.
- Greedy can be made much faster in practice by a simple strategy made possible, once again, via the use of submodularity.

Greedy running time

- Greedy computes a new maximum $n = |V|$ times, and each maximum computation requires $O(n)$ comparisons, leading to $O(n^2)$ computation for greedy.
- This is the best we can do for arbitrary functions, but $O(n^2)$ is not practical to some.
- Greedy can be made much faster in practice by a simple strategy made possible, once again, via the use of submodularity.
- This is called Minoux's 1977 Accelerated Greedy strategy (and has been rediscovered a few times, e.g., "Lazy greedy"), and runs much faster while still producing same answer.

Greedy running time

- Greedy computes a new maximum $n = |V|$ times, and each maximum computation requires $O(n)$ comparisons, leading to $O(n^2)$ computation for greedy.
- This is the best we can do for arbitrary functions, but $O(n^2)$ is not practical to some.
- Greedy can be made much faster in practice by a simple strategy made possible, once again, via the use of submodularity.
- This is called Minoux's 1977 Accelerated Greedy strategy (and has been rediscovered a few times, e.g., "Lazy greedy"), and runs much faster while still producing same answer.
- We describe it next:

Minoux's Accelerated Greedy for Submodular Functions

- At stage i in the algorithm, we have a set of gains $f(v|S_i)$ for all $v \notin S_i$. Store these values $\alpha_v \leftarrow f(v|S_i)$ in sorted priority queue.

Minoux's Accelerated Greedy for Submodular Functions

- At stage i in the algorithm, we have a set of gains $f(v|S_i)$ for all $v \notin S_i$. Store these values $\alpha_v \leftarrow f(v|S_i)$ in sorted priority queue.
- Priority queue, $O(1)$ to find max, $O(\log n)$ to insert in right place.

Minoux's Accelerated Greedy for Submodular Functions

- At stage i in the algorithm, we have a set of gains $f(v|S_i)$ for all $v \notin S_i$. Store these values $\alpha_v \leftarrow f(v|S_i)$ in sorted priority queue.
- Priority queue, $O(1)$ to find max, $O(\log n)$ to insert in right place.
- Once we choose a max v, then set $S_{i+1} \leftarrow S_i + v$.

Minoux's Accelerated Greedy for Submodular Functions

- At stage i in the algorithm, we have a set of gains $f(v|S_i)$ for all $v \notin S_i$. Store these values $\alpha_v \leftarrow f(v|S_i)$ in sorted priority queue.
- Priority queue, $O(1)$ to find max, $O(\log n)$ to insert in right place.
- Once we choose a max v, then set $S_{i+1} \leftarrow S_i + v$.
- For $v \notin S_{i+1}$ we have $f(v|S_{i+1}) \leq f(v|S_i)$ by submodularity.

Minoux's Accelerated Greedy for Submodular Functions

- At stage i in the algorithm, we have a set of gains $f(v|S_i)$ for all $v \notin S_i$. Store these values $\alpha_v \leftarrow f(v|S_i)$ in sorted priority queue.
- Priority queue, $O(1)$ to find max, $O(\log n)$ to insert in right place.
- Once we choose a max v, then set $S_{i+1} \leftarrow S_i + v$.
- For $v \notin S_{i+1}$ we have $f(v|S_{i+1}) \leq f(v|S_i)$ by submodularity.
- Therefore, if we find a v' such that $f(v'|S_{i+1}) \geq \alpha_v$ for all $v \neq v'$, then since

$$f(v'|S_{i+1}) \geq \alpha_v = f(v|S_i) \geq f(v|S_{i+1}) \tag{13.14}$$

  we have the true max, and we need not re-evaluate gains of other elements again.

Minoux's Accelerated Greedy for Submodular Functions

- At stage i in the algorithm, we have a set of gains $f(v|S_i)$ for all $v \notin S_i$. Store these values $\alpha_v \leftarrow f(v|S_i)$ in sorted priority queue.
- Priority queue, $O(1)$ to find max, $O(\log n)$ to insert in right place.
- Once we choose a max v, then set $S_{i+1} \leftarrow S_i + v$.
- For $v \notin S_{i+1}$ we have $f(v|S_{i+1}) \leq f(v|S_i)$ by submodularity.
- Therefore, if we find a v' such that $f(v'|S_{i+1}) \geq \alpha_v$ for all $v \neq v'$, then since

$$f(v'|S_{i+1}) \geq \alpha_v = f(v|S_i) \geq f(v|S_{i+1}) \tag{13.14}$$

  we have the true max, and we need not re-evaluate gains of other elements again.
- Strategy is: find the $\operatorname{argmax}_{v' \in V \setminus S_{i+1}} \alpha_{v'}$, and then compute the real $f(v'|S_{i+1})$. If it is greater than all other α_v's then that's the next greedy step. Otherwise, replace $\alpha_{v'}$ with its real value, resort $(O(\log n))$, and repeat.

Minoux's Accelerated Greedy for Submodular Functions

- Minoux's algorithm is exact, in that it has the same guarantees as does the standard $O(n^2)$ greedy algorithm (will return the same answers, i.e., those having the $1 - 1/e$ guarantee).

Minoux's Accelerated Greedy for Submodular Functions

- Minoux's algorithm is exact, in that it has the same guarantees as does the standard $O(n^2)$ greedy algorithm (will return the same answers, i.e., those having the $1 - 1/e$ guarantee).
- In practice: Minoux's trick has enormous speedups ($\approx 700\times$) over the standard greedy procedure due to reduced function evaluations and use of good data structures (priority queue).

Minoux's Accelerated Greedy for Submodular Functions

- Minoux's algorithm is exact, in that it has the same guarantees as does the standard $O(n^2)$ greedy algorithm (will return the same answers, i.e., those having the $1 - 1/e$ guarantee).
- In practice: Minoux's trick has enormous speedups ($\approx 700\times$) over the standard greedy procedure due to reduced function evaluations and use of good data structures (priority queue).
- When choosing a of size k, naïve greedy algorithm is $O(nk)$ but accelerated variant at the very best does $O(n+k)$, so this limits the speedup.

Minoux's Accelerated Greedy for Submodular Functions

- Minoux's algorithm is exact, in that it has the same guarantees as does the standard $O(n^2)$ greedy algorithm (will return the same answers, i.e., those having the $1 - 1/e$ guarantee).
- In practice: Minoux's trick has enormous speedups ($\approx 700\times$) over the standard greedy procedure due to reduced function evaluations and use of good data structures (priority queue).
- When choosing a of size k, naïve greedy algorithm is $O(nk)$ but accelerated variant at the very best does $O(n + k)$, so this limits the speedup.
- Algorithm has been rediscovered (I think) independently (CELF - cost-effective lazy forward selection, Leskovec et al., 2007)

Minoux's Accelerated Greedy for Submodular Functions

- Minoux's algorithm is exact, in that it has the same guarantees as does the standard $O(n^2)$ greedy algorithm (will return the same answers, i.e., those having the $1 - 1/e$ guarantee).
- In practice: Minoux's trick has enormous speedups ($\approx 700\times$) over the standard greedy procedure due to reduced function evaluations and use of good data structures (priority queue).
- When choosing a of size k, naïve greedy algorithm is $O(nk)$ but accelerated variant at the very best does $O(n + k)$, so this limits the speedup.
- Algorithm has been rediscovered (I think) independently (CELF - cost-effective lazy forward selection, Leskovec et al., 2007)
- Can be used used for "big data" sets (e.g., social networks, selecting blogs of greatest influence, document summarization, etc.).

Priority Queue

- Use a priority queue Q as a data structure: operations include:

Priority Queue

- Use a priority queue Q as a data structure: operations include:
  - Insert an item (v, α) into queue, with $v \in V$ and $\alpha \in \mathbb{R}$.

$$\text{insert}(Q, (v, \alpha)) \tag{13.15}$$

Priority Queue

- Use a priority queue Q as a data structure: operations include:
    - Insert an item (v, α) into queue, with $v \in V$ and $\alpha \in \mathbb{R}$.

$$\text{insert}(Q, (v, \alpha)) \tag{13.15}$$

    - Pop the item (v, α) with maximum value α off the queue.

$$(v, \alpha) \leftarrow \text{pop}(Q) \tag{13.16}$$

Priority Queue

- Use a priority queue Q as a data structure: operations include:
  - Insert an item (v, α) into queue, with $v \in V$ and $\alpha \in \mathbb{R}$.

$$\text{insert}(Q, (v, \alpha)) \tag{13.15}$$

  - Pop the item (v, α) with maximum value α off the queue.

$$(v, \alpha) \leftarrow \text{pop}(Q) \tag{13.16}$$

  - Query the value of the max item in the queue

$$\max(Q) \in \mathbb{R} \tag{13.17}$$

Priority Queue

- Use a priority queue Q as a data structure: operations include:
    - Insert an item (v, α) into queue, with $v \in V$ and $\alpha \in \mathbb{R}$.

$$\text{insert}(Q, (v, \alpha)) \tag{13.15}$$

    - Pop the item (v, α) with maximum value α off the queue.

$$(v, \alpha) \leftarrow \text{pop}(Q) \tag{13.16}$$

    - Query the value of the max item in the queue

$$\max(Q) \in \mathbb{R} \tag{13.17}$$

- On next slide, we call a popped item "fresh" if the value (v, α) popped has the correct value $\alpha = f(v|S_i)$. Use extra "bit" to store this info

Priority Queue

- Use a priority queue Q as a data structure: operations include:
  - Insert an item (v, α) into queue, with $v \in V$ and $\alpha \in \mathbb{R}$.

$$\text{insert}(Q, (v, \alpha)) \tag{13.15}$$

  - Pop the item (v, α) with maximum value α off the queue.

$$(v, \alpha) \leftarrow \text{pop}(Q) \tag{13.16}$$

  - Query the value of the max item in the queue

$$\max(Q) \in \mathbb{R} \tag{13.17}$$

- On next slide, we call a popped item "fresh" if the value (v, α) popped has the correct value $\alpha = f(v|S_i)$. Use extra "bit" to store this info
- If a popped item is fresh, it must be the maximum — this can happen if, at given iteration, v was first popped and neither fresh nor maximum so placed back in the queue, and it then percolates back to the top at which point it is fresh — thereby avoid extra queue check.

Minoux's Accelerated Greedy Algorithm Submodular Max

Algorithm 2: Minoux's Accelerated Greedy Algorithm

```
1  Set S_0 ← ∅ ; i ← 0 ; Initialize priority queue Q ;
2  for v ∈ E do
3      INSERT(Q, f(v))
4  repeat
5      (v, α) ← pop(Q) ;
6      if α not "fresh" then
7          recompute α ← f(v|S_i)
8      if (popped α in line 5 was "fresh") OR (α ≥ max(Q)) then
9          Set S_{i+1} ← S_i ∪ {v} ;
10         i ← i + 1 ;
11     else
12         insert(Q, (v, α))
13 until i = |E|;
```

(Minimum) Submodular Set Cover

- Given polymatroid f, goal is to find a covering set of minimum cost:

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \tag{13.18}$$

where α is a "cover" requirement.

(Minimum) Submodular Set Cover

- Given polymatroid f, goal is to find a covering set of minimum cost:

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \tag{13.18}$$

  where α is a "cover" requirement.

- Normally take $\alpha = f(V)$ but defining $f'(A) = \min\{f(A), \alpha\}$ we can take any α. Hence, we have equivalent formulation:

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f'(S) \geq f'(V) \tag{13.19}$$

(Minimum) Submodular Set Cover

- Given polymatroid f, goal is to find a covering set of minimum cost:

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \tag{13.18}$$

where α is a "cover" requirement.

- Normally take $\alpha = f(V)$ but defining $f'(A) = \min\{f(A), \alpha\}$ we can take any α. Hence, we have equivalent formulation:

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f'(S) \geq f'(V) \tag{13.19}$$

- Note that this immediately generalizes standard set cover, in which case $f(A)$ is the cardinality of the union of sets indexed by A.

(Minimum) Submodular Set Cover

- Given polymatroid f, goal is to find a covering set of minimum cost:

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \tag{13.18}$$

  where α is a "cover" requirement.
- Normally take $\alpha = f(V)$ but defining $f'(A) = \min\{f(A), \alpha\}$ we can take any α. Hence, we have equivalent formulation:

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f'(S) \geq f'(V) \tag{13.19}$$

- Note that this immediately generalizes standard set cover, in which case $f(A)$ is the cardinality of the union of sets indexed by A.
- Greedy Algorithm: Pick the first chain item S_i chosen by aforementioned greedy algorithm such that $f(S_i) \geq \alpha$ and output that as solution.

(Minimum) Submodular Set Cover: Approximation Analysis

- For integer valued f, this greedy algorithm an $O(\log(\max_{s \in V} f(\{s\})))$ approximation. Let S^* be optimal, and S^{G} be greedy solution, then

$$|S^{\mathsf{G}}| \leq |S^*| H(\max_{s \in V} f(\{s\})) = |S^*| O(\log_e(\max_{s \in V} f(\{s\}))) \tag{13.20}$$

where H is the harmonic function, i.e., $H(d) = \sum_{i=1}^{d} (1/i)$.

(Minimum) Submodular Set Cover: Approximation Analysis

- For integer valued f, this greedy algorithm an $O(\log(\max_{s \in V} f(\{s\})))$ approximation. Let S^* be optimal, and S^{G} be greedy solution, then

$$|S^{\mathsf{G}}| \leq |S^*| H(\max_{s \in V} f(\{s\})) = |S^*| O(\log_e(\max_{s \in V} f(\{s\}))) \tag{13.20}$$

  where H is the harmonic function, i.e., $H(d) = \sum_{i=1}^{d} (1/i)$.

- If f is not integral value, then bounds we get are of the form:

$$|S^{\mathsf{G}}| \leq |S^*| \left(1 + \log_e \frac{f(V)}{f(V) - f(S_{T-1})}\right) \tag{13.21}$$

  wehre S_T is the final greedy solution that occurs at step T.

(Minimum) Submodular Set Cover: Approximation Analysis

- For integer valued f, this greedy algorithm an $O(\log(\max_{s \in V} f(\{s\})))$ approximation. Let S^* be optimal, and S^{G} be greedy solution, then

$$|S^{\mathsf{G}}| \leq |S^*| H(\max_{s \in V} f(\{s\})) = |S^*| O(\log_e(\max_{s \in V} f(\{s\}))) \tag{13.20}$$

  where H is the harmonic function, i.e., $H(d) = \sum_{i=1}^{d} (1/i)$.

- If f is not integral value, then bounds we get are of the form:

$$|S^{\mathsf{G}}| \leq |S^*| \left(1 + \log_e \frac{f(V)}{f(V) - f(S_{T-1})}\right) \tag{13.21}$$

  wehre S_T is the final greedy solution that occurs at step T.

- Set cover is hard to approximate with a factor better than $(1-\epsilon)\log \alpha$, where α is the desired cover constraint.

Summary: Monotone Submodular Maximization

- Only makes sense when there is a constraint.

Summary: Monotone Submodular Maximization

- Only makes sense when there is a constraint.
- We discussed cardinality constraint

Summary: Monotone Submodular Maximization

- Only makes sense when there is a constraint.
- We discussed cardinality constraint
- Generalizes the max k-cover problem, and also similar to the set cover problem.

Summary: Monotone Submodular Maximization

- Only makes sense when there is a constraint.
- We discussed cardinality constraint
- Generalizes the max k-cover problem, and also similar to the set cover problem.
- Simple greedy algorithm gets $1 - e^{-\ell/k}$ approximation, where k is size of optimal set we compare against, and ℓ is size of set greedy algorithm chooses.

Summary: Monotone Submodular Maximization

- Only makes sense when there is a constraint.
- We discussed cardinality constraint
- Generalizes the max k-cover problem, and also similar to the set cover problem.
- Simple greedy algorithm gets $1 - e^{-\ell/k}$ approximation, where k is size of optimal set we compare against, and ℓ is size of set greedy algorithm chooses.
- Submodular cover: min. $|S|$ s.t. $f(S) \geq \alpha$.

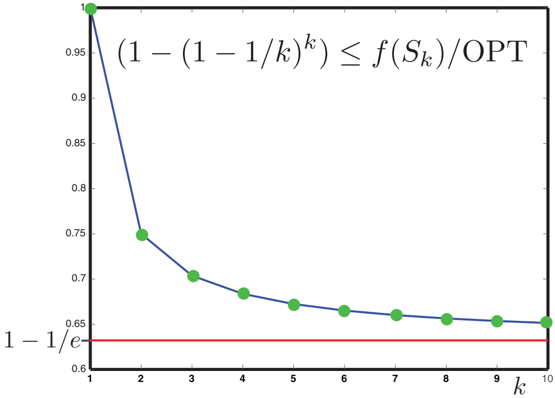
Summary: Monotone Submodular Maximization

- Only makes sense when there is a constraint.
- We discussed cardinality constraint
- Generalizes the max k-cover problem, and also similar to the set cover problem.
- Simple greedy algorithm gets $1 - e^{-\ell/k}$ approximation, where k is size of optimal set we compare against, and ℓ is size of set greedy algorithm chooses.
- Submodular cover: min. $|S|$ s.t. $f(S) \geq \alpha$.
- Minoux's accelerated greedy trick.

The Greedy Algorithm: $1 - 1/e$ intuition.

- At step $i < k$, greedy chooses v_i to maximize $f(v|S_i)$.
- Let S^* be optimal solution (of size k) and OPT $= f(S^*)$. By submodularity, we will show:

$$\exists v \in V \setminus S_i : f(v|S_i) = f(S_i + v|S_i) \geq \frac{1}{k}(\text{OPT} - f(S_i)) \tag{13.1}$$



Equation (13.10) will show that Equation (13.1) $\Rightarrow$:

$$\begin{aligned}
&\text{OPT} - f(S_{i+1}) \\
&\qquad \leq (1 - 1/k)(\text{OPT} - f(S_i)) \\
&\Rightarrow \text{OPT} - f(S_k) \\
&\qquad \leq (1 - 1/k)^k \text{OPT} \\
&\qquad \leq 1/e\text{OPT} \\
&\Rightarrow \text{OPT}(1 - 1/e) \leq f(S_k)
\end{aligned}$$

Randomized greedy

- How can we produce a randomized greedy strategy, one where each greedy sweep produces a set that, on average, has a $1 - 1/e$ guarantee?

Randomized greedy

- How can we produce a randomized greedy strategy, one where each greedy sweep produces a set that, on average, has a $1 - 1/e$ guarantee?
- Suppose the following holds:

$$E[f(a_{i+1}|A_i)] \geq \frac{f(OPT) - f(A_i)}{k} \tag{13.22}$$

where $A_i = (a_1, a_2, \ldots, a_i)$ are the first i elements chosen by the strategy.

Curvature of a Submodular function

- For any submodular function, we have $f(j|S) \leq f(j|\emptyset)$ so that $f(j|S)/f(j|\emptyset) \leq 1$ whenever $f(j|\emptyset) \neq 0$.

Curvature of a Submodular function

- For any submodular function, we have $f(j|S) \leq f(j|\emptyset)$ so that $f(j|S)/f(j|\emptyset) \leq 1$ whenever $f(j|\emptyset) \neq 0$.
- For $f : 2^V \to \mathbb{R}_+$ (non-negative) functions, we also have $f(j|S)/f(j|\emptyset) \geq 0$ — and $= 0$ whenever j is "spanned" by S.

Curvature of a Submodular function

- For any submodular function, we have $f(j|S) \leq f(j|\emptyset)$ so that $f(j|S)/f(j|\emptyset) \leq 1$ whenever $f(j|\emptyset) \neq 0$.
- For $f : 2^V \to \mathbb{R}_+$ (non-negative) functions, we also have $f(j|S)/f(j|\emptyset) \geq 0$ — and $= 0$ whenever j is "spanned" by S.
- The total curvature of a submodular function is defined as follows:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{f(j) \neq 0} \frac{f(j|V \setminus j)}{f(j)} \qquad (13.23)$$

Curvature of a Submodular function

- For any submodular function, we have $f(j|S) \leq f(j|\emptyset)$ so that $f(j|S)/f(j|\emptyset) \leq 1$ whenever $f(j|\emptyset) \neq 0$.
- For $f : 2^V \to \mathbb{R}_+$ (non-negative) functions, we also have $f(j|S)/f(j|\emptyset) \geq 0$ — and $= 0$ whenever j is "spanned" by S.
- The total curvature of a submodular function is defined as follows:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{f(j) \neq 0} \frac{f(j|V \setminus j)}{f(j)} \tag{13.23}$$

- $c \in [0, 1]$.

Curvature of a Submodular function

- For any submodular function, we have $f(j|S) \leq f(j|\emptyset)$ so that $f(j|S)/f(j|\emptyset) \leq 1$ whenever $f(j|\emptyset) \neq 0$.
- For $f : 2^V \to \mathbb{R}_+$ (non-negative) functions, we also have $f(j|S)/f(j|\emptyset) \geq 0$ — and $= 0$ whenever j is "spanned" by S.
- The total curvature of a submodular function is defined as follows:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{f(j) \neq 0} \frac{f(j|V \setminus j)}{f(j)} \tag{13.23}$$

- $c \in [0, 1]$. When $c = 0$, $f(j|S) = f(j|\emptyset)$ for all S, j, a sufficient condition for modularity, and we saw in Theorem ?? that greedy is optimal for max weight indep. set of a matroid.

Curvature of a Submodular function

- For any submodular function, we have $f(j|S) \leq f(j|\emptyset)$ so that $f(j|S)/f(j|\emptyset) \leq 1$ whenever $f(j|\emptyset) \neq 0$.
- For $f : 2^V \to \mathbb{R}_+$ (non-negative) functions, we also have $f(j|S)/f(j|\emptyset) \geq 0$ — and $= 0$ whenever j is "spanned" by S.
- The total curvature of a submodular function is defined as follows:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{f(j) \neq 0} \frac{f(j|V \setminus j)}{f(j)} \quad (13.23)$$

- $c \in [0, 1]$. When $c = 0$, $f(j|S) = f(j|\emptyset)$ for all S, j, a sufficient condition for modularity, and we saw in Theorem ?? that greedy is optimal for max weight indep. set of a matroid.
- For f with curvature c, then $\forall A \subseteq V$, $\forall v \notin a$, $\forall c' \geq c$:

$$f(A + v) - f(A) \geq (1 - c') f(v) \quad (13.24)$$

$$f(v) \geq f(v|A) = f(v) \frac{f(v|A)}{f(v)} \geq f(v) \min_{v'} \frac{f(v'|A)}{f(v')} = (1 - c) f(v) \geq (1 - c') f(v) \quad (13.25)$$

Curvature of a Submodular function

- For any submodular function, we have $f(j|S) \leq f(j|\emptyset)$ so that $f(j|S)/f(j|\emptyset) \leq 1$ whenever $f(j|\emptyset) \neq 0$.
- For $f : 2^V \to \mathbb{R}_+$ (non-negative) functions, we also have $f(j|S)/f(j|\emptyset) \geq 0$ — and $= 0$ whenever j is "spanned" by S.
- The total curvature of a submodular function is defined as follows:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{f(j) \neq 0} \frac{f(j|V \setminus j)}{f(j)} \tag{13.23}$$

- $c \in [0, 1]$. When $c = 0$, $f(j|S) = f(j|\emptyset)$ for all S, j, a sufficient condition for modularity, and we saw in Theorem ?? that greedy is optimal for max weight indep. set of a matroid.
- For f with curvature c, then $\forall A \subseteq V$, $\forall v \notin a$, $\forall c' \geq c$:

$$f(A + v) - f(A) \geq (1 - c') f(v) \tag{13.24}$$

- When $c = 1$ then submodular function is "maximally curved", i.e., there exists is a subset that fully spans some other element.

Curvature of a Submodular function

- For any submodular function, we have $f(j|S) \leq f(j|\emptyset)$ so that $f(j|S)/f(j|\emptyset) \leq 1$ whenever $f(j|\emptyset) \neq 0$.
- For $f : 2^V \to \mathbb{R}_+$ (non-negative) functions, we also have $f(j|S)/f(j|\emptyset) \geq 0$ — and $= 0$ whenever j is "spanned" by S.
- The total curvature of a submodular function is defined as follows:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{f(j) \neq 0} \frac{f(j|V \setminus j)}{f(j)} \tag{13.23}$$

- $c \in [0,1]$. When $c = 0$, $f(j|S) = f(j|\emptyset)$ for all S, j, a sufficient condition for modularity, and we saw in Theorem ?? that greedy is optimal for max weight indep. set of a matroid.
- For f with curvature c, then $\forall A \subseteq V$, $\forall v \notin a$, $\forall c' \geq c$:

$$f(A + v) - f(A) \geq (1 - c')f(v) \tag{13.24}$$

- When $c = 1$ then submodular function is "maximally curved", i.e., there exists is a subset that fully spans some other element.
- Matroid rank functions with some dependence is maximally curved.

Curvature of a Submodular function

- By submodularity, total curvature can be computed in either form:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{j : f(j|\emptyset) \neq 0} \frac{f(j|V \setminus \{j\})}{f(j|\emptyset)} \tag{13.25}$$

Curvature of a Submodular function

- By submodularity, total curvature can be computed in either form:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{j : f(j|\emptyset) \neq 0} \frac{f(j|V \setminus \{j\})}{f(j|\emptyset)} \tag{13.25}$$

- Note: Matroid rank is either modular $c = 0$ or maximally curved $c = 1$ — hence, matroid rank can have only the extreme points of curvature, namely 0 or 1.

Curvature of a Submodular function

- By submodularity, total curvature can be computed in either form:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{j : f(j|\emptyset) \neq 0} \frac{f(j|V \setminus \{j\})}{f(j|\emptyset)} \tag{13.25}$$

- Note: Matroid rank is either modular $c = 0$ or maximally curved $c = 1$ — hence, matroid rank can have only the extreme points of curvature, namely 0 or 1.
- Polymatroid functions are, in this sense, more nuanced, in that they allow non-extreme curvature, with $c \in [0, 1]$.

Curvature of a Submodular function

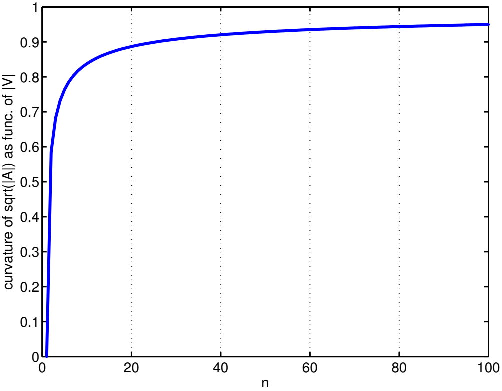
- By submodularity, total curvature can be computed in either form:

$$c \triangleq 1 - \min_{S, j \notin S: f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{j: f(j|\emptyset) \neq 0} \frac{f(j|V \setminus \{j\})}{f(j|\emptyset)} \quad (13.25)$$

- Note: Matroid rank is either modular $c = 0$ or maximally curved $c = 1$ — hence, matroid rank can have only the extreme points of curvature, namely 0 or 1.
- Polymatroid functions are, in this sense, more nuanced, in that they allow non-extreme curvature, with $c \in [0, 1]$.
- It will be remembered the notion of "partial dependence" within polymatroid functions.

Curvature for $f(S) = \sqrt{|S|}$

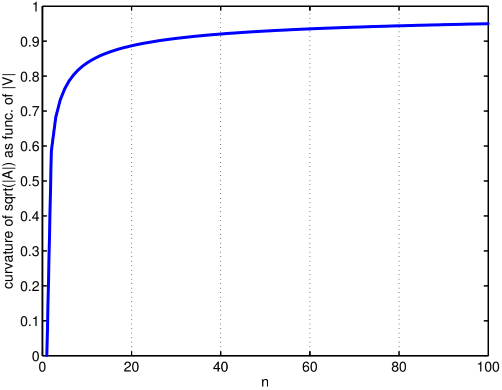
Curvature of $f(S) = \sqrt{|S|}$ as function of $|V| = n$



- $f(S) = \sqrt{|S|}$ with $|V| = n$ has curvature $1 - (\sqrt{n} - \sqrt{n-1})$.

Curvature for $f(S) = \sqrt{|S|}$

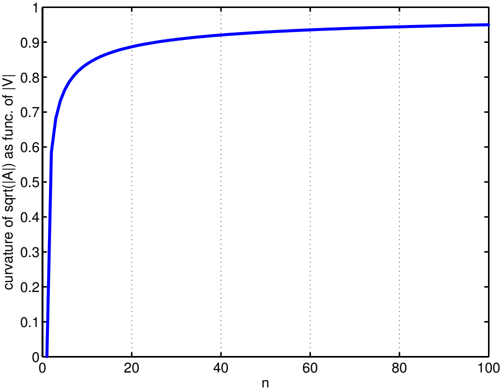
Curvature of $f(S) = \sqrt{|S|}$ as function of $|V| = n$



- $f(S) = \sqrt{|S|}$ with $|V| = n$ has curvature $1 - (\sqrt{n} - \sqrt{n-1})$.
- Approximation gets worse with bigger ground set.

Curvature for $f(S) = \sqrt{|S|}$

Curvature of $f(S) = \sqrt{|S|}$ as function of $|V| = n$



- $f(S) = \sqrt{|S|}$ with $|V| = n$ has curvature $1 - (\sqrt{n} - \sqrt{n-1})$.
- Approximation gets worse with bigger ground set.
- Functions of the form $f(S) = \sqrt{m(S)}$ where $m : V \to \mathbb{R}_+$, approximation worse with n if $\min_{i,j} |m(i) - m(j)|$ has a fixed lower bound with increasing n.

Curvature and approximation

- Curvature limitation can help the greedy algorithm in terms of approximation bounds.

Curvature and approximation

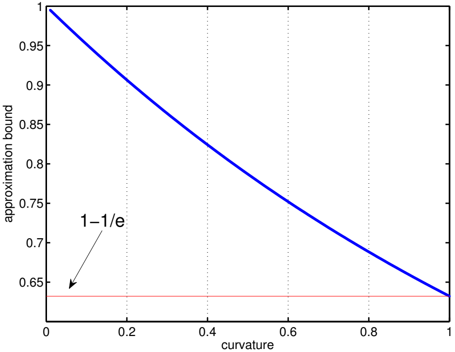
- Curvature limitation can help the greedy algorithm in terms of approximation bounds.
- Conforti & Cornuéjols showed that greedy gives a $1/(1+c)$ approximation to $\max\{f(S) : S \in \mathcal{I}\}$ when f has total curvature c.

Curvature and approximation

- Curvature limitation can help the greedy algorithm in terms of approximation bounds.
- Conforti & Cornuéjols showed that greedy gives a $1/(1+c)$ approximation to $\max\{f(S) : S \in \mathcal{I}\}$ when f has total curvature c.
- Hence, greedy subject to matroid constraint is a $\max(1/(1+c), 1/2)$ approximation algorithm, and if $c < 1$ then it is better than 1/2 (e.g., with $c = 1/4$ then we have a 0.8 algorithm).

Curvature and approximation

- Curvature limitation can help the greedy algorithm in terms of approximation bounds.
- Conforti & Cornuéjols showed that greedy gives a $1/(1+c)$ approximation to $\max\{f(S) : S \in \mathcal{I}\}$ when f has total curvature c.
- Hence, greedy subject to matroid constraint is a $\max(1/(1+c), 1/2)$ approximation algorithm, and if $c < 1$ then it is better than 1/2 (e.g., with $c = 1/4$ then we have a 0.8 algorithm).
- For k-uniform matroid (i.e., k-cardinality constraints), then approximation factor becomes $\frac{1}{c}(1 - e^{-c})$



Submodular and Supermodular Curvature Approximation

- Let f be a polymatroid function and let g be a non-negative monotone non-decreasing supermodular function (e.g., $g(A) = \phi(m(A))$ where $\phi()$ is non-decreasing convex).

Submodular and Supermodular Curvature Approximation

- Let f be a polymatroid function and let g be a non-negative monotone non-decreasing supermodular function (e.g., $g(A) = \phi(m(A))$ where $\phi()$ is non-decreasing convex).
- Let $\kappa_f = 1 - \min_v \frac{f(v|V\setminus\{v\})}{f(v)}$ be the total submodular total curvature,

Submodular and Supermodular Curvature Approximation

- Let f be a polymatroid function and let g be a non-negative monotone non-decreasing supermodular function (e.g., $g(A) = \phi(m(A))$ where $\phi()$ is non-decreasing convex).
- Let $\kappa_f = 1 - \min_v \frac{f(v|V \setminus \{v\})}{f(v)}$ be the total submodular total curvature,
- Define $\kappa^g = 1 - \min_v \frac{g(v)}{g(v|V \setminus \{v\})}$ as a "supermodular curvature"

Submodular and Supermodular Curvature Approximation

- Let f be a polymatroid function and let g be a non-negative monotone non-decreasing supermodular function (e.g., $g(A) = \phi(m(A))$ where $\phi()$ is non-decreasing convex).
- Let $\kappa_f = 1 - \min_v \frac{f(v|V \setminus \{v\})}{f(v)}$ be the total submodular total curvature,
- Define $\kappa^g = 1 - \min_v \frac{g(v)}{g(v|V \setminus \{v\})}$ as a "supermodular curvature"
- $\kappa^g \in [0, 1]$ and $\kappa^g = 0$ means g is modular, $\kappa^g = 1$ means g is "fully curved"

Submodular and Supermodular Curvature Approximation

- Let f be a polymatroid function and let g be a non-negative monotone non-decreasing supermodular function (e.g., $g(A) = \phi(m(A))$ where $\phi()$ is non-decreasing convex).
- Let $\kappa_f = 1 - \min_v \frac{f(v|V \setminus \{v\})}{f(v)}$ be the total submodular total curvature,
- Define $\kappa^g = 1 - \min_v \frac{g(v)}{g(v|V \setminus \{v\})}$ as a "supermodular curvature"
- $\kappa^g \in [0, 1]$ and $\kappa^g = 0$ means g is modular, $\kappa^g = 1$ means g is "fully curved"
- Form function $h(A) = f(A) + g(A)$, then h is neither suBmodular nor suPermodular, and is known as a BP-function.

Submodular and Supermodular Curvature Approximation

- Let f be a polymatroid function and let g be a non-negative monotone non-decreasing supermodular function (e.g., $g(A) = \phi(m(A))$ where $\phi()$ is non-decreasing convex).
- Let $\kappa_f = 1 - \min_v \frac{f(v|V\setminus\{v\})}{f(v)}$ be the total submodular total curvature,
- Define $\kappa^g = 1 - \min_v \frac{g(v)}{g(v|V\setminus\{v\})}$ as a "supermodular curvature"
- $\kappa^g \in [0,1]$ and $\kappa^g = 0$ means g is modular, $\kappa^g = 1$ means g is "fully curved"
- Form function $h(A) = f(A) + g(A)$, then h is neither suBmodular nor suPermodular, and is known as a BP-function.
- Then the greedy algorithm on h has a guarantee of: $\frac{1}{\kappa_f}(1 - e^{-(1-\kappa_g)\kappa_f})$.

Submodular and Supermodular Curvature Approximation

- Let f be a polymatroid function and let g be a non-negative monotone non-decreasing supermodular function (e.g., $g(A) = \phi(m(A))$ where $\phi()$ is non-decreasing convex).
- Let $\kappa_f = 1 - \min_v \frac{f(v|V\setminus\{v\})}{f(v)}$ be the total submodular total curvature,
- Define $\kappa^g = 1 - \min_v \frac{g(v)}{g(v|V\setminus\{v\})}$ as a "supermodular curvature"
- $\kappa^g \in [0, 1]$ and $\kappa^g = 0$ means g is modular, $\kappa^g = 1$ means g is "fully curved"
- Form function $h(A) = f(A) + g(A)$, then h is neither suBmodular nor suPermodular, and is known as a BP-function.
- Then the greedy algorithm on h has a guarantee of: $\frac{1}{\kappa_f}(1 - e^{-(1-\kappa_g)\kappa_f})$.
- For purely supermodular optimization (i.e., $\kappa_f = 0$) we get that greedy has a guarantee of $1 - \kappa_g$.

Generalizations

- Consider a k-uniform matroid $\mathcal{M} = (V, \mathcal{I})$ where $\mathcal{I} = \{S \subseteq V : |S| \leq k\}$, and consider problem $\max\{f(A) : A \in \mathcal{I}\}$

Generalizations

- Consider a k-uniform matroid $\mathcal{M} = (V, \mathcal{I})$ where $\mathcal{I} = \{S \subseteq V : |S| \leq k\}$, and consider problem $\max\{f(A) : A \in \mathcal{I}\}$
- Hence, the greedy algorithm is $1 - 1/e$ optimal for maximizing polymatroidal f subject to a k-uniform matroid constraint.

Generalizations

- Consider a k-uniform matroid $\mathcal{M} = (V, \mathcal{I})$ where $\mathcal{I} = \{S \subseteq V : |S| \leq k\}$, and consider problem $\max \{f(A) : A \in \mathcal{I}\}$
- Hence, the greedy algorithm is $1 - 1/e$ optimal for maximizing polymatroidal f subject to a k-uniform matroid constraint.
- Might be useful to allow an arbitrary matroid (e.g., partition matroid $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \leq k_i$ for all $i = 1, \ldots, \ell\}$., or a transversal, etc).

Generalizations

- Consider a k-uniform matroid $\mathcal{M} = (V, \mathcal{I})$ where $\mathcal{I} = \{S \subseteq V : |S| \leq k\}$, and consider problem $\max \{f(A) : A \in \mathcal{I}\}$
- Hence, the greedy algorithm is $1 - 1/e$ optimal for maximizing polymatroidal f subject to a k-uniform matroid constraint.
- Might be useful to allow an arbitrary matroid (e.g., partition matroid $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \leq k_i$ for all $i = 1, \dots, \ell\}$., or a transversal, etc).
- Knapsack constraint: if each item $v \in V$ has a cost $c(v)$, we may ask for $c(S) \leq b$ where b is a budget, in units of costs.

Generalizations

- Consider a k-uniform matroid $\mathcal{M} = (V, \mathcal{I})$ where $\mathcal{I} = \{S \subseteq V : |S| \leq k\}$, and consider problem $\max\{f(A) : A \in \mathcal{I}\}$
- Hence, the greedy algorithm is $1 - 1/e$ optimal for maximizing polymatroidal f subject to a k-uniform matroid constraint.
- Might be useful to allow an arbitrary matroid (e.g., partition matroid $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \leq k_i \text{ for all } i = 1, \dots, \ell\}$., or a transversal, etc).
- Knapsack constraint: if each item $v \in V$ has a cost $c(v)$, we may ask for $c(S) \leq b$ where b is a budget, in units of costs. Q: Is $\mathcal{I} = \{I : c(I) \leq b\}$ the independent sets of a matroid?

Generalizations

- Consider a k-uniform matroid $\mathcal{M} = (V, \mathcal{I})$ where $\mathcal{I} = \{S \subseteq V : |S| \leq k\}$, and consider problem $\max\{f(A) : A \in \mathcal{I}\}$
- Hence, the greedy algorithm is $1 - 1/e$ optimal for maximizing polymatroidal f subject to a k-uniform matroid constraint.
- Might be useful to allow an arbitrary matroid (e.g., partition matroid $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \leq k_i$ for all $i = 1, \ldots, \ell\}$., or a transversal, etc).
- Knapsack constraint: if each item $v \in V$ has a cost $c(v)$, we may ask for $c(S) \leq b$ where b is a budget, in units of costs. Q: Is $\mathcal{I} = \{I : c(I) \leq b\}$ the independent sets of a matroid?
- We may wish to maximize f subject to multiple matroid constraints. I.e., $S \in \mathcal{I}_1, S \in \mathcal{I}_2, \ldots, S \in \mathcal{I}_p$ where $\mathcal{I}_i$ are independent sets of the i^{th} matroid.

Generalizations

- Consider a k-uniform matroid $\mathcal{M} = (V, \mathcal{I})$ where $\mathcal{I} = \{S \subseteq V : |S| \leq k\}$, and consider problem $\max \{f(A) : A \in \mathcal{I}\}$
- Hence, the greedy algorithm is $1 - 1/e$ optimal for maximizing polymatroidal f subject to a k-uniform matroid constraint.
- Might be useful to allow an arbitrary matroid (e.g., partition matroid $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \leq k_i \text{ for all } i = 1, \ldots, \ell\}$., or a transversal, etc).
- Knapsack constraint: if each item $v \in V$ has a cost $c(v)$, we may ask for $c(S) \leq b$ where b is a budget, in units of costs. Q: Is $\mathcal{I} = \{I : c(I) \leq b\}$ the independent sets of a matroid?
- We may wish to maximize f subject to multiple matroid constraints. I.e., $S \in \mathcal{I}_1, S \in \mathcal{I}_2, \ldots, S \in \mathcal{I}_p$ where $\mathcal{I}_i$ are independent sets of the i^{th} matroid.
- Combinations of the above (e.g., knapsack & multiple matroid constraints).

Greedy over multiple matroids

- Obvious heuristic is to use the greedy step but always stay feasible.

Greedy over multiple matroids

- Obvious heuristic is to use the greedy step but always stay feasible.
- I.e., Starting with $S_0 = \emptyset$, we repeat the following greedy step

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^{p} \mathcal{I}_i} f(S_i \cup \{v\}) \right\} \tag{13.26}$$

Greedy over multiple matroids

- Obvious heuristic is to use the greedy step but always stay feasible.
- I.e., Starting with $S_0 = \emptyset$, we repeat the following greedy step

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^{p} \mathcal{I}_i} f(S_i \cup \{v\}) \right\} \tag{13.26}$$

- That is, we keep choosing next whatever feasible element looks best.

Greedy over multiple matroids

- Obvious heuristic is to use the greedy step but always stay feasible.
- I.e., Starting with $S_0 = \emptyset$, we repeat the following greedy step

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^{p} \mathcal{I}_i} f(S_i \cup \{v\}) \right\} \tag{13.26}$$

- That is, we keep choosing next whatever feasible element looks best.
- This algorithm is simple and also has a guarantee

Greedy over multiple matroids

- Obvious heuristic is to use the greedy step but always stay feasible.
- I.e., Starting with $S_0 = \emptyset$, we repeat the following greedy step

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^{p} \mathcal{I}_i}{\operatorname{argmax}} f(S_i \cup \{v\}) \right\} \tag{13.26}$$

- That is, we keep choosing next whatever feasible element looks best.
- This algorithm is simple and also has a guarantee

Theorem 13.5.1

Given a polymatroid function f, and set of matroids $\{M_j = (E, \mathcal{I}_j)\}_{j=1}^{p}$, the above greedy algorithm returns sets S_i such that for each i we have $f(S_i) \geq \frac{1}{p+1} \max_{|S| \leq i, S \in \bigcap_{i=1}^{p} \mathcal{I}_i} f(S)$, assuming such sets exists.

Greedy over multiple matroids

- Obvious heuristic is to use the greedy step but always stay feasible.
- I.e., Starting with $S_0 = \emptyset$, we repeat the following greedy step

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^{p} \mathcal{I}_i}{\operatorname{argmax}} f(S_i \cup \{v\}) \right\} \tag{13.26}$$

- That is, we keep choosing next whatever feasible element looks best.
- This algorithm is simple and also has a guarantee

Theorem 13.5.1

Given a polymatroid function f, and set of matroids $\{M_j = (E, \mathcal{I}_j)\}_{j=1}^p$, the above greedy algorithm returns sets S_i such that for each i we have $f(S_i) \geq \frac{1}{p+1} \max_{|S| \leq i, S \in \bigcap_{i=1}^p \mathcal{I}_i} f(S)$, assuming such sets exists.

- For one matroid, we have a 1/2 approximation.

Greedy over multiple matroids

- Obvious heuristic is to use the greedy step but always stay feasible.
- I.e., Starting with $S_0 = \emptyset$, we repeat the following greedy step

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^p \mathcal{I}_i} f(S_i \cup \{v\}) \right\} \tag{13.26}$$

- That is, we keep choosing next whatever feasible element looks best.
- This algorithm is simple and also has a guarantee

Theorem 13.5.1

Given a polymatroid function f, and set of matroids $\{M_j = (E, \mathcal{I}_j)\}_{j=1}^p$, the above greedy algorithm returns sets S_i such that for each i we have $f(S_i) \geq \frac{1}{p+1} \max_{|S| \leq i, S \in \bigcap_{i=1}^p \mathcal{I}_i} f(S)$, assuming such sets exists.

- For one matroid, we have a 1/2 approximation.
- Very easy algorithm, Minoux trick still possible, while addresses multiple matroid constraints

Greedy over multiple matroids

- Obvious heuristic is to use the greedy step but always stay feasible.
- I.e., Starting with $S_0 = \emptyset$, we repeat the following greedy step

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^{p} \mathcal{I}_i}{\operatorname{argmax}} f(S_i \cup \{v\}) \right\} \tag{13.26}$$

- That is, we keep choosing next whatever feasible element looks best.
- This algorithm is simple and also has a guarantee

Theorem 13.5.1

Given a polymatroid function f, and set of matroids $\{M_j = (E, \mathcal{I}_j)\}_{j=1}^{p}$, the above greedy algorithm returns sets S_i such that for each i we have $f(S_i) \geq \frac{1}{p+1} \max_{|S| \leq i, S \in \bigcap_{i=1}^{p} \mathcal{I}_i} f(S)$, assuming such sets exists.

- For one matroid, we have a 1/2 approximation.
- Very easy algorithm, Minoux trick still possible, while addresses multiple matroid constraints — but the bound is not that good when there are many matroids.

Matroid Intersection and Bipartite Matching

- Why might we want to do matroid intersection?

Matroid Intersection and Bipartite Matching

- Why might we want to do matroid intersection?
- Consider bipartite graph $G = (V, F, E)$. Define two partition matroids $M_V = (E, \mathcal{I}_V)$, and $M_F = (E, \mathcal{I}_F)$.

Matroid Intersection and Bipartite Matching

- Why might we want to do matroid intersection?
- Consider bipartite graph $G = (V, F, E)$. Define two partition matroids $M_V = (E, \mathcal{I}_V)$, and $M_F = (E, \mathcal{I}_F)$.
- Independence in each matroid corresponds to:

Matroid Intersection and Bipartite Matching

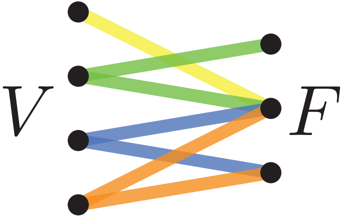
- Why might we want to do matroid intersection?
- Consider bipartite graph $G = (V, F, E)$. Define two partition matroids $M_V = (E, \mathcal{I}_V)$, and $M_F = (E, \mathcal{I}_F)$.
- Independence in each matroid corresponds to:
  1. $I \in \mathcal{I}_V$ if $|I \cap (V, f)| \leq 1$ for all $f \in F$,

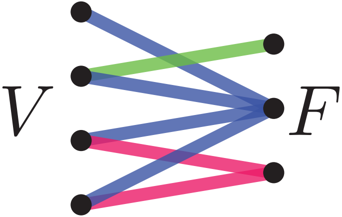
Matroid Intersection and Bipartite Matching

- Why might we want to do matroid intersection?
- Consider bipartite graph $G = (V, F, E)$. Define two partition matroids $M_V = (E, \mathcal{I}_V)$, and $M_F = (E, \mathcal{I}_F)$.
- Independence in each matroid corresponds to:
  1. $I \in \mathcal{I}_V$ if $|I \cap (V, f)| \leq 1$ for all $f \in F$,
  2. and $I \in \mathcal{I}_F$ if $|I \cap (v, F)| \leq 1$ for all $v \in V$.

Matroid Intersection and Bipartite Matching

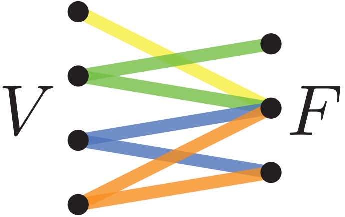
- Why might we want to do matroid intersection?
- Consider bipartite graph $G = (V, F, E)$. Define two partition matroids $M_V = (E, \mathcal{I}_V)$, and $M_F = (E, \mathcal{I}_F)$.
- Independence in each matroid corresponds to:
  1. $I \in \mathcal{I}_V$ if $|I \cap (V, f)| \leq 1$ for all $f \in F$,
  2. and $I \in \mathcal{I}_F$ if $|I \cap (v, F)| \leq 1$ for all $v \in V$.

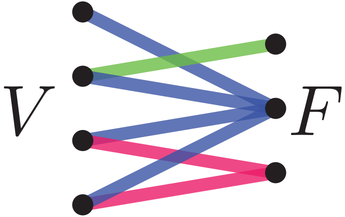




Matroid Intersection and Bipartite Matching

- Why might we want to do matroid intersection?
- Consider bipartite graph $G = (V, F, E)$. Define two partition matroids $M_V = (E, \mathcal{I}_V)$, and $M_F = (E, \mathcal{I}_F)$.
- Independence in each matroid corresponds to:
  1. $I \in \mathcal{I}_V$ if $|I \cap (V, f)| \leq 1$ for all $f \in F$,
  2. and $I \in \mathcal{I}_F$ if $|I \cap (v, F)| \leq 1$ for all $v \in V$.

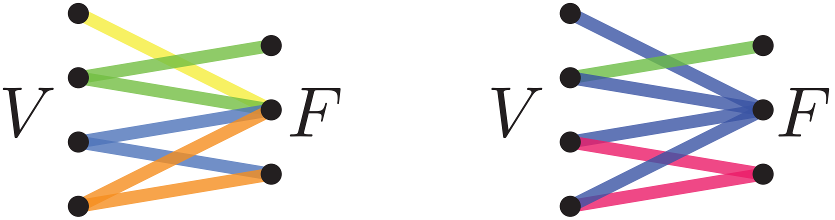




- Therefore, a matching in G is simultaneously independent in both M_V and M_F and finding the maximum matching is finding the maximum cardinality set independent in both matroids.

Matroid Intersection and Bipartite Matching

- Why might we want to do matroid intersection?
- Consider bipartite graph $G = (V, F, E)$. Define two partition matroids $M_V = (E, \mathcal{I}_V)$, and $M_F = (E, \mathcal{I}_F)$.
- Independence in each matroid corresponds to:
  1. $I \in \mathcal{I}_V$ if $|I \cap (V, f)| \leq 1$ for all $f \in F$,
  2. and $I \in \mathcal{I}_F$ if $|I \cap (v, F)| \leq 1$ for all $v \in V$.



- Therefore, a matching in G is simultaneously independent in both M_V and M_F and finding the maximum matching is finding the maximum cardinality set independent in both matroids.
- In bipartite graph case, therefore, can be solved in polynomial time.

Matroid Intersection and Network Communication

- Let $G_1 = (V_1, E)$ and $G_2 = (V_2, E)$ be two graphs on an isomorphic set of edges (lets just give them same names E).

Matroid Intersection and Network Communication

- Let $G_1 = (V_1, E)$ and $G_2 = (V_2, E)$ be two graphs on an isomorphic set of edges (lets just give them same names E).
- Consider two cycle matroids associated with these graphs $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$. They might be very different (e.g., an edge might be between two distinct nodes in G_1 but the same edge is a loop in multi-graph G_2.)

Matroid Intersection and Network Communication

- Let $G_1 = (V_1, E)$ and $G_2 = (V_2, E)$ be two graphs on an isomorphic set of edges (lets just give them same names E).
- Consider two cycle matroids associated with these graphs $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$. They might be very different (e.g., an edge might be between two distinct nodes in G_1 but the same edge is a loop in multi-graph G_2.)
- We may wish to find the maximum size edge-induced subgraph that is still forest in both graphs (i.e., adding any edges will create a circuit in either M_1, M_2, or both).

Matroid Intersection and Network Communication

- Let $G_1 = (V_1, E)$ and $G_2 = (V_2, E)$ be two graphs on an isomorphic set of edges (lets just give them same names E).
- Consider two cycle matroids associated with these graphs $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$. They might be very different (e.g., an edge might be between two distinct nodes in G_1 but the same edge is a loop in multi-graph G_2.)
- We may wish to find the maximum size edge-induced subgraph that is still forest in both graphs (i.e., adding any edges will create a circuit in either M_1, M_2, or both).
- This is again a matroid intersection problem.

Matroid Intersection and TSP

- Definition: a Hamiltonian cycle is a cycle that passes through each node exactly once.

Matroid Intersection and TSP

- Definition: a Hamiltonian cycle is a cycle that passes through each node exactly once.
- Given directed graph G, goal is to find such a Hamiltonian cycle.

Matroid Intersection and TSP

- Definition: a Hamiltonian cycle is a cycle that passes through each node exactly once.
- Given directed graph G, goal is to find such a Hamiltonian cycle.
- From G with n nodes, create G' with $n+1$ nodes by duplicating (w.l.o.g.) a particular node $v_1 \in V(G)$ to v_1^+, v_1^-, and have all outgoing edges from v_1 come instead from v_1^- and all edges incoming to v_1 go instead to v_1^+.

Matroid Intersection and TSP

- Definition: a Hamiltonian cycle is a cycle that passes through each node exactly once.
- Given directed graph G, goal is to find such a Hamiltonian cycle.
- From G with n nodes, create G' with $n+1$ nodes by duplicating (w.l.o.g.) a particular node $v_1 \in V(G)$ to v_1^+, v_1^-, and have all outgoing edges from v_1 come instead from v_1^- and all edges incoming to v_1 go instead to v_1^+.
- Let M_1 be the cycle matroid on G'.

Matroid Intersection and TSP

- Definition: a Hamiltonian cycle is a cycle that passes through each node exactly once.
- Given directed graph G, goal is to find such a Hamiltonian cycle.
- From G with n nodes, create G' with $n+1$ nodes by duplicating (w.l.o.g.) a particular node $v_1 \in V(G)$ to v_1^+, v_1^-, and have all outgoing edges from v_1 come instead from v_1^- and all edges incoming to v_1 go instead to v_1^+.
- Let M_1 be the cycle matroid on G'.
- Let M_2 be the partition matroid having as independent sets those that have no more than one edge leaving any node — i.e., $I \in \mathcal{I}(M_2)$ if $|I \cap \delta^-(v)| \leq 1$ for all $v \in V(G')$.

Matroid Intersection and TSP

- Definition: a Hamiltonian cycle is a cycle that passes through each node exactly once.
- Given directed graph G, goal is to find such a Hamiltonian cycle.
- From G with n nodes, create G' with $n+1$ nodes by duplicating (w.l.o.g.) a particular node $v_1 \in V(G)$ to v_1^+, v_1^-, and have all outgoing edges from v_1 come instead from v_1^- and all edges incoming to v_1 go instead to v_1^+.
- Let M_1 be the cycle matroid on G'.
- Let M_2 be the partition matroid having as independent sets those that have no more than one edge leaving any node — i.e., $I \in \mathcal{I}(M_2)$ if $|I \cap \delta^-(v)| \leq 1$ for all $v \in V(G')$.
- Let M_3 be the partition matroid having as independent sets those that have no more than one edge entering any node — i.e., $I \in \mathcal{I}(M_3)$ if $|I \cap \delta^+(v)| \leq 1$ for all $v \in V(G')$.

Matroid Intersection and TSP

- Definition: a Hamiltonian cycle is a cycle that passes through each node exactly once.
- Given directed graph G, goal is to find such a Hamiltonian cycle.
- From G with n nodes, create G' with $n+1$ nodes by duplicating (w.l.o.g.) a particular node $v_1 \in V(G)$ to v_1^+, v_1^-, and have all outgoing edges from v_1 come instead from v_1^- and all edges incoming to v_1 go instead to v_1^+.
- Let M_1 be the cycle matroid on G'.
- Let M_2 be the partition matroid having as independent sets those that have no more than one edge leaving any node — i.e., $I \in \mathcal{I}(M_2)$ if $|I \cap \delta^-(v)| \leq 1$ for all $v \in V(G')$.
- Let M_3 be the partition matroid having as independent sets those that have no more than one edge entering any node — i.e., $I \in \mathcal{I}(M_3)$ if $|I \cap \delta^+(v)| \leq 1$ for all $v \in V(G')$.
- Then a Hamiltonian cycle exists iff there is an n-element intersection of M_1, M_2, and M_3.

Matroid Intersection and TSP

- Recall, the traveling salesperson problem (TSP) is the problem to, given a directed graph, start at a node, visit all cities, and return to the starting point. Optimization version does this tour at minimum cost.

Matroid Intersection and TSP

- Recall, the traveling salesperson problem (TSP) is the problem to, given a directed graph, start at a node, visit all cities, and return to the starting point. Optimization version does this tour at minimum cost.
- Since TSP is NP-complete, we obviously can't solve matroid intersections of 3 more matroids, unless P=NP.

Matroid Intersection and TSP

- Recall, the traveling salesperson problem (TSP) is the problem to, given a directed graph, start at a node, visit all cities, and return to the starting point. Optimization version does this tour at minimum cost.
- Since TSP is NP-complete, we obviously can't solve matroid intersections of 3 more matroids, unless P=NP.
- But bipartite graph example gives us hope for 2 matroids, as in that case we can easily solve $\max |X|$ s.t. $x \in \mathcal{I}_1 \cap \mathcal{I}_2$.

Greedy over multiple matroids: Generalized Bipartite Matching

- Generalized bipartite matching (i.e., max bipartite matching with submodular costs on the edges). Use two partition matroids (as mentioned earlier in class)

Greedy over multiple matroids: Generalized Bipartite Matching

- Generalized bipartite matching (i.e., max bipartite matching with submodular costs on the edges). Use two partition matroids (as mentioned earlier in class)
- Useful in natural language processing: Ex. Computer language translation, find an alignment between two language strings.

Greedy over multiple matroids: Generalized Bipartite Matching

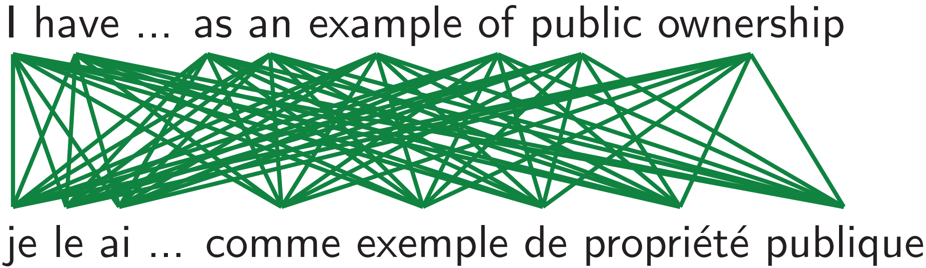
- Generalized bipartite matching (i.e., max bipartite matching with submodular costs on the edges). Use two partition matroids (as mentioned earlier in class)
- Useful in natural language processing: Ex. Computer language translation, find an alignment between two language strings.
- Consider bipartite graph $G = (E, F, V)$ where E and F are the left/right set of nodes, respectively, and V is the set of edges.

Greedy over multiple matroids: Generalized Bipartite Matching

- Generalized bipartite matching (i.e., max bipartite matching with submodular costs on the edges). Use two partition matroids (as mentioned earlier in class)
- Useful in natural language processing: Ex. Computer language translation, find an alignment between two language strings.
- Consider bipartite graph $G = (E, F, V)$ where E and F are the left/right set of nodes, respectively, and V is the set of edges.
- E corresponds to, say, an English language sentence and F corresponds to a French language sentence — goal is to form a matching (an alignment) between the two.

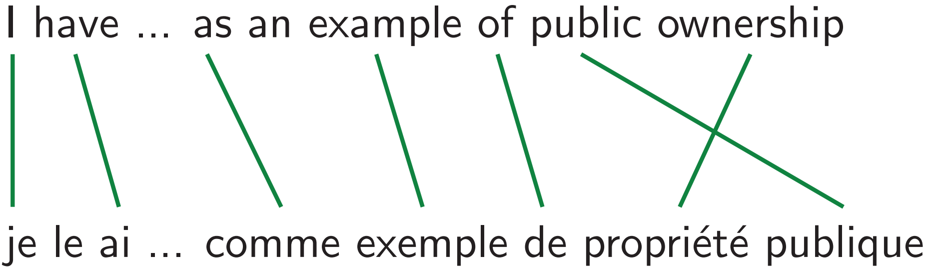
Greedy over > 1 matroids: Multiple Language Alignment

- Consider English string and French string, set up as a bipartite graph.



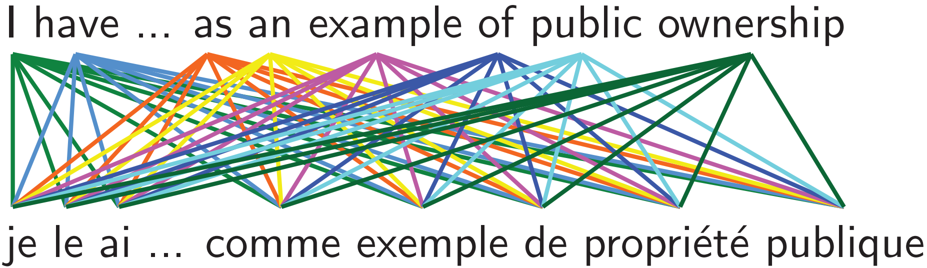
Greedy over > 1 matroids: Multiple Language Alignment

- One possible alignment, a matching, with score as sum of edge weights.



Greedy over > 1 matroids: Multiple Language Alignment

- Edges incident to English words constitute an edge partition

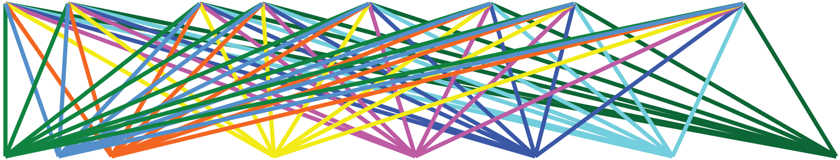


- The two edge partitions can be used to set up two 1-partition matroids on the edges.
- For each matroid, a set of edges is independent only if the set intersects each partition block no more than one time.

Greedy over > 1 matroids: Multiple Language Alignment

- Edges incident to French words constitute an edge partition

I have ... as an example of public ownership



je le ai ... comme exemple de propriété publique

- The two edge partitions can be used to set up two 1-partition matroids on the edges.
- For each matroid, a set of edges is independent only if the set intersects each partition block no more than one time.

Greedy over > 1 matroids: Multiple Language Alignment

- Typical to use bipartite matching to find an alignment between the two language strings.

Greedy over > 1 matroids: Multiple Language Alignment

- Typical to use bipartite matching to find an alignment between the two language strings.
- As we saw, this is equivalent to two 1-partition matroids and a non-negative modular cost function on the edges.

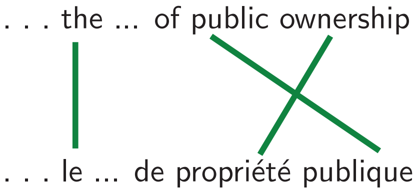
Greedy over > 1 matroids: Multiple Language Alignment

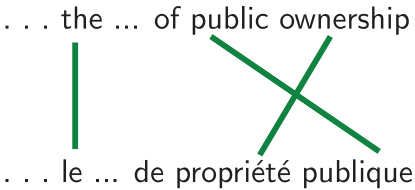
- Typical to use bipartite matching to find an alignment between the two language strings.
- As we saw, this is equivalent to two 1-partition matroids and a non-negative modular cost function on the edges.
- We can generalize this using a polymatroid cost function on the edges, and two k-partition matroids, allowing for "fertility" in the models:

Greedy over > 1 matroids: Multiple Language Alignment

- Typical to use bipartite matching to find an alignment between the two language strings.
- As we saw, this is equivalent to two 1-partition matroids and a non-negative modular cost function on the edges.
- We can generalize this using a polymatroid cost function on the edges, and two k-partition matroids, allowing for "fertility" in the models:

Fertility at most 1

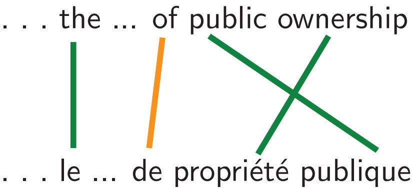


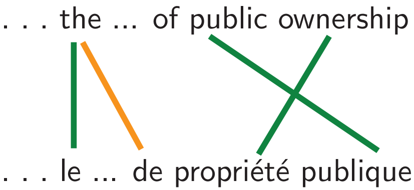


Greedy over > 1 matroids: Multiple Language Alignment

- Typical to use bipartite matching to find an alignment between the two language strings.
- As we saw, this is equivalent to two 1-partition matroids and a non-negative modular cost function on the edges.
- We can generalize this using a polymatroid cost function on the edges, and two k-partition matroids, allowing for "fertility" in the models:

Fertility at most 2





Greedy over > 1 matroids: Multiple Language Alignment

- Generalizing further, each block of edges in each partition matroid can have its own "fertility" limit:
  $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \leq k_i \text{ for all } i = 1, \ldots, \ell\}$.

Greedy over > 1 matroids: Multiple Language Alignment

- Generalizing further, each block of edges in each partition matroid can have its own "fertility" limit: $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \leq k_i \text{ for all } i = 1, \ldots, \ell\}$.
- Maximizing submodular function subject to multiple matroid constraints addresses this problem.

Greedy over multiple matroids: Submodular Welfare

- Submodular Welfare Maximization: Consider E a set of m goods to be distributed/partitioned among n people ("players").

Greedy over multiple matroids: Submodular Welfare

- Submodular Welfare Maximization: Consider E a set of m goods to be distributed/partitioned among n people ("players").
- Each players has a submodular "valuation" function, $g_i : 2^E \to \mathbb{R}_+$ that measures how "desirable" or "valuable" a given subset $A \subseteq E$ of goods are to that player.

Greedy over multiple matroids: Submodular Welfare

- Submodular Welfare Maximization: Consider E a set of m goods to be distributed/partitioned among n people ("players").
- Each players has a submodular "valuation" function, $g_i : 2^E \to \mathbb{R}_+$ that measures how "desirable" or "valuable" a given subset $A \subseteq E$ of goods are to that player.
- Assumption: No good can be shared between multiple players, each good must be allocated to a single player.

Greedy over multiple matroids: Submodular Welfare

- Submodular Welfare Maximization: Consider E a set of m goods to be distributed/partitioned among n people ("players").
- Each players has a submodular "valuation" function, $g_i : 2^E \to \mathbb{R}_+$ that measures how "desirable" or "valuable" a given subset $A \subseteq E$ of goods are to that player.
- Assumption: No good can be shared between multiple players, each good must be allocated to a single player.
- Goal of submodular welfare: Partition the goods $E = E_1 \cup E_2 \cup \dots \cup E_n$ into n blocks in order to maximize the submodular social welfare, measured as:

$$\text{submodular-social-welfare}(E_1, E_2, \ldots, E_n) = \sum_{i=1}^{n} g_i(E_i). \tag{13.27}$$

Greedy over multiple matroids: Submodular Welfare

- Submodular Welfare Maximization: Consider E a set of m goods to be distributed/partitioned among n people ("players").
- Each players has a submodular "valuation" function, $g_i : 2^E \to \mathbb{R}_+$ that measures how "desirable" or "valuable" a given subset $A \subseteq E$ of goods are to that player.
- Assumption: No good can be shared between multiple players, each good must be allocated to a single player.
- Goal of submodular welfare: Partition the goods $E = E_1 \cup E_2 \cup \dots \cup E_n$ into n blocks in order to maximize the submodular social welfare, measured as:

$$\text{submodular-social-welfare}(E_1, E_2, \dots, E_n) = \sum_{i=1}^{n} g_i(E_i). \tag{13.27}$$

- We can solve this via submodular maximization subject to multiple matroid independence constraints as we next describe ...

Submodular Welfare: Submodular Max over matroid partition

- Create new ground set E' as disjoint union of n copies of the ground set. I.e.,

$$E' = \underbrace{E \uplus E \uplus \cdots \uplus E}_{n\times} \tag{13.28}$$

Submodular Welfare: Submodular Max over matroid partition

- Create new ground set E' as disjoint union of n copies of the ground set. I.e.,

$$E' = \underbrace{E \uplus E \uplus \cdots \uplus E}_{n\times} \tag{13.28}$$

- Let $E^{(i)} \subset E'$ be the i^{th} block of E'.

Submodular Welfare: Submodular Max over matroid partition

- Create new ground set E' as disjoint union of n copies of the ground set. I.e.,

$$E' = \underbrace{E \uplus E \uplus \dots \uplus E}_{n\times} \tag{13.28}$$

- Let $E^{(i)} \subset E'$ be the i^{th} block of E'.
- For any $e \in E$, the corresponding element in $E^{(i)}$ is called $(e, i) \in E^{(i)}$ (each original element is tagged by integer).

Submodular Welfare: Submodular Max over matroid partition

- Create new ground set E' as disjoint union of n copies of the ground set. I.e.,

$$E' = \underbrace{E \uplus E \uplus \dots \uplus E}_{n\times} \tag{13.28}$$

- Let $E^{(i)} \subset E'$ be the i^{th} block of E'.
- For any $e \in E$, the corresponding element in $E^{(i)}$ is called $(e, i) \in E^{(i)}$ (each original element is tagged by integer).
- For $e \in E$, define $E_e = \{(e', i) \in E' : e' = e\}$.

Submodular Welfare: Submodular Max over matroid partition

- Create new ground set E' as disjoint union of n copies of the ground set. I.e.,

$$E' = \underbrace{E \uplus E \uplus \cdots \uplus E}_{n\times} \tag{13.28}$$

- Let $E^{(i)} \subset E'$ be the i^{th} block of E'.
- For any $e \in E$, the corresponding element in $E^{(i)}$ is called $(e, i) \in E^{(i)}$ (each original element is tagged by integer).
- For $e \in E$, define $E_e = \{(e', i) \in E' : e' = e\}$.
- Hence, $\{E_e\}_{e \in E}$ is a partition of E', each block of the partition for one of the original elements in E.

Submodular Welfare: Submodular Max over matroid partition

- Create new ground set E' as disjoint union of n copies of the ground set. I.e.,

$$E' = \underbrace{E \uplus E \uplus \cdots \uplus E}_{n\times} \tag{13.28}$$

- Let $E^{(i)} \subset E'$ be the i^{th} block of E'.
- For any $e \in E$, the corresponding element in $E^{(i)}$ is called $(e,i) \in E^{(i)}$ (each original element is tagged by integer).
- For $e \in E$, define $E_e = \{(e',i) \in E' : e' = e\}$.
- Hence, $\{E_e\}_{e\in E}$ is a partition of E', each block of the partition for one of the original elements in E.
- Create a 1-partition matroid $\mathcal{M} = (E', \mathcal{I})$ where

$$\mathcal{I} = \left\{S \subseteq E' : \forall e \in E, |S \cap E_e| \leq 1\right\} \tag{13.29}$$

Submodular Welfare: Submodular Max over matroid partition

- Hence, S is independent in matroid $\mathcal{M} = (E', I)$ if S uses each original element no more than once.

Submodular Welfare: Submodular Max over matroid partition

- Hence, S is independent in matroid $\mathcal{M} = (E', I)$ if S uses each original element no more than once.
- Create submodular function $f' : 2^{E'} \to \mathbb{R}_+$ with $f'(S) = \sum_{i=1}^{n} g_i(S \cap E^{(i)})$.

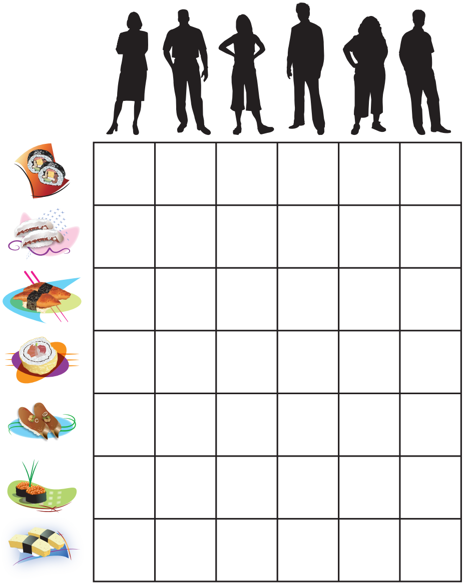
Submodular Welfare: Submodular Max over matroid partition

- Hence, S is independent in matroid $\mathcal{M} = (E', I)$ if S uses each original element no more than once.
- Create submodular function $f' : 2^{E'} \to \mathbb{R}_+$ with $f'(S) = \sum_{i=1}^n g_i(S \cap E^{(i)})$.
- Submodular welfare maximization becomes matroid constrained submodular max $\max\{f'(S) : S \in \mathcal{I}\}$, so greedy algorithm gives a $1/2$ approximation.

Submodular Social Welfare

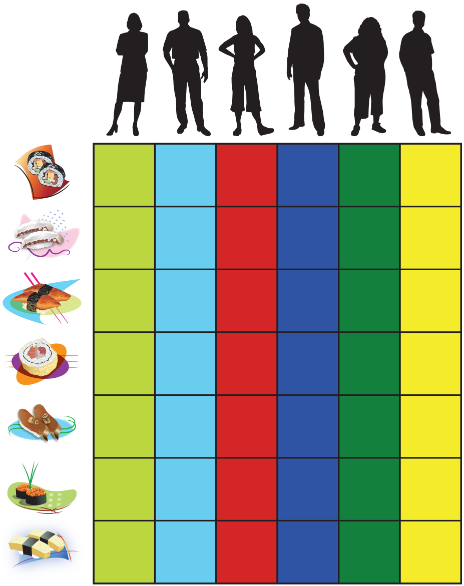
- Have $n = 6$ people (who don't like to share) and $|E| = m = 7$ pieces of sushi. E.g., $e \in E$ might be $e =$ "salmon roll".

Submodular Social Welfare



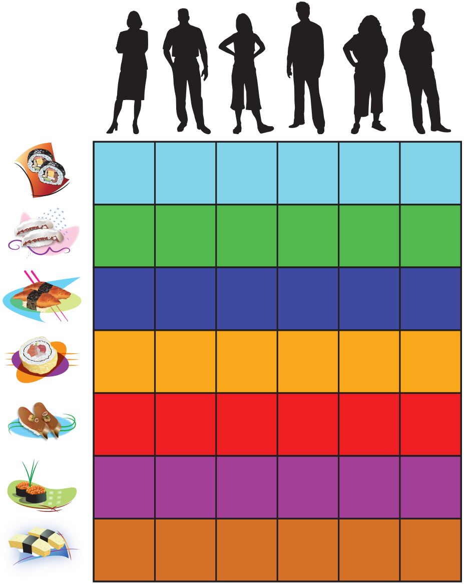
- Have $n = 6$ people (who don't like to share) and $|E| = m = 7$ pieces of sushi. E.g., $e \in E$ might be $e =$ "salmon roll".
- Goal: distribute sushi to people to maximize social welfare.

Submodular Social Welfare



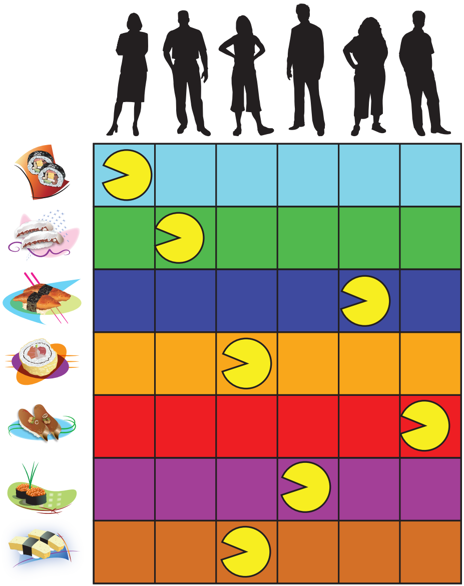
- Have $n = 6$ people (who don't like to share) and $|E| = m = 7$ pieces of sushi. E.g., $e \in E$ might be $e =$ "salmon roll".
- Goal: distribute sushi to people to maximize social welfare.
- Ground set disjoint union $E \uplus E \uplus E \uplus E \uplus E \uplus E$.

Submodular Social Welfare



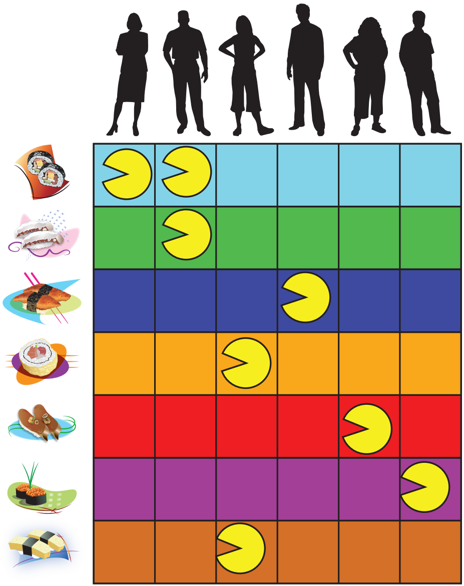
- Have $n = 6$ people (who don't like to share) and $|E| = m = 7$ pieces of sushi. E.g., $e \in E$ might be $e =$ "salmon roll".
- Goal: distribute sushi to people to maximize social welfare.
- Ground set disjoint union $E \uplus E \uplus E \uplus E \uplus E \uplus E$.
- Partition matroid partitions: $E_{e_1} \cup E_{e_2} \cup E_{e_3} \cup E_{e_4} \cup E_{e_5} \cup E_{e_6} \cup E_{e_7}$.

Submodular Social Welfare



- Have $n = 6$ people (who don't like to share) and $|E| = m = 7$ pieces of sushi. E.g., $e \in E$ might be $e =$ "salmon roll".
- Goal: distribute sushi to people to maximize social welfare.
- Ground set disjoint union $E \uplus E \uplus E \uplus E \uplus E \uplus E$.
- Partition matroid partitions: $E_{e_1} \cup E_{e_2} \cup E_{e_3} \cup E_{e_4} \cup E_{e_5} \cup E_{e_6} \cup E_{e_7}$.
- independent allocation

Submodular Social Welfare



- Have $n = 6$ people (who don't like to share) and $|E| = m = 7$ pieces of sushi. E.g., $e \in E$ might be $e =$ "salmon roll".
- Goal: distribute sushi to people to maximize social welfare.
- Ground set disjoint union $E \uplus E \uplus E \uplus E \uplus E \uplus E$.
- Partition matroid partitions: $E_{e_1} \cup E_{e_2} \cup E_{e_3} \cup E_{e_4} \cup E_{e_5} \cup E_{e_6} \cup E_{e_7}$.
- independent allocation
- non-independent allocation

Monotone Submodular over Knapsack Constraint

- The constraint $|A| \leq k$ is a simple cardinality constraint.

Monotone Submodular over Knapsack Constraint

- The constraint $|A| \leq k$ is a simple cardinality constraint.
- Consider a non-negative integral modular function $c : E \to \mathbb{Z}_+$.

Monotone Submodular over Knapsack Constraint

- The constraint $|A| \leq k$ is a simple cardinality constraint.
- Consider a non-negative integral modular function $c : E \to \mathbb{Z}_+$.
- A knapsack constraint would be of the form $c(A) \leq b$ where B is some integer budget that must not be exceeded. That is $\max\{f(A) : A \subseteq V, c(A) \leq b\}$.

Monotone Submodular over Knapsack Constraint

- The constraint $|A| \leq k$ is a simple cardinality constraint.
- Consider a non-negative integral modular function $c : E \to \mathbb{Z}_+$.
- A knapsack constraint would be of the form $c(A) \leq b$ where B is some integer budget that must not be exceeded. That is $\max \{f(A) : A \subseteq V, c(A) \leq b\}$.
- Important: A knapsack constraint yields an independence system (down closed) but it is not a matroid!

Monotone Submodular over Knapsack Constraint

- The constraint $|A| \leq k$ is a simple cardinality constraint.
- Consider a non-negative integral modular function $c : E \to \mathbb{Z}_+$.
- A knapsack constraint would be of the form $c(A) \leq b$ where B is some integer budget that must not be exceeded. That is $\max \{f(A) : A \subseteq V, c(A) \leq b\}$.
- Important: A knapsack constraint yields an independence system (down closed) but it is not a matroid!
- $c(e)$ may be seen as the cost of item e and if $c(e) = 1$ for all e, then we recover the cardinality constraint we saw earlier.

Monotone Submodular over Knapsack Constraint

- Greedy can be seen as choosing the best gain: Starting with $S_0 = \emptyset$, we repeat the following greedy step

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i}{\operatorname{argmax}} \Big(f(S_i \cup \{v\}) - f(S_i) \Big) \right\} \tag{13.30}$$

the gain is $f(\{v\}|S_i) = f(S_i + v) - f(S_i)$, so greedy just chooses next the currently unselected element with greatest gain.

Monotone Submodular over Knapsack Constraint

- Greedy can be seen as choosing the best gain: Starting with $S_0 = \emptyset$, we repeat the following greedy step

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i} \Big(f(S_i \cup \{v\}) - f(S_i) \Big) \right\} \tag{13.30}$$

the gain is $f(\{v\}|S_i) = f(S_i + v) - f(S_i)$, so greedy just chooses next the currently unselected element with greatest gain.

- Core idea in knapsack case: Greedy can be extended to choose next whatever looks cost-normalized best, i.e., Starting some initial set S_0, we repeat the following cost-normalized greedy step

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i} \frac{f(S_i \cup \{v\}) - f(S_i)}{c(v)} \right\} \tag{13.31}$$

which we repeat until $c(S_{i+1}) > b$ and then take S_i as the solution.

A Knapsack Constraint

- There are a number of ways of getting approximation bounds using this strategy.
- If we run the normalized greedy procedure starting with $S_0 = \emptyset$, and compare the solution found with the max of the singletons $\max_{v \in V} f(\{v\})$, choosing the max, then we get a $(1 - e^{-1/2}) \approx 0.39$ approximation, in $O(n^2)$ time (Minoux trick also possible for further speed)
- Partial enumeration: On the other hand, we can get a $(1 - e^{-1}) \approx 0.63$ approximation in $O(n^5)$ time if we run the above procedure starting from all sets of cardinality three (so restart for all S_0 such that $|S_0| = 3$), and compare that with the best singleton and pairwise solution.
- Extending something similar to this to d simultaneous knapsack constraints is possible as well.

Local Search Algorithms

From J. Vondrak

- Local search involves switching up to t elements, as long as it provides a (non-trivial) improvement; can iterate in several phases. Some examples follow:
- 1/3 approximation to unconstrained non-monotone maximization [Feige, Mirrokni, Vondrak, 2007]
- $1/(k + 2 + \frac{1}{k} + \delta_t)$ approximation for non-monotone maximization subject to k matroids [Lee, Mirrokni, Nagarajan, Sviridenko, 2009]
- $1/(k + \delta_t)$ approximation for monotone submodular maximization subject to $k \geq 2$ matroids [Lee, Sviridenko, Vondrak, 2010].

What About Non-monotone

- Alternatively, we may wish to maximize non-monotone submodular functions. This includes of course graph cuts, and this problem is APX-hard, so maximizing non-monotone functions, even unconstrainedly, is hard.

What About Non-monotone

- Alternatively, we may wish to maximize non-monotone submodular functions. This includes of course graph cuts, and this problem is APX-hard, so maximizing non-monotone functions, even unconstrainedly, is hard.
- If f is an arbitrary submodular function (so neither polymatroidal, nor necessarily positive or negative), then verifying if the maximum of f is positive or negative is already NP-hard.

What About Non-monotone

- Alternatively, we may wish to maximize non-monotone submodular functions. This includes of course graph cuts, and this problem is APX-hard, so maximizing non-monotone functions, even unconstrainedly, is hard.
- If f is an arbitrary submodular function (so neither polymatroidal, nor necessarily positive or negative), then verifying if the maximum of f is positive or negative is already NP-hard.
- Therefore, submodular function max in such case is inapproximable unless P=NP (since any such procedure would give us the sign of the max).

What About Non-monotone

- Alternatively, we may wish to maximize non-monotone submodular functions. This includes of course graph cuts, and this problem is APX-hard, so maximizing non-monotone functions, even unconstrainedly, is hard.
- If f is an arbitrary submodular function (so neither polymatroidal, nor necessarily positive or negative), then verifying if the maximum of f is positive or negative is already NP-hard.
- Therefore, submodular function max in such case is inapproximable unless P=NP (since any such procedure would give us the sign of the max).
- Thus, any approximation algorithm must be for unipolar submodular functions. E.g., non-negative but otherwise arbitrary submodular functions.

What About Non-monotone

- Alternatively, we may wish to maximize non-monotone submodular functions. This includes of course graph cuts, and this problem is APX-hard, so maximizing non-monotone functions, even unconstrainedly, is hard.
- If f is an arbitrary submodular function (so neither polymatroidal, nor necessarily positive or negative), then verifying if the maximum of f is positive or negative is already NP-hard.
- Therefore, submodular function max in such case is inapproximable unless P=NP (since any such procedure would give us the sign of the max).
- Thus, any approximation algorithm must be for unipolar submodular functions. E.g., non-negative but otherwise arbitrary submodular functions.
- We may get a $(\frac{1}{3} - \frac{\epsilon}{n})$ approximation for maximizing non-monotone non-negative submodular functions, with most $O(\frac{1}{\epsilon} n^3 \log n)$ function calls using approximate local maxima.

Submodularity and local optima

- Given any submodular function f, a set $S \subseteq V$ is a local maximum of f if $f(S - v) \leq f(S)$ for all $v \in S$ and $f(S + v) \leq f(S)$ for all $v \in V \setminus S$ (i.e., local in a Hamming ball of radius 1).

Submodularity and local optima

- Given any submodular function f, a set $S \subseteq V$ is a local maximum of f if $f(S - v) \leq f(S)$ for all $v \in S$ and $f(S + v) \leq f(S)$ for all $v \in V \setminus S$ (i.e., local in a Hamming ball of radius 1).
- The following interesting result is true for any submodular function:

Submodularity and local optima

- Given any submodular function f, a set $S \subseteq V$ is a local maximum of f if $f(S - v) \leq f(S)$ for all $v \in S$ and $f(S + v) \leq f(S)$ for all $v \in V \setminus S$ (i.e., local in a Hamming ball of radius 1).
- The following interesting result is true for any submodular function:

Lemma 13.6.1

Given a submodular function f, if S is a local maximum of f, and $I \subseteq S$ or $I \supseteq S$, then $f(I) \leq f(S)$.

Submodularity and local optima

- Given any submodular function f, a set $S \subseteq V$ is a local maximum of f if $f(S - v) \leq f(S)$ for all $v \in S$ and $f(S + v) \leq f(S)$ for all $v \in V \setminus S$ (i.e., local in a Hamming ball of radius 1).
- The following interesting result is true for any submodular function:

Lemma 13.6.1

Given a submodular function f, if S is a local maximum of f, and $I \subseteq S$ or $I \supseteq S$, then $f(I) \leq f(S)$.

- Idea of proof: Given $v_1, v_2 \in S$, suppose $f(S - v_1) \leq f(S)$ and $f(S - v_2) \leq f(S)$. Submodularity requires $f(S - v_1) + f(S - v_2) \geq f(S) + f(S - v_1 - v_2)$ which would be impossible unless $f(S - v_1 - v_2) \leq f(S)$.

Submodularity and local optima

- Given any submodular function f, a set $S \subseteq V$ is a local maximum of f if $f(S - v) \leq f(S)$ for all $v \in S$ and $f(S + v) \leq f(S)$ for all $v \in V \setminus S$ (i.e., local in a Hamming ball of radius 1).
- The following interesting result is true for any submodular function:

Lemma 13.6.1

Given a submodular function f, if S is a local maximum of f, and $I \subseteq S$ or $I \supseteq S$, then $f(I) \leq f(S)$.

- Idea of proof: Given $v_1, v_2 \in S$, suppose $f(S - v_1) \leq f(S)$ and $f(S - v_2) \leq f(S)$. Submodularity requires $f(S - v_1) + f(S - v_2) \geq f(S) + f(S - v_1 - v_2)$ which would be impossible unless $f(S - v_1 - v_2) \leq f(S)$.
- Similarly, given $v_1, v_2 \notin S$, and $f(S + v_1) \leq f(S)$ and $f(S + v_2) \leq f(S)$. Submodularity requires $f(S + v_1) + f(S + v_2) \geq f(S) + f(S + v_1 + v_2)$ which requires $f(S + v_1 + v_2) \leq f(S)$.

Submodularity and local optima

- Given any submodular function f, a set $S \subseteq V$ is a local maximum of f if $f(S - v) \leq f(S)$ for all $v \in S$ and $f(S + v) \leq f(S)$ for all $v \in V \setminus S$ (i.e., local in a Hamming ball of radius 1).
- The following interesting result is true for any submodular function:

Lemma 13.6.1

Given a submodular function f, if S is a local maximum of f, and $I \subseteq S$ or $I \supseteq S$, then $f(I) \leq f(S)$.

- In other words, once we have identified a local maximum, the two intervals in the Boolean lattice $[\emptyset, S]$ and $[S, V]$ can be ruled out as a possible improvement over S.

Submodularity and local optima

- Given any submodular function f, a set $S \subseteq V$ is a local maximum of f if $f(S - v) \leq f(S)$ for all $v \in S$ and $f(S + v) \leq f(S)$ for all $v \in V \setminus S$ (i.e., local in a Hamming ball of radius 1).
- The following interesting result is true for any submodular function:

Lemma 13.6.1

Given a submodular function f, if S is a local maximum of f, and $I \subseteq S$ or $I \supseteq S$, then $f(I) \leq f(S)$.

- In other words, once we have identified a local maximum, the two intervals in the Boolean lattice $[\emptyset, S]$ and $[S, V]$ can be ruled out as a possible improvement over S.
- Finding a local maximum is already hard (PLS-complete), but it is possible to find an approximate local maximum relatively efficiently.

Submodularity and local optima

- Given any submodular function f, a set $S \subseteq V$ is a local maximum of f if $f(S - v) \leq f(S)$ for all $v \in S$ and $f(S + v) \leq f(S)$ for all $v \in V \setminus S$ (i.e., local in a Hamming ball of radius 1).
- The following interesting result is true for any submodular function:

Lemma 13.6.1

Given a submodular function f, if S is a local maximum of f, and $I \subseteq S$ or $I \supseteq S$, then $f(I) \leq f(S)$.

- In other words, once we have identified a local maximum, the two intervals in the Boolean lattice $[\emptyset, S]$ and $[S, V]$ can be ruled out as a possible improvement over S.
- Finding a local maximum is already hard (PLS-complete), but it is possible to find an approximate local maximum relatively efficiently.
- This is the approach that yields the $(\frac{1}{3} - \frac{\epsilon}{n})$ approximation algorithm.

Linear time algorithm unconstrained non-monotone max

- Tight randomized tight $1/2$ approximation algorithm for unconstrained non-monotone non-negative submodular maximization.

Linear time algorithm unconstrained non-monotone max

- Tight randomized tight $1/2$ approximation algorithm for unconstrained non-monotone non-negative submodular maximization.
- Buchbinder, Feldman, Naor, Schwartz 2012.

Linear time algorithm unconstrained non-monotone max

- Tight randomized tight $1/2$ approximation algorithm for unconstrained non-monotone non-negative submodular maximization.
- Buchbinder, Feldman, Naor, Schwartz 2012. Recall $[a]_+ = \max(a, 0)$.

Linear time algorithm unconstrained non-monotone max

- Tight randomized tight $1/2$ approximation algorithm for unconstrained non-monotone non-negative submodular maximization.
- Buchbinder, Feldman, Naor, Schwartz 2012. Recall $[a]_+ = \max(a, 0)$.

Algorithm 6: Randomized Linear-time non-monotone submodular max

```
Set L ← ∅ ; U ← V     /* Lower L, upper U. Invariant: L ⊆ U */ ;
Order elements of V = (v_1, v_2, ..., v_n) arbitrarily ;
for i ← 0 ... |V| do
    a ← [f(v_i|L)]_+; b ← [−f(U|U \ {v_i})]_+ ;
    if a = b = 0 then p ← 1/2 ;
    ;
    else p ← a/(a + b);
    ;
    if Flip of coin with Pr(heads) = p draws heads then
        L ← L ∪ {v_i} ;
    Otherwise /* if the coin drew tails, an event with prob. 1 − p */
        U ← U \ {v}
return L (which is the same as U at this point)
```

Linear time algorithm unconstrained non-monotone max

- Each “sweep” of the algorithm is $O(n)$.

Linear time algorithm unconstrained non-monotone max

- Each "sweep" of the algorithm is $O(n)$.
- Running the algorithm $1\times$ (with an arbitrary variable order) results in a $1/3$ approximation.

Linear time algorithm unconstrained non-monotone max

- Each "sweep" of the algorithm is $O(n)$.
- Running the algorithm $1\times$ (with an arbitrary variable order) results in a $1/3$ approximation.
- The $1/2$ guarantee is in expected value (the expected solution has the $1/2$ guarantee).

Linear time algorithm unconstrained non-monotone max

- Each "sweep" of the algorithm is $O(n)$.
- Running the algorithm $1\times$ (with an arbitrary variable order) results in a $1/3$ approximation.
- The $1/2$ guarantee is in expected value (the expected solution has the $1/2$ guarantee).
- In practice, run it multiple times, each with a different random permutation of the elements, and then take the cumulative best.

Linear time algorithm unconstrained non-monotone max

- Each "sweep" of the algorithm is $O(n)$.
- Running the algorithm $1\times$ (with an arbitrary variable order) results in a $1/3$ approximation.
- The $1/2$ guarantee is in expected value (the expected solution has the $1/2$ guarantee).
- In practice, run it multiple times, each with a different random permutation of the elements, and then take the cumulative best.
- It may be possible to choose the random order smartly to get better results in practice.

More general still: multiple constraints different types

- In the past several years, there has been a plethora of papers on maximizing both monotone and non-monotone submodular functions under various combinations of one or more knapsack and/or matroid constraints.

More general still: multiple constraints different types

- In the past several years, there has been a plethora of papers on maximizing both monotone and non-monotone submodular functions under various combinations of one or more knapsack and/or matroid constraints.
- The approximation quality is usually some function of the number of matroids, and is often not a function of the number of knapsacks.

More general still: multiple constraints different types

- In the past several years, there has been a plethora of papers on maximizing both monotone and non-monotone submodular functions under various combinations of one or more knapsack and/or matroid constraints.
- The approximation quality is usually some function of the number of matroids, and is often not a function of the number of knapsacks.
- Often the computational costs of the algorithms are prohibitive (e.g., exponential in k) with large constants, so these algorithms might not scale.

More general still: multiple constraints different types

- In the past several years, there has been a plethora of papers on maximizing both monotone and non-monotone submodular functions under various combinations of one or more knapsack and/or matroid constraints.
- The approximation quality is usually some function of the number of matroids, and is often not a function of the number of knapsacks.
- Often the computational costs of the algorithms are prohibitive (e.g., exponential in k) with large constants, so these algorithms might not scale.
- On the other hand, these algorithms offer deep and interesting intuition into submodular functions, beyond what we have covered here.

Some results on submodular maximization

- As we've seen, we can get $1 - 1/e$ for non-negative monotone submodular (polymatroid) functions with greedy algorithm under cardinality constraints, and this is tight.

Some results on submodular maximization

- As we've seen, we can get $1 - 1/e$ for non-negative monotone submodular (polymatroid) functions with greedy algorithm under cardinality constraints, and this is tight.
- For general matroid, greedy reduces to $1/2$ approximation (as we've seen).

Some results on submodular maximization

- As we've seen, we can get $1 - 1/e$ for non-negative monotone submodular (polymatroid) functions with greedy algorithm under cardinality constraints, and this is tight.
- For general matroid, greedy reduces to $1/2$ approximation (as we've seen).
- We can recover $1 - 1/e$ approximation using the continuous greedy algorithm on the multilinear extension and then using pipage rounding to re-integerize the solution (see J. Vondrak's publications).

Some results on submodular maximization

- As we've seen, we can get $1 - 1/e$ for non-negative monotone submodular (polymatroid) functions with greedy algorithm under cardinality constraints, and this is tight.
- For general matroid, greedy reduces to $1/2$ approximation (as we've seen).
- We can recover $1 - 1/e$ approximation using the continuous greedy algorithm on the multilinear extension and then using pipage rounding to re-integerize the solution (see J. Vondrak's publications).
- More general constraints are possible too, as we see on the next table (for references, see Jan Vondrak's publications `http://theory.stanford.edu/~jvondrak/`).

Submodular Max Summary - 2012: From J. Vondrak

Monotone Maximization

Constraint	Approximation	Hardness	Technique
$\|S\| \leq k$	$1 - 1/e$	$1 - 1/e$	greedy
matroid	$1 - 1/e$	$1 - 1/e$	multilinear ext.
$O(1)$ knapsacks	$1 - 1/e$	$1 - 1/e$	multilinear ext.
k matroids	$k + \epsilon$	$k/\log k$	local search
k matroids and $O(1)$ knapsacks	$O(k)$	$k/\log k$	multilinear ext.

Nonmonotone Maximization

Constraint	Approximation	Hardness	Technique
Unconstrained	$1/2$	$1/2$	combinatorial
matroid	$1/e$	0.48	multilinear ext.
$O(1)$ knapsacks	$1/e$	0.49	multilinear ext.
k matroids	$k + O(1)$	$k/\log k$	local search
k matroids and $O(1)$ knapsacks	$O(k)$	$k/\log k$	multilinear ext.