Submodular Functions, Optimization, and Applications to Machine Learning

— Spring Quarter, Lecture 19 —

http://j.ee.washington.edu/~bilmes/classes/ee596b_spring_2014/

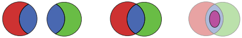
Prof. Jeff Bilmes

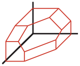
University of Washington, Seattle
Department of Electrical Engineering
http://melodi.ee.washington.edu/~bilmes

June 4th, 2014

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

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





Cumulative Outstanding Reading

- Good references for today: Schrijver-2003, Oxley-1992/2011, Welsh-1973, Goemans-2010, Cunningham-1984, Edmonds-1969, Choquet-1955, Grabisch/Marichal/Mesiar/Pap "Aggregation Functions", Lovász-1983, Bach-2011.
- Read Tom McCormick's overview paper on SFM http://people.commerce.ubc.ca/faculty/mccormick/sfmchap8a.pdf
- Read chapters 1 - 4 from Fujishige book.
- Matroid properties http://www-math.mit.edu/~goemans/18433S09/matroid-notes.pdf
- Read lecture 14 slides on lattice theory at our web page (http://j.ee.washington.edu/~bilmes/classes/ee596b_spring_2014/)
- Wolfe "Finding the Nearest Point in a Polytope", 1976.
- Fujishige & Isotani, "A Submodular Function Minimization Algorithm Based on the Minimum-Norm Base", 2009.

Sources for Today's Lecture

- "Submodular Function Maximization", Krause and Golovin.
- Chekuri, Vondrak, Zenklusen, "Submodular Function Maximization via the Multilinear Relaxation and Contention Resolution Schemes", 2011 (a recent paper (appeared yesterday) that, among other things, has a nice up-to-date summary on all the results on submodular max).
- Minoux, "Accelerated Greedy Algorithms for Maximizing Submodular Set Functions", 1977.
- Feige, Mirrokni, Vondrak, "Maximizing non-monotone submodular functions", 2007.
- Fujishige, "Submodular Functions and Optimization", 2005.
- Fujishige, "Submodular Systems and Related Topics", 1984.
- Fisher, Nemhauser, Wolsey, "An Analysis of Approximations for Maximizing Submodular Set Functions - II", 1978.
- Lin & Bilmes, "A Class Of Submodular Functions for Document Summarization", 2011.

Other readings

- J. Vondrak, "Submodularity and curvature: the optimal algorithm" in RIMS Kokyuroku Bessatsu B23, Kyoto, 2010.
- M. Conforti and G. Cornuéjols. Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the Rado-Edmonds theorem. Discrete Applied Math, 7(3):251-274, 1984.

Announcements, Assignments, and Reminders

- Weekly Office Hours: Wednesdays, 5:00-5:50, or by skype or google hangout (email me).

Class Road Map - IT-I

- L1 (3/31): Motivation, Applications, & Basic Definitions
- L2: (4/2): Applications, Basic Definitions, Properties
- L3: More examples and properties (e.g., closure properties), and examples, spanning trees
- L4: proofs of equivalent definitions, independence, start matroids
- L5: matroids, basic definitions and examples
- L6: More on matroids, System of Distinct Reps, Transversals, Transversal Matroid, Matroid and representation
- L7: Dual Matroids, other matroid properties, Combinatorial Geometries
- L8: Combinatorial Geometries, matroids and greedy, Polyhedra, Matroid Polytopes,
- L9: From Matroid Polytopes to Polymatroids.
- L10: Polymatroids and Submodularity
- L11: More properties of polymatroids, SFM special cases
- L12: polymatroid properties, extreme points polymatroids,
- L13: sat, dep, supp, exchange capacity, examples
- L14: Lattice theory: partially ordered sets; lattices; distributive, modular, submodular, and boolean lattices; ideals and join irreducibles.
- L15: Supp, Base polytope, polymatroids and entropic Venn diagrams, exchange capacity,
- L16: proof that minimum norm point yields min of submodular function, and the lattice of minimizers of a submodular function, Lovasz extension
- L17: Lovasz extension, Choquet Integration, more properties/examples of Lovasz extension, convex minimization and SFM.
- L18: Lovasz extension examples and structured convex norms, The Min-Norm Point Algorithm detailed.
- L19: symmetric submodular function minimization, maximizing monotone submodular function w. card and other constraints.
- L20:

Finals Week: June 9th-13th, 2014.

Symmetric Submodular Functions

- Given: $\breve{f} : 2^E \to \mathbb{R}$, if $\breve{f}$ is submodular and also has the property that $\breve{f}(A) = \breve{f}(E \setminus A)$ for all A, then $\breve{f}$ is said to be symmetric submodular

Symmetric Submodular Functions

- Given: $\breve{f} : 2^E \to \mathbb{R}$, if $\breve{f}$ is submodular and also has the property that $\breve{f}(A) = \breve{f}(E \setminus A)$ for all A, then $\breve{f}$ is said to be symmetric submodular
- Given any non-symmetric submodular function f, we can always symmetrize it, $f_{\text{symmetric}}(A) = f(A) + f(E \setminus A)$.

Symmetric Submodular Functions

- Given: $\breve{f} : 2^E \to \mathbb{R}$, if $\breve{f}$ is submodular and also has the property that $\breve{f}(A) = \breve{f}(E \setminus A)$ for all A, then $\breve{f}$ is said to be symmetric submodular
- Given any non-symmetric submodular function f, we can always symmetrize it, $f_{\text{symmetric}}(A) = f(A) + f(E \setminus A)$.
- *Symmetrize* and *normalize* f as $f \to \breve{f}$ via the operation: $\breve{f}(A) = f(A) + f(E \setminus A) - f(E)$, so that $\breve{f}(\emptyset) = 0$ if $f(\emptyset) = 0$.

Symmetric Submodular Functions

- Given: $\breve{f} : 2^E \to \mathbb{R}$, if $\breve{f}$ is submodular and also has the property that $\breve{f}(A) = \breve{f}(E \setminus A)$ for all A, then $\breve{f}$ is said to be symmetric submodular
- Given any non-symmetric submodular function f, we can always symmetrize it, $f_{\text{symmetric}}(A) = f(A) + f(E \setminus A)$.
- *Symmetrize* and *normalize* f as $f \to \breve{f}$ via the operation: $\breve{f}(A) = f(A) + f(E \setminus A) - f(E)$, so that $\breve{f}(\emptyset) = 0$ if $f(\emptyset) = 0$.
- Such an $\breve{f}$ is also non-negative since

$$2\breve{f}(A) = \breve{f}(A) + \breve{f}(E \setminus A) \geq \breve{f}(\emptyset) + \breve{f}(E) = 2\breve{f}(\emptyset) \geq 0 \tag{19.1}$$

$\breve{f}(A) \geq 0$

Symmetric Submodular Functions

- Given: $\breve{f} : 2^E \to \mathbb{R}$, if $\breve{f}$ is submodular and also has the property that $\breve{f}(A) = \breve{f}(E \setminus A)$ for all A, then $\breve{f}$ is said to be symmetric submodular
- Given any non-symmetric submodular function f, we can always symmetrize it, $f_{\text{symmetric}}(A) = f(A) + f(E \setminus A)$.
- *Symmetrize* and *normalize* f as $f \to \breve{f}$ via the operation: $\breve{f}(A) = f(A) + f(E \setminus A) - f(E)$, so that $\breve{f}(\emptyset) = 0$ if $f(\emptyset) = 0$.
- Such an $\breve{f}$ is also non-negative since

$$2\breve{f}(A) = \breve{f}(A) + \breve{f}(E \setminus A) \geq \breve{f}(\emptyset) + \breve{f}(E) = 2\breve{f}(\emptyset) \geq 0 \tag{19.1}$$

- Equivalence class: $f \to \breve{f}$ same up to modular shift since $\breve{f} = \breve{g}$ if $f = g + m$ with m modular $\Rightarrow$ consider only polymatroidal f.

Symmetric Submodular Functions

- Given: $\breve{f} : 2^E \to \mathbb{R}$, if $\breve{f}$ is submodular and also has the property that $\breve{f}(A) = \breve{f}(E \setminus A)$ for all A, then $\breve{f}$ is said to be symmetric submodular
- Given any non-symmetric submodular function f, we can always symmetrize it, $f_{\text{symmetric}}(A) = f(A) + f(E \setminus A)$.
- *Symmetrize* and *normalize* f as $f \to \breve{f}$ via the operation: $\breve{f}(A) = f(A) + f(E \setminus A) - f(E)$, so that $\breve{f}(\emptyset) = 0$ if $f(\emptyset) = 0$.
- Such an $\breve{f}$ is also non-negative since

$$2\breve{f}(A) = \breve{f}(A) + \breve{f}(E \setminus A) \geq \breve{f}(\emptyset) + \breve{f}(E) = 2\breve{f}(\emptyset) \geq 0 \quad (19.1)$$

- Equivalence class: $f \to \breve{f}$ same up to modular shift since $\breve{f} = \breve{g}$ if $f = g + m$ with m modular $\Rightarrow$ consider only polymatroidal f.
- Combinatorial mutual information function, so $\breve{f}(A) = I_f(A; V \setminus A)$ where $I_f(A; B) = f(A) + f(B) - f(A \cup B) - f(A \cap B)$.

Symmetric Submodular Functions

- Given: $\breve{f} : 2^E \to \mathbb{R}$, if $\breve{f}$ is submodular and also has the property that $\breve{f}(A) = \breve{f}(E \setminus A)$ for all A, then $\breve{f}$ is said to be symmetric submodular
- Given any non-symmetric submodular function f, we can always symmetrize it, $f_{\text{symmetric}}(A) = f(A) + f(E \setminus A)$.
- *Symmetrize* and *normalize* f as $f \to \breve{f}$ via the operation: $\breve{f}(A) = f(A) + f(E \setminus A) - f(E)$, so that $\breve{f}(\emptyset) = 0$ if $f(\emptyset) = 0$.
- Such an $\breve{f}$ is also non-negative since

$$2\breve{f}(A) = \breve{f}(A) + \breve{f}(E \setminus A) \geq \breve{f}(\emptyset) + \breve{f}(E) = 2\breve{f}(\emptyset) \geq 0 \qquad (19.1)$$

- Equivalence class: $f \to \breve{f}$ same up to modular shift since $\breve{f} = \breve{g}$ if $f = g + m$ with m modular $\Rightarrow$ consider only polymatroidal f.
- Combinatorial mutual information function, so $\breve{f}(A) = I_f(A; V \setminus A)$ where $I_f(A; B) = f(A) + f(B) - f(A \cup B) - f(A \cap B)$.
- Example: $f(A) = H(X_A) =$ entropy, then $\breve{f} = I(X_A; X_{E \setminus A}) =$ symmetric mutual information.

Separators of submodular function via symmetrized version

- Such a symmetrized submodular function measures a form of "dependence" between A and $\bar{A} \triangleq E \setminus A$.

Separators of submodular function via symmetrized version

- Such a symmetrized submodular function measures a form of "dependence" between A and $\bar{A} \triangleq E \setminus A$.

Theorem 19.3.1

We are given an f that is normalized & submodular. If $\exists A$ s.t. $\breve{f}(A) \triangleq f(A) + f(\bar{A}) - f(E) = 0$ then f is "decomposable" w.r.t. A — this means $f(B) = f(B \cap A) + f(B \cap \bar{A}), \forall B$.

Separators of submodular function via symmetrized version

- Such a symmetrized submodular function measures a form of "dependence" between A and $\bar{A} \triangleq E \setminus A$.

Theorem 19.3.1

We are given an f that is normalized & submodular. If $\exists A$ s.t. $\breve{f}(A) \triangleq f(A) + f(\bar{A}) - f(E) = 0$ then f is "decomposable" w.r.t. A — this means $f(B) = f(B \cap A) + f(B \cap \bar{A})$, $\forall B$.

Proof.

...

Separators of submodular function via symmetrized version

- Such a symmetrized submodular function measures a form of "dependence" between A and $\bar{A} \triangleq E \setminus A$.

Theorem 19.3.1

We are given an f that is normalized & submodular. If $\exists A$ s.t. $\breve{f}(A) \triangleq f(A) + f(\bar{A}) - f(E) = 0$ then f is "decomposable" w.r.t. A — this means $f(B) = f(B \cap A) + f(B \cap \bar{A})$, $\forall B$.

Proof.

- By submodularity (subadditivity for non-intersecting sets), we have:

$$f(B) = f\Big((B \cap A) \cup (B \cap \bar{A})\Big) \leq f(B \cap A) + f(B \cap \bar{A}) \tag{19.2}$$

...

Separators of submodular function via symmetrized version

- Such a symmetrized submodular function measures a form of "dependence" between A and $\bar{A} \triangleq E \setminus A$.

Theorem 19.3.1

We are given an f that is normalized & submodular. If $\exists A$ s.t. $\breve{f}(A) \triangleq f(A) + f(\bar{A}) - f(E) = 0$ then f is "decomposable" w.r.t. A — this means $f(B) = f(B \cap A) + f(B \cap \bar{A})$, $\forall B$.

Proof.

- By submodularity (subadditivity for non-intersecting sets), we have:

$$f(B) = f\Big((B \cap A) \cup (B \cap \bar{A})\Big) \leq f(B \cap A) + f(B \cap \bar{A}) \tag{19.2}$$

- Hence, $f(B) \leq f(B \cap A) + f(B \cap \bar{A})$.

...

Separators of submodular function via symmetrized version

. . . proof of Theorem 19.3.1 cont.

□

Separators of submodular function via symmetrized version

. . . proof of Theorem 19.3.1 cont.

- By submodularity

$f(B) - f(B \cap A) - f(B \cap \bar{A})$

$\square$

Separators of submodular function via symmetrized version

. . . proof of Theorem 19.3.1 cont.

- By submodularity

$$f(B) - f(B \cap A) - f(B \cap \bar{A}) \geq f(A \cup B) - f(A) - f(B \cap \bar{A}) \tag{19.3}$$

$\square$

Separators of submodular function via symmetrized version

. . . proof of Theorem 19.3.1 cont.

- By submodularity

$$f(B) - f(B \cap A) - f(B \cap \bar{A}) \geq f(A \cup B) - f(A) - f(B \cap \bar{A}) \quad (19.3)$$
$$\geq f((A \cup B) \cup \bar{A}) - f(A) - f(\bar{A}) \quad (19.4)$$

$\square$

Separators of submodular function via symmetrized version

. . . proof of Theorem 19.3.1 cont.

- By submodularity

$$f(B) - f(B \cap A) - f(B \cap \bar{A}) \geq f(A \cup B) - f(A) - f(B \cap \bar{A}) \quad (19.3)$$

$$\geq f((A \cup B) \cup \bar{A}) - f(A) - f(\bar{A}) \quad (19.4)$$

$$= f(E) - f(A) + f(\bar{A}) = 0 \quad (19.5)$$

$\square$

Separators of submodular function via symmetrized version

. . . proof of Theorem 19.3.1 cont.

- By submodularity

$$f(B) - f(B \cap A) - f(B \cap \bar{A}) \geq f(A \cup B) - f(A) - f(B \cap \bar{A}) \quad (19.3)$$
$$\geq f((A \cup B) \cup \bar{A}) - f(A) - f(\bar{A}) \quad (19.4)$$
$$= f(E) - f(A) + f(\bar{A}) = 0 \quad (19.5)$$

- Eqn. (19.3) follows since $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$,

$\square$

Separators of submodular function via symmetrized version

. . . proof of Theorem 19.3.1 cont.

- By submodularity

$$f(B) - f(B \cap A) - f(B \cap \bar{A}) \geq f(A \cup B) - f(A) - f(B \cap \bar{A}) \quad (19.3)$$
$$\geq f((A \cup B) \cup \bar{A}) - f(A) - f(\bar{A}) \quad (19.4)$$
$$= f(E) - f(A) + f(\bar{A}) = 0 \quad (19.5)$$

- Eqn. (19.3) follows since $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$, and Eqn. (19.4) follows since $B \cap \bar{A} = (A \cup B) \cap \bar{A}$ and $f(A \cup B) + f(\bar{A}) \geq f((A \cup B) \cup \bar{A}) + f((A \cup B) \cap \bar{A})$.

$\square$

Separators of submodular function via symmetrized version

. . . proof of Theorem 19.3.1 cont.

- By submodularity

$$f(B) - f(B \cap A) - f(B \cap \bar{A}) \geq f(A \cup B) - f(A) - f(B \cap \bar{A}) \tag{19.3}$$
$$\geq f((A \cup B) \cup \bar{A}) - f(A) - f(\bar{A}) \tag{19.4}$$
$$= f(E) - f(A) + f(\bar{A}) = 0 \tag{19.5}$$

- Eqn. (19.3) follows since $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$, and Eqn. (19.4) follows since $B \cap \bar{A} = (A \cup B) \cap \bar{A}$ and $f(A \cup B) + f(\bar{A}) \geq f((A \cup B) \cup \bar{A}) + f((A \cup B) \cap \bar{A})$.
- Hence, both $f(B) \geq f(B \cap A) + f(B \cap \bar{A})$ (from above) and $f(B) \leq f(B \cap A) + f(B \cap \bar{A})$ (previous slide).

□

Separators of submodular function via symmetrized version

- Again, let $\breve{f}$ be the symmetrized version of f.

Separators of submodular function via symmetrized version

- Again, let $\breve{f}$ be the symmetrized version of f.
- Definition: If $\breve{f}(A) = 0$, then any $A' \subseteq A$ and $\bar{A}' \subseteq E \setminus A$ are "independent" w.r.t. submodular g, and A is called a separator.

Separators of submodular function via symmetrized version

- Again, let $\breve{f}$ be the symmetrized version of f.
- Definition: If $\breve{f}(A) = 0$, then any $A' \subseteq A$ and $\bar{A}' \subseteq E \setminus A$ are "independent" w.r.t. submodular g, and A is called a separator.
- random variables: $X_A \perp\!\!\!\perp X_B \Rightarrow X_{A'} \perp\!\!\!\perp X_{B'}\ \forall\ A' \subseteq A$ and $B' \subseteq B$.

Separators of submodular function via symmetrized version

- Again, let $\breve{f}$ be the symmetrized version of f.
- Definition: If $\breve{f}(A) = 0$, then any $A' \subseteq A$ and $\bar{A}' \subseteq E \setminus A$ are "independent" w.r.t. submodular g, and A is called a separator.
- random variables: $X_A \perp\!\!\!\perp X_B \Rightarrow X_{A'} \perp\!\!\!\perp X_{B'}\ \forall\ A' \subseteq A$ and $B' \subseteq B$.
- Set of separators of $\breve{f}$ is closed under intersection, union, and complementation. Hence, the separators partition E.

Separators of submodular function via symmetrized version

- Again, let $\breve{f}$ be the symmetrized version of f.
- Definition: If $\breve{f}(A) = 0$, then any $A' \subseteq A$ and $\bar{A}' \subseteq E \setminus A$ are "independent" w.r.t. submodular g, and A is called a separator.
- random variables: $X_A \perp\!\!\!\perp X_B \Rightarrow X_{A'} \perp\!\!\!\perp X_{B'}\ \forall\ A' \subseteq A$ and $B' \subseteq B$.
- Set of separators of $\breve{f}$ is closed under intersection, union, and complementation. Hence, the separators partition E.
- In following slides, $\breve{f}$ is symmetrized & normalized version of f.

Review

Next slide is from Lecture 4.

Many (Equivalent) Definitions of Submodularity

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B), \quad \forall A, B \subseteq V \tag{19.6}$$

$$f(j|S) \geq f(j|T), \; \forall S \subseteq T \subseteq V, \text{ with } j \in V \setminus T \tag{19.7}$$

$$f(C|S) \geq f(C|T), \forall S \subseteq T \subseteq V, \text{ with } C \subseteq V \setminus T \tag{19.8}$$

$$f(j|S) \geq f(j|S \cup \{k\}), \; \forall S \subseteq V \text{ with } j \in V \setminus (S \cup \{k\}) \tag{19.9}$$

$$f(A \cup B|A \cap B) \leq f(A|A \cap B) + f(B|A \cap B), \quad \forall A, B \subseteq V \tag{19.10}$$

$$f(T) \leq f(S) + \sum_{j \in T \setminus S} f(j|S) - \sum_{j \in S \setminus T} f(j|S \cup T - \{j\}), \; \forall S, T \subseteq V \tag{19.11}$$

$$f(T) \leq f(S) + \sum_{j \in T \setminus S} f(j|S), \; \forall S \subseteq T \subseteq V \tag{19.12}$$

$$f(T) \leq f(S) - \sum_{j \in S \setminus T} f(j|S \setminus \{j\}) + \sum_{j \in T \setminus S} f(j|S \cap T) \; \forall S, T \subseteq V \tag{19.13}$$

$$f(T) \leq f(S) - \sum_{j \in S \setminus T} f(j|S \setminus \{j\}), \; \forall T \subseteq S \subseteq V \tag{19.14}$$

Minimization of a Symmetric Submodular Functions

- Minimizing symmetric submodular functions can be done in strongly polynomial time $O(n^3)$. The algorithm by Nagamochi & Ibaracki 1992 for graph cuts shown by Queyranne in 1995 to work for sym. SFM.

Minimization of a Symmetric Submodular Functions

- Minimizing symmetric submodular functions can be done in strongly polynomial time $O(n^3)$. The algorithm by Nagamochi & Ibaracki 1992 for graph cuts shown by Queyranne in 1995 to work for sym. SFM.
- The algorithm finds (as a subroutine) MA (maximum adjacency) or a maximum back orders (not same as greedy order).

```
Choose v_1 arbitrarily ;
W_1 ← (v_1)                  /* The first of an ordered list W_i. */ ;
for i ← 1 ... |V| − 1 do
    Choose v_{i+1} ∈ argmin_{u∈V\W_i} f(W_i|{u}) ;
    W_{i+1} ← (W_i, v_{i+1}) ; /* Append v_{i+1} to end of W_i */
```

Minimization of a Symmetric Submodular Functions

- Minimizing symmetric submodular functions can be done in strongly polynomial time $O(n^3)$. The algorithm by Nagamochi & Ibaracki 1992 for graph cuts shown by Queyranne in 1995 to work for sym. SFM.
- The algorithm finds (as a subroutine) MA (maximum adjacency) or a maximum back orders (not same as greedy order).

Choose v_1 arbitrarily ;
$W_1 \leftarrow (v_1)$ /* The first of an ordered list W_i. */ ;
for $i \leftarrow 1 \ldots |V| - 1$ **do**
  Choose $v_{i+1} \in \text{argmin}_{u \in V \setminus W_i} f(W_i | \{u\})$;
  $W_{i+1} \leftarrow (W_i, v_{i+1})$; /* Append v_{i+1} to end of W_i */

- Note algorithm operates on non-symmetric function f. If f is already symmetric and normalized, then $f = \breve{f}$.

Minimization of a Symmetric Submodular Functions

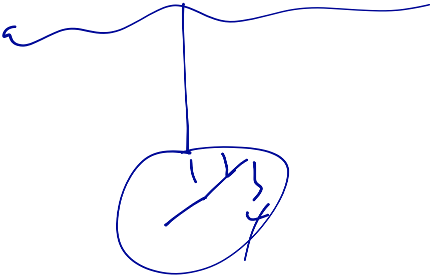
- Minimizing symmetric submodular functions can be done in strongly polynomial time $O(n^3)$. The algorithm by Nagamochi & Ibaracki 1992 for graph cuts shown by Queyranne in 1995 to work for sym. SFM.
- The algorithm finds (as a subroutine) MA (maximum adjacency) or a maximum back orders (not same as greedy order).

```
Choose v_1 arbitrarily ;
W_1 ← (v_1)                    /* The first of an ordered list W_i. */ ;
for i ← 1 ... |V| − 1 do
    Choose v_{i+1} ∈ argmin_{u ∈ V \ W_i} f(W_i | {u}) ;
    W_{i+1} ← (W_i, v_{i+1}) ; /* Append v_{i+1} to end of W_i */
```

- Note algorithm operates on non-symmetric function f. If f is already symmetric and normalized, then $f = \breve{f}$.
- The final ordered set $W_n = (v_1, v_2, \dots, v_n)$ is special in that the last two nodes (v_{n-1}, v_n) serve as a surrogate minimizer for a special case.

Pendent pair

- A ordered pair of elements (t, u) is called a pendent pair if u is a minimizer amongst all sets that separate u and t.



Pendent pair

- A ordered pair of elements (t, u) is called a pendent pair if u is a minimizer amongst all sets that separate u and t.
- That is (t, u) is a pendent pair if

$$\{u\} \in \operatorname*{argmin}_{A \subseteq V} \left\{ \breve{f}(A) : u \in A, t \notin A \right\} \tag{19.6}$$

Pendent pair

- A ordered pair of elements (t, u) is called a pendent pair if u is a minimizer amongst all sets that separate u and t.
- That is (t, u) is a pendent pair if

$$\{u\} \in \operatorname{argmin}\left\{\breve{f}(A) : u \in A, t \notin A\right\} \tag{19.6}$$

- That is,

$$\breve{f}(\{u\}) \leq \breve{f}(A) \ \ \forall A \text{ s.t. } t \notin A \ni u \tag{19.7}$$

Pendent pair

- A ordered pair of elements (t, u) is called a pendent pair if u is a minimizer amongst all sets that separate u and t.
- That is (t, u) is a pendent pair if

$$\{u\} \in \operatorname{argmin} \left\{ \breve{f}(A) : u \in A, t \notin A \right\} \tag{19.6}$$

- That is,

$$\breve{f}(\{u\}) \leq \breve{f}(A) \;\; \forall A \text{ s.t. } t \notin A \ni u \tag{19.7}$$

Theorem 19.3.2

In the ordered set $W = (v_1, \ldots, v_n)$ generated by the MA algorithm, then (v_{n-1}, v_n) is a pendent pair.

Pendent pair

- A ordered pair of elements (t, u) is called a pendent pair if u is a minimizer amongst all sets that separate u and t.
- That is (t, u) is a pendent pair if

$$\{u\} \in \operatorname{argmin} \left\{ \breve{f}(A) : u \in A, t \notin A \right\} \tag{19.6}$$

- That is,

$$\breve{f}(\{u\}) \leq \breve{f}(A) \;\; \forall A \text{ s.t. } t \notin A \ni u \tag{19.7}$$

Theorem 19.3.2

In the ordered set $W = (v_1, \ldots, v_n)$ generated by the MA algorithm, then (v_{n-1}, v_n) is a pendent pair.

- Interestingly, this algorithm is the same as maximum cardinality search (MCS), when f represents a graph cut function (recall, MCS is used to efficiently test graph chordality).

Minimization of a Symmetric Submodular Functions

- Now, given a pendent pair (t, u) there are two cases.

Minimization of a Symmetric Submodular Functions

- Now, given a pendent pair (t, u) there are two cases.
- Either: The global minimizer, say X^* of $\breve{f}$ is such that $t \notin X^* \ni u$ or we, by symmetry, can w.l.o.g. choose the minimizer so that both $\{t, u\} \in X^*$.

Minimization of a Symmetric Submodular Functions

- Now, given a pendent pair (t, u) there are two cases.
- Either: The global minimizer, say X^* of $\breve{f}$ is such that $t \notin X^* \ni u$ or we, by symmetry, can w.l.o.g. choose the minimizer so that both $\{t, u\} \in X^*$.
- We store the score (min value) in the first case,

Minimization of a Symmetric Submodular Functions

- Now, given a pendent pair (t, u) there are two cases.
- Either: The global minimizer, say X^* of $\breve{f}$ is such that $t \notin X^* \ni u$ or we, by symmetry, can w.l.o.g. choose the minimizer so that both $\{t, u\} \in X^*$.
- We store the score (min value) in the first case, then, consider a new element "tu" and clustered ground set $V' = V \setminus \{t, u\} \cup \{tu\}$, and new symmetric submodular function $f' : 2^{V'} \to \mathbb{R}$ with

$$\breve{f}'(X) = \begin{cases} \breve{f}(X) & \text{if } tu \notin X \\ \breve{f}(X \cup \{t, u\} \setminus \{tu\}) & \text{if } tu \in X \end{cases} \tag{19.8}$$

Minimization of a Symmetric Submodular Functions

- Now, given a pendent pair (t, u) there are two cases.
- Either: The global minimizer, say X^* of $\breve{f}$ is such that $t \notin X^* \ni u$ or we, by symmetry, can w.l.o.g. choose the minimizer so that both $\{t, u\} \in X^*$.
- We store the score (min value) in the first case, then, consider a new element "tu" and clustered ground set $V' = V \setminus \{t, u\} \cup \{tu\}$, and new symmetric submodular function $f' : 2^{V'} \to \mathbb{R}$ with

$$\breve{f}'(X) = \begin{cases} \breve{f}(X) & \text{if } tu \notin X \\ \breve{f}(X \cup \{t, u\} \setminus \{tu\}) & \text{if } tu \in X \end{cases} \tag{19.8}$$

- We then find a new pendent pair on f' using the above algorithm, store the new min value, and merge, and repeat.

Minimization of a Symmetric Submodular Functions

- Now, given a pendent pair (t, u) there are two cases.
- Either: The global minimizer, say X^* of $\breve{f}$ is such that $t \notin X^* \ni u$ or we, by symmetry, can w.l.o.g. choose the minimizer so that both $\{t, u\} \in X^*$.
- We store the score (min value) in the first case, then, consider a new element "tu" and clustered ground set $V' = V \setminus \{t, u\} \cup \{tu\}$, and new symmetric submodular function $f' : 2^{V'} \to \mathbb{R}$ with

$$\breve{f}'(X) = \begin{cases} \breve{f}(X) & \text{if } tu \notin X \\ \breve{f}(X \cup \{t, u\} \setminus \{tu\}) & \text{if } tu \in X \end{cases} \tag{19.8}$$

- We then find a new pendent pair on f' using the above algorithm, store the new min value, and merge, and repeat.
- We do this n times. We take the min over all of the stored values.

Minimization of a Symmetric Submodular Functions

- Now, given a pendent pair (t, u) there are two cases.
- Either: The global minimizer, say X^* of $\breve{f}$ is such that $t \notin X^* \ni u$ or we, by symmetry, can w.l.o.g. choose the minimizer so that both $\{t, u\} \in X^*$.
- We store the score (min value) in the first case, then, consider a new element "tu" and clustered ground set $V' = V \setminus \{t, u\} \cup \{tu\}$, and new symmetric submodular function $f' : 2^{V'} \to \mathbb{R}$ with

$$\breve{f}'(X) = \begin{cases} \breve{f}(X) & \text{if } tu \notin X \\ \breve{f}(X \cup \{t, u\} \setminus \{tu\}) & \text{if } tu \in X \end{cases} \tag{19.8}$$

- We then find a new pendent pair on f' using the above algorithm, store the new min value, and merge, and repeat.
- We do this n times. We take the min over all of the stored values.
- The pendent pair corresponding to the min element, say (t', u') will (most probability) correspond to nested clusters, so we use the original ground elements corresponding to u'.

Minimization of a Symmetric Submodular Functions

Theorem 19.3.3

The final resultant u' when expanded to original ground elements minimizes the symmetric submodular function f in $O(n^3)$ time.

- This has become known as Queyranne's algorithm for symmetric submodular function minimization.
- This was done in 1995 and it is said that this result, at that time, rekindled the efforts to find general combinatorial SFM.
- The actual algorithm was originally developed by Nagamochi and Ibaraki for a simple algorithm for finding graph cut. Queyranne showed it worked for any symmetric submodular function.
- Hence, it seems reasonable that symmetric SFM is faster than general SFM (although this question is still unknown).
- Quoting Fujishige from NIPS 2012, he said that he "hopes general purpose SFM is $O(n^4)$" ☺.

Maximization of Submodular Functions

- We spent much time on submodular function minimization (SFM) and saw this can be done in polynomial time.

Maximization of Submodular Functions

- We spent much time on submodular function minimization (SFM) and saw this can be done in polynomial time.
- Submodular maximization is also quite useful.

Maximization of Submodular Functions

- We spent much time on submodular function minimization (SFM) and saw this can be done in polynomial time.
- Submodular maximization is also quite useful.
- Applications: sensor placement, facility location, document summarization, or any kind of covering problem (choose a small set of elements that cover some domain as much as possible).

Maximization of Submodular Functions

- We spent much time on submodular function minimization (SFM) and saw this can be done in polynomial time.
- Submodular maximization is also quite useful.
- Applications: sensor placement, facility location, document summarization, or any kind of covering problem (choose a small set of elements that cover some domain as much as possible).
- For polymatroid function (or any monotone non-decreasing function), unconstrained maximization is trivial (take ground set).

Maximization of Submodular Functions

- We spent much time on submodular function minimization (SFM) and saw this can be done in polynomial time.
- Submodular maximization is also quite useful.
- Applications: sensor placement, facility location, document summarization, or any kind of covering problem (choose a small set of elements that cover some domain as much as possible).
- For polymatroid function (or any monotone non-decreasing function), unconstrained maximization is trivial (take ground set).
- Thus, when we do monotone submodular maximization, we either

Maximization of Submodular Functions

- We spent much time on submodular function minimization (SFM) and saw this can be done in polynomial time.
- Submodular maximization is also quite useful.
- Applications: sensor placement, facility location, document summarization, or any kind of covering problem (choose a small set of elements that cover some domain as much as possible).
- For polymatroid function (or any monotone non-decreasing function), unconstrained maximization is trivial (take ground set).
- Thus, when we do monotone submodular maximization, we either
  - Find the maximum under some constraint

Maximization of Submodular Functions

- We spent much time on submodular function minimization (SFM) and saw this can be done in polynomial time.
- Submodular maximization is also quite useful.
- Applications: sensor placement, facility location, document summarization, or any kind of covering problem (choose a small set of elements that cover some domain as much as possible).
- For polymatroid function (or any monotone non-decreasing function), unconstrained maximization is trivial (take ground set).
- Thus, when we do monotone submodular maximization, we either
  - Find the maximum under some constraint
  - Find the maximum for a non-polymatroid submodular function

Maximization of Submodular Functions

- We spent much time on submodular function minimization (SFM) and saw this can be done in polynomial time.
- Submodular maximization is also quite useful.
- Applications: sensor placement, facility location, document summarization, or any kind of covering problem (choose a small set of elements that cover some domain as much as possible).
- For polymatroid function (or any monotone non-decreasing function), unconstrained maximization is trivial (take ground set).
- Thus, when we do monotone submodular maximization, we either
  - Find the maximum under some constraint
  - Find the maximum for a non-polymatroid submodular function
  - Do both.

Maximization of Submodular Functions

- We spent much time on submodular function minimization (SFM) and saw this can be done in polynomial time.
- Submodular maximization is also quite useful.
- Applications: sensor placement, facility location, document summarization, or any kind of covering problem (choose a small set of elements that cover some domain as much as possible).
- For polymatroid function (or any monotone non-decreasing function), unconstrained maximization is trivial (take ground set).
- Thus, when we do monotone submodular maximization, we either
  - Find the maximum under some constraint
  - Find the maximum for a non-polymatroid submodular function
  - Do both.
- There is also a sort of dual problem that is often considered together with max, and those are minimum cover problems (to be defined).

The Set Cover Problem

- Let E be a ground set and let $E_1, E_2, \ldots, E_m$ be a set of subsets.

The Set Cover Problem

- Let E be a ground set and let $E_1, E_2, \ldots, E_m$ be a set of subsets.
- Let $V = \{1, 2, \ldots, m\}$ be the set of integers.

The Set Cover Problem

- Let E be a ground set and let $E_1, E_2, \ldots, E_m$ be a set of subsets.
- Let $V = \{1, 2, \ldots, m\}$ be the set of integers.
- Define $f : 2^V \to \mathbb{Z}_+$ as $f(X) = |\bigcup_{v \in X} E_v|$

The Set Cover Problem

- Let E be a ground set and let $E_1, E_2, \dots, E_m$ be a set of subsets.
- Let $V = \{1, 2, \dots, m\}$ be the set of integers.
- Define $f : 2^V \to \mathbb{Z}_+$ as $f(X) = |\bigcup_{v \in X} E_v|$
- Then f is the set cover function. As we say, f is monotone submodular (a polymatroid).

The Set Cover Problem

- Let E be a ground set and let $E_1, E_2, \ldots, E_m$ be a set of subsets.
- Let $V = \{1, 2, \ldots, m\}$ be the set of integers.
- Define $f : 2^V \to \mathbb{Z}_+$ as $f(X) = |\bigcup_{v \in X} E_v|$
- Then f is the set cover function. As we say, f is monotone submodular (a polymatroid).
- The set cover problem asks for the smallest subset X of V such that $f(X) = |E|$ (smallest subset of the subsets of E) where E is still covered. I.e.,

$$\text{minimize} |X| \text{ subject to } f(X) \geq |E| \tag{19.9}$$

The Set Cover Problem

- Let E be a ground set and let $E_1, E_2, \ldots, E_m$ be a set of subsets.
- Let $V = \{1, 2, \ldots, m\}$ be the set of integers.
- Define $f : 2^V \to \mathbb{Z}_+$ as $f(X) = |\bigcup_{v \in X} E_v|$
- Then f is the set cover function. As we say, f is monotone submodular (a polymatroid).
- The set cover problem asks for the smallest subset X of V such that $f(X) = |E|$ (smallest subset of the subsets of E) where E is still covered. I.e.,

$$\text{minimize}|X| \text{ subject to } f(X) \geq |E| \tag{19.9}$$

- We might wish to use a more general modular function $m(X)$ rather than cardinality $|X|$.

The Set Cover Problem

- Let E be a ground set and let $E_1, E_2, \ldots, E_m$ be a set of subsets.
- Let $V = \{1, 2, \ldots, m\}$ be the set of integers.
- Define $f : 2^V \to \mathbb{Z}_+$ as $f(X) = |\bigcup_{v \in X} E_v|$
- Then f is the set cover function. As we say, f is monotone submodular (a polymatroid).
- The set cover problem asks for the smallest subset X of V such that $f(X) = |E|$ (smallest subset of the subsets of E) where E is still covered. I.e.,

$$\text{minimize}|X| \text{ subject to } f(X) \geq |E| \tag{19.9}$$

- We might wish to use a more general modular function $m(X)$ rather than cardinality $|X|$.
- This problem is NP-hard, and Feige in 1998 showed that it cannot be approximated with a ratio better than $(1 - \epsilon) \log n$ unless NP is slightly superpolynomial ($n^{O(\log \log n)}$).

What About Non-monotone

- So even simple case of cardinality constrained submodular function maximization is NP-hard.
- This will be true of most submodular max (and related) problems.
- Hence, the only hope is approximation algorithms. Question is, what is the tradeoff between running time and approximation quality, and is it possible to get tight bounds (i.e., an algorithm that achieves an approximation ratio, and a proof that one can't do better than that unless some extremely unlike event were to be true, such as P=NP).

The Max k-Cover Problem

- Let E be a ground set and let $E_1, E_2, \ldots, E_m$ be a set of subsets.

The Max k-Cover Problem

- Let E be a ground set and let $E_1, E_2, \ldots, E_m$ be a set of subsets.
- Let $V = \{1, 2, \ldots, m\}$ be the set of integers.

The Max k-Cover Problem

- Let E be a ground set and let $E_1, E_2, \dots, E_m$ be a set of subsets.
- Let $V = \{1, 2, \dots, m\}$ be the set of integers.
- Define $f : 2^V \to \mathbb{Z}_+$ as $f(X) = |\bigcup_{v \in V} E_v|$

The Max k-Cover Problem

- Let E be a ground set and let $E_1, E_2, \ldots, E_m$ be a set of subsets.
- Let $V = \{1, 2, \ldots, m\}$ be the set of integers.
- Define $f : 2^V \to \mathbb{Z}_+$ as $f(X) = |\bigcup_{v \in V} E_v|$
- Then f is the set cover function. As we saw, f is monotone submodular (a polymatroid).

The Max k-Cover Problem

- Let E be a ground set and let $E_1, E_2, \ldots, E_m$ be a set of subsets.
- Let $V = \{1, 2, \ldots, m\}$ be the set of integers.
- Define $f : 2^V \to \mathbb{Z}_+$ as $f(X) = |\bigcup_{v \in V} E_v|$
- Then f is the set cover function. As we saw, f is monotone submodular (a polymatroid).
- The max k cover problem asks, given a k, what sized k set of sets X can we choose that covers the most? I.e., that maximizes $f(X)$ as in:

$$\max f(X) \text{ subject to } |X| \leq k \tag{19.10}$$

The Max k-Cover Problem

- Let E be a ground set and let $E_1, E_2, \ldots, E_m$ be a set of subsets.
- Let $V = \{1, 2, \ldots, m\}$ be the set of integers.
- Define $f : 2^V \to \mathbb{Z}_+$ as $f(X) = |\bigcup_{v \in V} E_v|$
- Then f is the set cover function. As we saw, f is monotone submodular (a polymatroid).
- The max k cover problem asks, given a k, what sized k set of sets X can we choose that covers the most? I.e., that maximizes $f(X)$ as in:

$$\max f(X) \text{ subject to } |X| \leq k \tag{19.10}$$

- This problem is NP-hard, and Feige in 1998 showed that it cannot be approximated with a ratio better than $(1 - 1/e)$.

Cardinality Constrained Max. of Polymatroid Functions

- Now we are given an arbitrary polymatroid function f.

Cardinality Constrained Max. of Polymatroid Functions

- Now we are given an arbitrary polymatroid function f.
- Given k, goal is: find $A^* \in \operatorname{argmax}\{f(A) : |A| \leq k\}$

Cardinality Constrained Max. of Polymatroid Functions

- Now we are given an arbitrary polymatroid function f.
- Given k, goal is: find $A^* \in \operatorname{argmax} \{f(A) : |A| \leq k\}$
- w.l.o.g., we can find $A^* \in \operatorname{argmax} \{f(A) : |A| = k\}$

Cardinality Constrained Max. of Polymatroid Functions

- Now we are given an arbitrary polymatroid function f.
- Given k, goal is: find $A^* \in \operatorname{argmax} \{f(A) : |A| \leq k\}$
- w.l.o.g., we can find $A^* \in \operatorname{argmax} \{f(A) : |A| = k\}$
- An important result by Nemhauser et. al. (1978) states that for normalized ($f(\emptyset) = 0$) monotone submodular functions (i.e., polymatroids) can be approximately maximized using a simple greedy algorithm.

Cardinality Constrained Max. of Polymatroid Functions

- Now we are given an arbitrary polymatroid function f.
- Given k, goal is: find $A^* \in \operatorname{argmax}\{f(A) : |A| \leq k\}$
- w.l.o.g., we can find $A^* \in \operatorname{argmax}\{f(A) : |A| = k\}$
- An important result by Nemhauser et. al. (1978) states that for normalized ($f(\emptyset) = 0$) monotone submodular functions (i.e., polymatroids) can be approximately maximized using a simple greedy algorithm.
- Starting with $S_0 = \emptyset$, we repeat the following greedy step for $i = 0 \dots (k-1)$:

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i} f(S_i \cup \{v\}) \right\} \tag{19.11}$$

The Greedy Algorithm for Submodular Max

A bit more precisely:

Algorithm 2: The Greedy Algorithm

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

The Greedy Algorithm for Submodular Max

- This algorithm has a guarantee

The Greedy Algorithm for Submodular Max

- This algorithm has a guarantee

Theorem 19.4.1

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

0.63

The Greedy Algorithm for Submodular Max

- This algorithm has a guarantee

Theorem 19.4.1

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 find $A^* \in \operatorname{argmax} \{f(A) : |A| \leq k\}$, we repeat the greedy step until $k = i + 1$:

The Greedy Algorithm for Submodular Max

- This algorithm has a guarantee

Theorem 19.4.1

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 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 +1 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 OPT $= f(S^*)$. By submodularity, we will show:

$$\exists v \in S^* \setminus S_i : f(S_i + v|S_i) \geq \frac{1}{k}(\text{OPT} - f(S_i)) \tag{19.12}$$

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

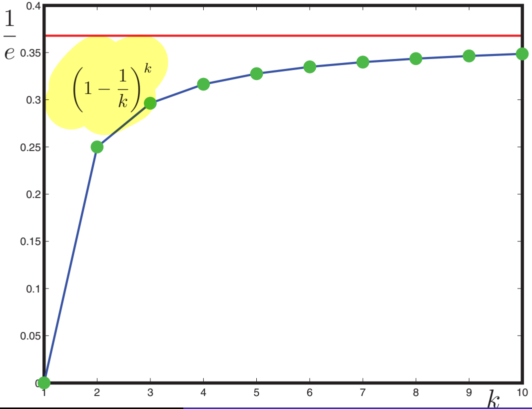
- At step $i < k$, greedy chooses v_i to maximize $f(v|S_i)$.
- Let S^* be optimal solution (of size k) and OPT $= f(S^*)$. By submodularity, we will show:

$$\exists v \in S^* \setminus S_i : f(S_i + v|S_i) \geq \frac{1}{k}(\text{OPT} - f(S_i)) \tag{19.12}$$

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:

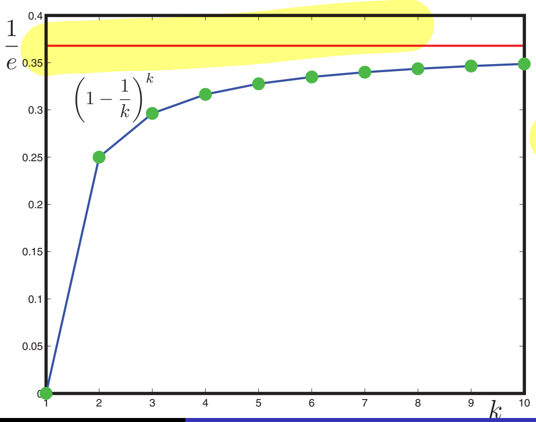
$$\exists v \in S^* \setminus S_i : f(S_i + v|S_i) \geq \frac{1}{k}(\text{OPT} - f(S_i)) \tag{19.12}$$



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:

$$\exists v \in S^* \setminus S_i : f(S_i + v|S_i) \geq \frac{1}{k}(\mathsf{OPT} - f(S_i)) \tag{19.12}$$



Equation (19.21) will show that: (19.12) $\Rightarrow$

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

Cardinality Constrained Polymatroid Max Theorem

Theorem 19.4.2 (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. (19.11)). Then for all $k, \ell \in \mathbb{Z}_{++}$, we have:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) \max_{S:|S| \leq k} f(S) \tag{19.13}$$

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 19.4.2 (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. (19.11)). Then for all $k, \ell \in \mathbb{Z}_{++}$, we have:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) \max_{S:|S| \leq k} f(S) \tag{19.13}$$

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 19.4.2 (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. (19.11)). Then for all $k, \ell \in \mathbb{Z}_{++}$, we have:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) \max_{S:|S| \leq k} f(S) \tag{19.13}$$

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 19.4.2 (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. (19.11)). Then for all $k, \ell \in \mathbb{Z}_{++}$, we have:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) \max_{S:|S| \leq k} f(S) \tag{19.13}$$

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 19.4.2 (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. (19.11)). Then for all $k, \ell \in \mathbb{Z}_{++}$, we have:

$$f(S_\ell) \geq (1 - e^{-\ell/k}) \max_{S:|S| \leq k} f(S) \tag{19.13}$$

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 19.4.2.

...

Cardinality Constrained Polymatroid Max Theorem

Proof of Theorem 19.4.2.

- 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 19.4.2.

- 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 19.4.2.

- 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 19.4.2.

- 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.

...

Cardinality Constrained Polymatroid Max Theorem

Proof of Theorem 19.4.2.

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

$= S_\ell$

...

Cardinality Constrained Polymatroid Max Theorem

Proof of Theorem 19.4.2.

- 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 $(v_1, v_2, \ldots, v_k)$ be the greedy order chosen by the algorithm.
- Then the following inequalities (on the next slide) follow:

. . .

Cardinality Constrained Polymatroid Max Theorem

... proof of Theorem 19.4.2 cont.

...

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$f(S^*)$$

. . .

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$f(S^*) \leq f(S^* \cup S_i) \tag{19.14}$$

. . .

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$f(S^*) \leq f(S^* \cup S_i) \tag{19.14}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^* | S_i \cup \{v_1^*, v_2^*, \dots, v_{j-1}^*\}) \tag{19.15}$$

. . .

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$f(S^*) \leq f(S^* \cup S_i) \tag{19.14}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^* | S_i \cup \{v_1^*, v_2^*, \ldots, v_{j-1}^*\}) \tag{19.15}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v | S_i) \tag{19.16}$$

. . .

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$f(S^*) \leq f(S^* \cup S_i) \tag{19.14}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^* | S_i \cup \{v_1^*, v_2^*, \dots, v_{j-1}^*\}) \tag{19.15}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v|S_i) \tag{19.16}$$

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

. . .

Cardinality Constrained Polymatroid Max Theorem

... proof of Theorem 19.4.2 cont.

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

$$f(S^*) \leq f(S^* \cup S_i) \tag{19.14}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^* | S_i \cup \{v_1^*, v_2^*, \ldots, v_{j-1}^*\}) \tag{19.15}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v|S_i) \tag{19.16}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v_{i+1}|S_i) = f(S_i) + \sum_{v \in S^*} f(S_{i+1}|S_i) \tag{19.17}$$

...

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$f(S^*) \leq f(S^* \cup S_i) \tag{19.14}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^* | S_i \cup \{v_1^*, v_2^*, \dots, v_{j-1}^*\}) \tag{19.15}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v|S_i) \tag{19.16}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v_{i+1}|S_i) = f(S_i) + \sum_{v \in S^*} f(S_{i+1}|S_i) \tag{19.17}$$

$$= f(S_i) + kf(S_{i+1}|S_i) \tag{19.18}$$

. . .

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$f(S^*) \leq f(S^* \cup S_i) \tag{19.14}$$

$$= f(S_i) + \sum_{j=1}^{k} f(v_j^* | S_i \cup \{v_1^*, v_2^*, \dots, v_{j-1}^*\}) \tag{19.15}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v | S_i) \tag{19.16}$$

$$\leq f(S_i) + \sum_{v \in S^*} f(v_{i+1} | S_i) = f(S_i) + \sum_{v \in S^*} f(S_{i+1} | S_i) \tag{19.17}$$

$$= f(S_i) + k f(S_{i+1} | S_i) \tag{19.18}$$

- Therefore, we have Equation 19.12, i.e.,:

$$f(S^*) - f(S_i) \leq k f(S_{i+1} | S_i) = k(f(S_{i+1}) - f(S_i)) \tag{19.19}$$

δ_i $\leq k(\delta_i - \delta_{i+1})$

. . .

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

Cardinality Constrained Polymatroid Max Theorem

... proof of Theorem 19.4.2 cont.

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

Cardinality Constrained Polymatroid Max Theorem

... proof of Theorem 19.4.2 cont.

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

$$\delta_i \leq k(\delta_i - \delta_{i+1}) \tag{19.20}$$

or

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$\delta_i \leq k(\delta_i - \delta_{i+1}) \tag{19.20}$$

or

$$\delta_{i+1} \leq (1 - \frac{1}{k})\delta_i \tag{19.21}$$

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$\delta_i \leq k(\delta_i - \delta_{i+1}) \tag{19.20}$$

or

$$\delta_{i+1} \leq (1 - \frac{1}{k})\delta_i \tag{19.21}$$

- The relationship between δ_0 and δ_ℓ is then

$$\delta_l \leq (1 - \frac{1}{k})^\ell \delta_0 \tag{19.22}$$

Cardinality Constrained Polymatroid Max Theorem

... proof of Theorem 19.4.2 cont.

- Define $\delta_i \triangleq f(S^*) - f(S_i)$, so $\delta_i - \delta_{i+1} = f(S_{i+1}) - f(S_i)$, giving
$$\delta_i \leq k(\delta_i - \delta_{i+1}) \tag{19.20}$$
or
$$\delta_{i+1} \leq (1 - \frac{1}{k})\delta_i \tag{19.21}$$
- The relationship between δ_0 and δ_ℓ is then
$$\delta_l \leq (1 - \frac{1}{k})^\ell \delta_0 \tag{19.22}$$
- Now, $\delta_0 = f(S^*) - f(\emptyset) \leq f(S^*)$ since $f \geq 0$.

Cardinality Constrained Polymatroid Max Theorem

... proof of Theorem 19.4.2 cont.

- Define $\delta_i \triangleq f(S^*) - f(S_i)$, so $\delta_i - \delta_{i+1} = f(S_{i+1}) - f(S_i)$, giving
$$\delta_i \leq k(\delta_i - \delta_{i+1}) \tag{19.20}$$
or
$$\delta_{i+1} \leq (1 - \frac{1}{k})\delta_i \tag{19.21}$$
- The relationship between δ_0 and δ_ℓ is then
$$\delta_l \leq (1 - \frac{1}{k})^\ell \delta_0 \tag{19.22}$$
$x = 1/k$
- 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
$$\delta_\ell \leq (1 - \frac{1}{k})^\ell \delta_0 \leq e^{-\ell/k} f(S^*) \tag{19.23}$$

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

□

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \tag{19.24}$$

□

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \tag{19.24}$$

$\square$

- 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, $f(S_k) \geq (1 - 1/e) f(S^*) \approx 0.6321 f(S^*)$.

Cardinality Constrained Polymatroid Max Theorem

. . . proof of Theorem 19.4.2 cont.

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

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \quad (19.24)$$

$\square$

- 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 19.4.2 cont.

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

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \tag{19.24}$$

□

- 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 19.4.2 cont.

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

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \tag{19.24}$$

□

- 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 19.4.2 cont.

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

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*) \tag{19.24}$$

□

- 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 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 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 (typically $n \log n$) 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 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 (typically $n \log n$) 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

$$f(v'|S_{i+1}) \geq \alpha_v = f(v|S_i) \geq f(v|S_{i+1}) \tag{19.25}$$

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

$$f(v'|S_{i=1}) \geq \alpha_v = f(v|S_i) \geq f(v|S_{i+1}) \tag{19.25}$$

  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, and repeat.

Minoux's Accelerated Greedy for Submodular Functions

- Minoux's algorithm is exact, in that it has the same guarantees as does the $O(n^2)$ greedy Algorithm 2 (this means it will return either the same answers, or answers that have 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 $O(n^2)$ greedy Algorithm 2 (this means it will return either the same answers, or answers that have 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 $O(n^2)$ greedy Algorithm 2 (this means it will return either the same answers, or answers that have 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 $O(n^2)$ greedy Algorithm 2 (this means it will return either the same answers, or answers that have 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 $O(n^2)$ greedy Algorithm 2 (this means it will return either the same answers, or answers that have 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 $\alpha \in \mathbb{R}$.

$$\text{INSERT}(Q, (v, \alpha)) \tag{19.26}$$

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 $\alpha \in \mathbb{R}$.

$$\textsc{insert}(Q, (v, \alpha)) \tag{19.26}$$

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

$$(v, \alpha) \leftarrow \textsc{pop}(Q) \tag{19.27}$$

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 $\alpha \in \mathbb{R}$.

$$\text{INSERT}(Q, (v, \alpha)) \tag{19.26}$$

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

$$(v, \alpha) \leftarrow \text{POP}(Q) \tag{19.27}$$

  - Query the value of the max item in the queue

$$\text{MAX}(Q) \in \mathbb{R} \tag{19.28}$$

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 $\alpha \in \mathbb{R}$.
  $$\text{INSERT}(Q, (v, \alpha)) \tag{19.26}$$
  - Pop the item (v, α) with maximum value α off the queue.
  $$(v, \alpha) \leftarrow \text{POP}(Q) \tag{19.27}$$
  - Query the value of the max item in the queue
  $$\text{MAX}(Q) \in \mathbb{R} \tag{19.28}$$
- 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 $\alpha \in \mathbb{R}$.

$$\text{INSERT}(Q, (v, \alpha)) \tag{19.26}$$

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

$$(v, \alpha) \leftarrow \text{POP}(Q) \tag{19.27}$$

  - Query the value of the max item in the queue

$$\text{MAX}(Q) \in \mathbb{R} \tag{19.28}$$

- 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 3: Minoux's Accelerated Greedy Algorithm

1 Set $S_0 \leftarrow \emptyset$; $i \leftarrow 0$; Initialize priority queue Q ;
2 **for** $v \in E$ **do**
3     $\text{INSERT}(Q, f(v))$
4 **repeat**
5     $(v, \alpha) \leftarrow \text{POP}(Q)$;
6   **if** α *not "fresh"* **then**
7   recompute $\alpha \leftarrow f(v|S_i)$
8   **if** *(popped α in line 5 was "fresh") OR ($\alpha \geq \text{MAX}(Q)$)* **then**
9   Set $S_{i+1} \leftarrow S_i \cup \{v\}$;
10         $i \leftarrow i + 1$;
11   **else**
12         $\text{INSERT}(Q, (v, \alpha))$
13 **until** $i = |E|$;

Minimum Submodular Cover

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

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \tag{19.29}$$

where α is a “cover” requirement.

Minimum Submodular Cover

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

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \tag{19.29}$$

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:

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f'(S) \geq f'(V) \tag{19.30}$$

Minimum Submodular Cover

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

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \tag{19.29}$$

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:

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f'(S) \geq f'(V) \tag{19.30}$$

- 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 Cover

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

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \tag{19.29}$$

  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:

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f'(S) \geq f'(V) \tag{19.30}$$

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

Minimum Submodular Cover

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

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \tag{19.29}$$

  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:

$$S^* \in \operatorname*{argmin}_{S \subseteq V} |S| \text{ such that } f'(S) \geq f'(V) \tag{19.30}$$

- Note that this immediately generalizes standard set cover, in which case $f(A)$ is the cardinality of the union of sets indexed by A.
- Algorithm: Pick the first S_i chosen by aforementioned greedy algorithm such that $f(S_i) \geq \alpha$.
- For integer valued f, this greedy algorithm an $O(\log(\max_{s \in V} f(\{s\})))$ approximation. 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.

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 $\mathcal{I} = \{S \subseteq V : |S| \le k\}$, 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 $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \le k_i \text{ for all } i = 1, \ldots, \ell\}$., or a transversal, etc).
- Knapsack constraint: if each item $v \in V$ has a cost $c(v)$, we may ask for $c(S) \le b$ where b is a budget, in units of costs.

$$\mathcal{I} = \{ \underline{I} : c(I) \le b \}$$

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.
- 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 $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \leq k_i$ 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.
- 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

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^{p} \mathcal{I}_i} f(S_i \cup \{v\}) \right\} \tag{19.31}$$

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

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^p \mathcal{I}_i} f(S_i \cup \{v\}) \right\} \tag{19.31}$$

- 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

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^p \mathcal{I}_i}{\operatorname{argmax}} f(S_i \cup \{v\}) \right\} \tag{19.31}$$

- 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

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^p \mathcal{I}_i} f(S_i \cup \{v\}) \right\} \tag{19.31}$$

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

Theorem 19.5.1

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

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^p \mathcal{I}_i}{\operatorname{argmax}} f(S_i \cup \{v\}) \right\} \tag{19.31}$$

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

Theorem 19.5.1

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

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^{p} \mathcal{I}_i}{\operatorname{argmax}} f(S_i \cup \{v\}) \right\} \tag{19.31}$$

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

Theorem 19.5.1

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

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i \,:\, S_i + v \in \bigcap_{i=1}^p \mathcal{I}_i} f(S_i \cup \{v\}) \right\} \tag{19.31}$$

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

Theorem 19.5.1

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.

Review

- Recall bipartite matching via matroid intersection from Lecture 16.

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.

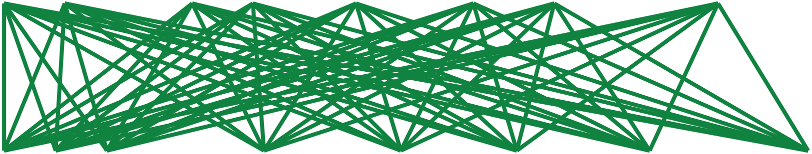
Greedy over multiple matroids: Generalized Bipartite Matching

- 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.

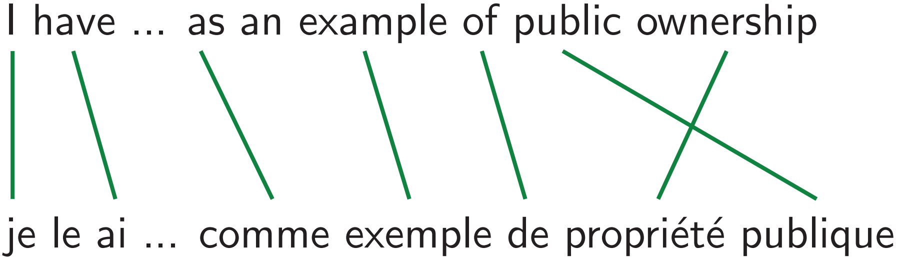
I have ... as an example of public ownership



je le ai ... comme exemple de propriété publique

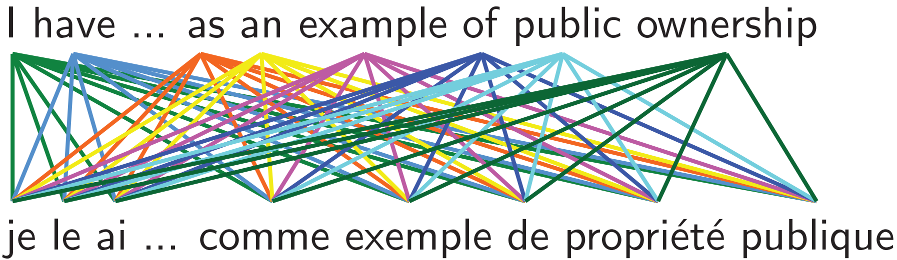
Greedy over > 1 matroids: Multiple Language Alignment

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



Greedy over > 1 matroids: Multiple Language Alignment

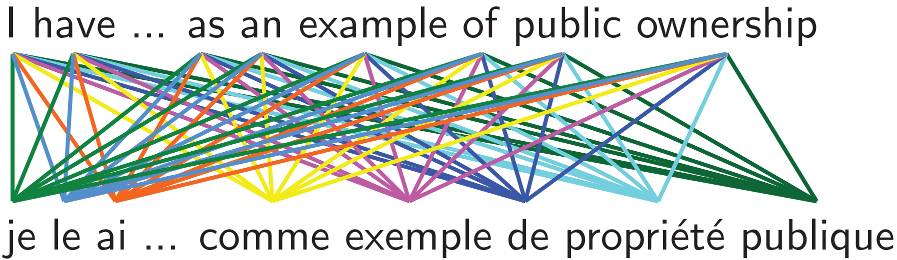
- Edges incident to English words constitute an edge partition



- 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



- 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 in lecture 16, 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 in lecture 16, 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

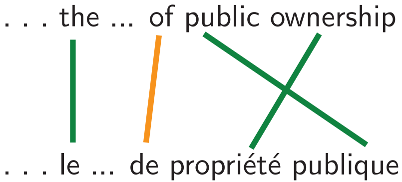
- Typical to use bipartite matching to find an alignment between the two language strings.
- As we saw in lecture 16, this is equivalent to two 1-partition matroids and a non-negative modular cost function on the edges.
- We can generalize this using a polymatroid cost function on the edges, and two k-partition matroids, allowing for "fertility" in the models:

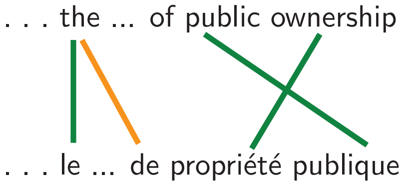
Fertility at most 1

Greedy over > 1 matroids: Multiple Language Alignment

- Typical to use bipartite matching to find an alignment between the two language strings.
- As we saw in lecture 16, 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:
  $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \leq k_i$ 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 $E = E_1 \cup E_2 \cup \cdots \cup E_n$ into n blocks in order to maximize the submodular social welfare, measured as:

$$\text{submodular-social-welfare}(E_1, E_2, \dots, E_n) = \sum_{i=1}^{n} g_i(E_i). \quad (19.32)$$

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 $E = E_1 \cup E_2 \cup \cdots \cup E_n$ into n blocks in order to maximize the submodular social welfare, measured as:

$$\text{submodular-social-welfare}(E_1, E_2, \dots, E_n) = \sum_{i=1}^{n} g_i(E_i). \quad (19.32)$$

- 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.,

$$E' = \underbrace{E \uplus E \uplus \cdots \uplus E}_{n\times} \tag{19.33}$$

Submodular Welfare: Submodular Max over matroid partition

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

$$E' = \underbrace{E \uplus E \uplus \cdots \uplus E}_{n \times} \tag{19.33}$$

- 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.,

$$E' = \underbrace{E \uplus E \uplus \cdots \uplus E}_{n\times} \tag{19.33}$$

- 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.,

$$E' = \underbrace{E \uplus E \uplus \cdots \uplus E}_{n\times} \tag{19.33}$$

- 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.,

$$E' = \underbrace{E \uplus E \uplus \cdots \uplus E}_{n\times} \tag{19.33}$$

- 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.,

$$E' = \underbrace{E \uplus E \uplus \cdots \uplus E}_{n\times} \tag{19.33}$$

- 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

$$\mathcal{I} = \left\{ S \subseteq E' : \forall e \in E, |S \cap E_e| \le 1 \right\} \tag{19.34}$$

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.

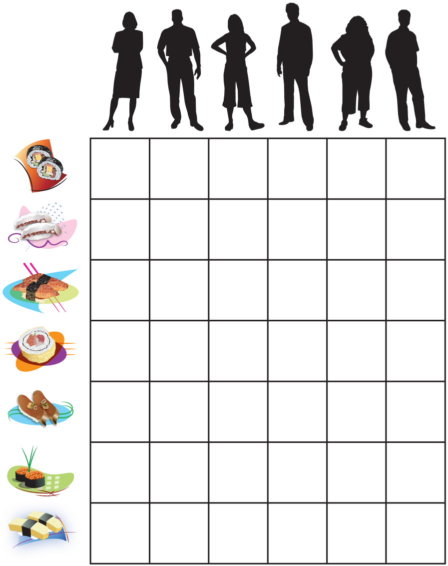
Submodular Welfare: Submodular Max over matroid partition

- 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

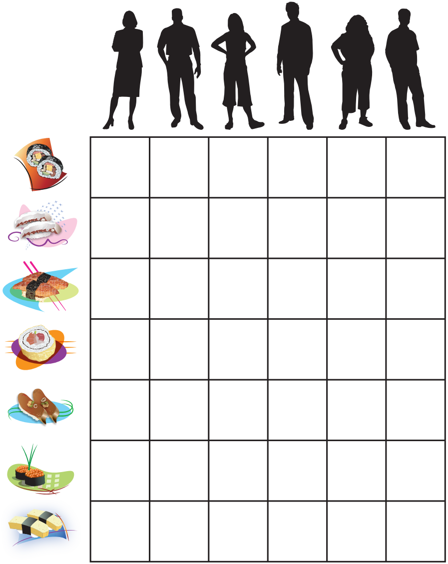
- Hence, S is independent in matroid $\mathcal{M} = (E', I)$ if S uses each original element no more than once.
- Create submodular function $f' : 2^{E'} \to \mathbb{R}_+$ with $f'(S) = \sum_{i=1}^{n} g_i(S \cap E^{(i)})$.
- Submodular welfare maximization becomes matroid constrained submodular max $\max\{f'(S)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

Submodular Social Welfare



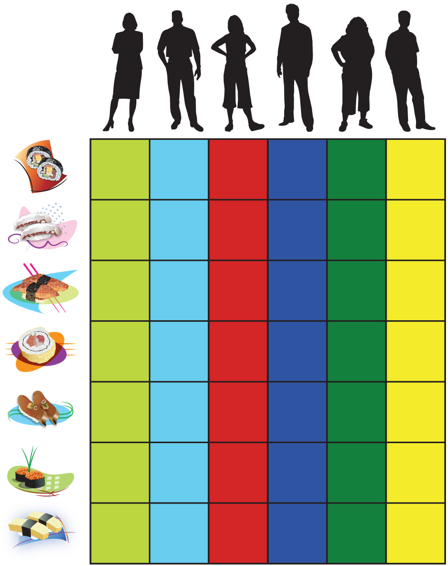
- Have $n = 6$ people (who don't like to share) and $|E| = m = 7$ pieces of sushi. E.g., $e \in E$ might be $e =$ "salmon roll".

Submodular Social Welfare



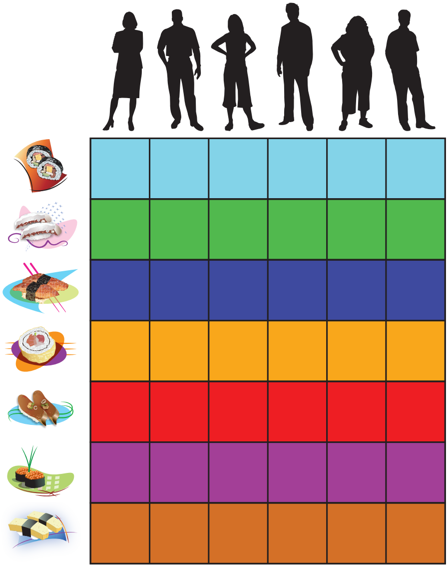
- Have $n = 6$ people (who don't like to share) and $|E| = m = 7$ pieces of sushi. E.g., $e \in E$ might be $e =$ "salmon roll".
- Goal: distribute sushi to people to maximize social welfare.

Submodular Social Welfare



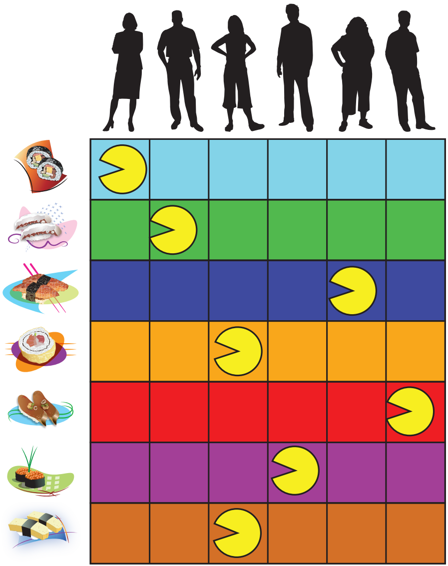
- Have $n = 6$ people (who don't like to share) and $|E| = m = 7$ pieces of sushi. E.g., $e \in E$ might be $e =$ "salmon roll".
- Goal: distribute sushi to people to maximize social welfare.
- Ground set disjoint union $E \uplus E \uplus E \uplus E \uplus E \uplus E$.

Submodular Social Welfare



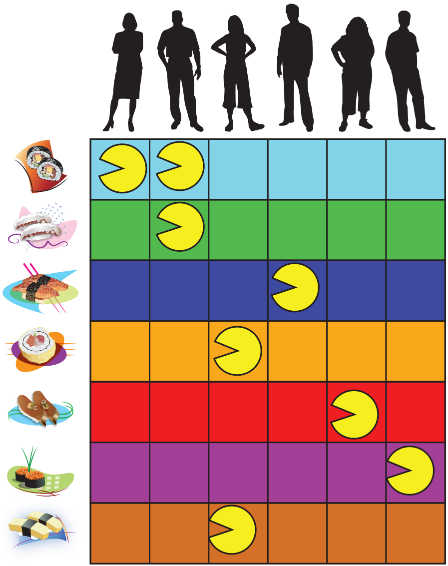
- Have $n = 6$ people (who don't like to share) and $|E| = m = 7$ pieces of sushi. E.g., $e \in E$ might be $e =$ "salmon roll".
- Goal: distribute sushi to people to maximize social welfare.
- Ground set disjoint union $E \uplus E \uplus E \uplus E \uplus E \uplus E$.
- Partition matroid partitions: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

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

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i}{\operatorname{argmax}} \Big(f(S_i \cup \{v\}) - f(S_i) \Big) \right\} \tag{19.35}$$

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

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i} \Big(f(S_i \cup \{v\}) - f(S_i) \Big) \right\} \tag{19.35}$$

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

$$S_{i+1} = S_i \cup \left\{ \operatorname*{argmax}_{v \in V \setminus S_i} \frac{f(S_i \cup \{v\}) - f(S_i)}{c(v)} \right\} \tag{19.36}$$

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 $v \in V \setminus S$.

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 $v \in V \setminus S$.
- 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 $v \in V \setminus S$.
- The following interesting result is true for any submodular function:

Lemma 19.5.2

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 $v \in V \setminus S$.
- The following interesting result is true for any submodular function:

Lemma 19.5.2

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 $v \in V \setminus S$.
- The following interesting result is true for any submodular function:

Lemma 19.5.2

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 $v \in V \setminus S$.
- The following interesting result is true for any submodular function:

Lemma 19.5.2

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 $v \in V \setminus S$.
- The following interesting result is true for any submodular function:

Lemma 19.5.2

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 $v \in V \setminus S$.
- The following interesting result is true for any submodular function:

Lemma 19.5.2

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 7: 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}
```

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
$\lvert S\rvert \leq k$	$1-1/e$	$1-1/e$	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.

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:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} \tag{19.37}$$

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:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} \tag{19.37}$$

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

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:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} \tag{19.37}$$

- $c \in [0, 1]$. When $c = 0$, $f(j|S) = f(j|\emptyset)$ for all S, j, a sufficient condition for modularity, and we saw in Lecture 6, 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:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} \tag{19.37}$$

- $c \in [0, 1]$. When $c = 0$, $f(j|S) = f(j|\emptyset)$ for all S, j, a sufficient condition for modularity, and we saw in Lecture 6, Theorem **??** that greedy is optimal for max weight indep. set of a matroid.
- When $c = 1$ then submodular function is "infinitely 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:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} \tag{19.37}$$

- $c \in [0, 1]$. When $c = 0$, $f(j|S) = f(j|\emptyset)$ for all S, j, a sufficient condition for modularity, and we saw in Lecture 6, Theorem **??** that greedy is optimal for max weight indep. set of a matroid.
- When $c = 1$ then submodular function is "infinitely curved", i.e., there exists is a subset that fully spans some other element.
- Matroid rank functions with some dependence is infinitely curved.

Curvature of a Submodular function

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

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{j : f(j|\emptyset) \neq 0} \frac{f(j|V \setminus \{j\})}{f(j|\emptyset)} \quad (19.38)$$

Curvature of a Submodular function

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

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{j : f(j|\emptyset) \neq 0} \frac{f(j|V \setminus \{j\})}{f(j|\emptyset)} \quad (19.38)$$

- Note: Matroid rank is either modular $c = 0$ or infinitely 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:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{j : f(j|\emptyset) \neq 0} \frac{f(j|V \setminus \{j\})}{f(j|\emptyset)} \quad (19.38)$$

- Note: Matroid rank is either modular $c = 0$ or infinitely 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:

$$c \triangleq 1 - \min_{S, j \notin S : f(j|\emptyset) \neq 0} \frac{f(j|S)}{f(j|\emptyset)} = 1 - \min_{j : f(j|\emptyset) \neq 0} \frac{f(j|V \setminus \{j\})}{f(j|\emptyset)} \quad (19.38)$$

- Note: Matroid rank is either modular $c = 0$ or infinitely 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 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.

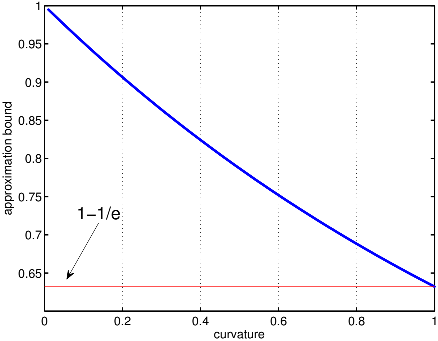
Curvature and approximation

- 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

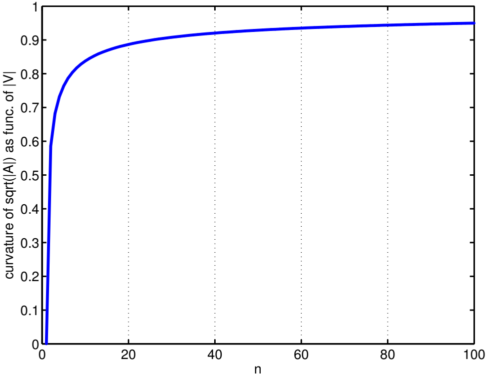
- Curvature limitation can help the greedy algorithm in terms of approximation bounds.
- Conforti & Cornuéjols showed that greedy gives a $1/(1+c)$ approximation to $\max\{f(S) : S \in \mathcal{I}\}$ when f has total curvature c.
- Hence, greedy subject to matroid constraint is a $\max(1/(1+c), 1/2)$ 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 $\frac{1}{c}(1-e^{-c})$



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

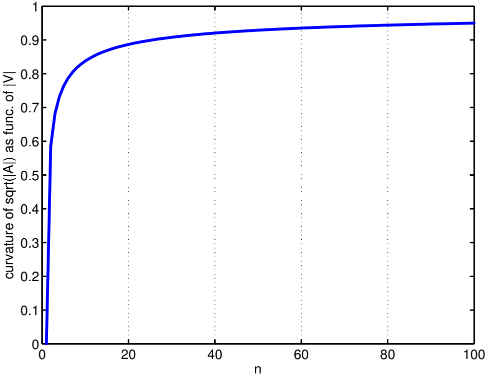
Curvature of $f(S) = \sqrt{|S|}$ as function of $|V| = n$



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

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

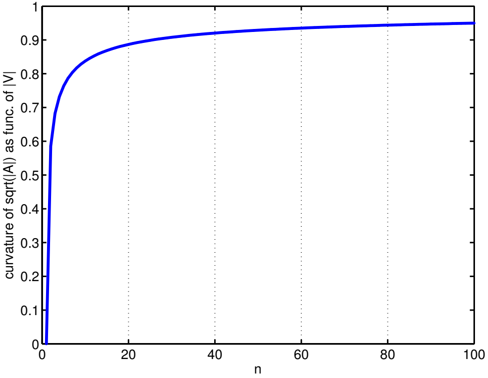
Curvature of $f(S) = \sqrt{|S|}$ as function of $|V| = n$



- $f(S) = \sqrt{|S|}$ 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.

Submodular Max and polyhedral approaches

- We've spent much time discussing SFM and the polymatroidal polytope, and in general polyhedral approaches for SFM.

Submodular Max and polyhedral approaches

- We've spent much time discussing SFM and the polymatroidal polytope, and in general polyhedral approaches for SFM.
- Most of the approaches for submodular max have not used such an approach, probably due to the difficulty in computing the "concave extension" of a submodular function (the convex extension is easy, namely the Lovász extension).

Submodular Max and polyhedral approaches

- We've spent much time discussing SFM and the polymatroidal polytope, and in general polyhedral approaches for SFM.
- Most of the approaches for submodular max have not used such an approach, probably due to the difficulty in computing the "concave extension" of a submodular function (the convex extension is easy, namely the Lovász extension).
- A paper by Chekuri, Vondrak, and Zenklusen (2011) make some progress on this front using multilinear extensions.

Multilinear extension

Definition 19.5.3

For a set function $f : 2^V \to \mathbb{R}$, define its multilinear extension $F : [0,1]^V \to \mathbb{R}$ by

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \tag{19.39}$$

Multilinear extension

Definition 19.5.3

For a set function $f : 2^V \to \mathbb{R}$, define its multilinear extension $F : [0,1]^V \to \mathbb{R}$ by

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \tag{19.39}$$

- Note that $F(x) = Ef(\hat{x})$ where $\hat{x}$ is a random binary vector over $\{0,1\}^V$ with elements independent w. probability x_i for $\hat{x}_i$.

Multilinear extension

Definition 19.5.3

For a set function $f : 2^V \to \mathbb{R}$, define its multilinear extension $F : [0,1]^V \to \mathbb{R}$ by

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \tag{19.39}$$

- Note that $F(x) = Ef(\hat{x})$ where $\hat{x}$ is a random binary vector over $\{0,1\}^V$ with elements independent w. probability x_i for $\hat{x}_i$.
- While this is defined for any set function, we have:

Multilinear extension

Definition 19.5.3

For a set function $f : 2^V \to \mathbb{R}$, define its multilinear extension $F : [0,1]^V \to \mathbb{R}$ by

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \tag{19.39}$$

- Note that $F(x) = Ef(\hat{x})$ where $\hat{x}$ is a random binary vector over $\{0,1\}^V$ with elements independent w. probability x_i for $\hat{x}_i$.
- While this is defined for any set function, we have:

Lemma 19.5.4

Let $F : [0,1]^V \to \mathbb{R}$ be multilinear extension of set function $f : 2^V \to \mathbb{R}$, then

Multilinear extension

Definition 19.5.3

For a set function $f : 2^V \to \mathbb{R}$, define its multilinear extension $F : [0,1]^V \to \mathbb{R}$ by

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \tag{19.39}$$

- Note that $F(x) = Ef(\hat{x})$ where $\hat{x}$ is a random binary vector over $\{0,1\}^V$ with elements independent w. probability x_i for $\hat{x}_i$.
- While this is defined for any set function, we have:

Lemma 19.5.4

Let $F : [0,1]^V \to \mathbb{R}$ be multilinear extension of set function $f : 2^V \to \mathbb{R}$, then

- *If f is monotone non-decreasing, then $\frac{\partial F}{\partial x_i} \geq 0$ for all $i \in V$, $x \in [0,1]^V$.*

Multilinear extension

Definition 19.5.3

For a set function $f : 2^V \to \mathbb{R}$, define its multilinear extension $F : [0,1]^V \to \mathbb{R}$ by

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \tag{19.39}$$

- Note that $F(x) = Ef(\hat{x})$ where $\hat{x}$ is a random binary vector over $\{0,1\}^V$ with elements independent w. probability x_i for $\hat{x}_i$.
- While this is defined for any set function, we have:

Lemma 19.5.4

Let $F : [0,1]^V \to \mathbb{R}$ be multilinear extension of set function $f : 2^V \to \mathbb{R}$, then

- *If f is monotone non-decreasing, then $\frac{\partial F}{\partial x_i} \geq 0$ for all $i \in V$, $x \in [0,1]^V$.*
- *If f is submodular, then $\frac{\partial^2 F}{\partial x_i \partial x_j} \leq 0$ for all $i, j\ inV$, $x \in [0,1]^V$.*

Multilinear extension

- Moreover, we have

Multilinear extension

- Moreover, we have

Lemma 19.5.5

Let $F : [0,1]^V \to \mathbb{R}$ *be multilinear extension of set function* $f : 2^V \to \mathbb{R}$*, then*

Multilinear extension

- Moreover, we have

Lemma 19.5.5

Let $F : [0,1]^V \rightarrow \mathbb{R}$ be multilinear extension of set function $f : 2^V \rightarrow \mathbb{R}$, then

- *If f is monotone non-decreasing, then F is non-decreasing along any line of direction $d \in \mathbb{R}^E$ with $d \geq 0$*

Multilinear extension

- Moreover, we have

Lemma 19.5.5

Let $F : [0,1]^V \to \mathbb{R}$ be multilinear extension of set function $f : 2^V \to \mathbb{R}$, then

- *If f is monotone non-decreasing, then F is non-decreasing along any line of direction $d \in \mathbb{R}^E$ with $d \geq 0$*
- *If f is submodular, then F is concave along any line of direction $d \geq 0$, and is convex along any line of direction $\mathbf{1}_v - \mathbf{1}_w$ for any $v, w \in V$.*

Multilinear extension

- Moreover, we have

Lemma 19.5.5

Let $F : [0,1]^V \to \mathbb{R}$ be multilinear extension of set function $f : 2^V \to \mathbb{R}$, then

- *If f is monotone non-decreasing, then F is non-decreasing along any line of direction $d \in \mathbb{R}^E$ with $d \geq 0$*
- *If f is submodular, then F is concave along any line of direction $d \geq 0$, and is convex along any line of direction $\mathbf{1}_v - \mathbf{1}_w$ for any $v, w \in V$.*

- Another connection between submodularity and convexity/concavity

Multilinear extension

- Moreover, we have

Lemma 19.5.5

Let $F : [0,1]^V \to \mathbb{R}$ be multilinear extension of set function $f : 2^V \to \mathbb{R}$, then

- *If f is monotone non-decreasing, then F is non-decreasing along any line of direction $d \in \mathbb{R}^E$ with $d \geq 0$*
- *If f is submodular, then F is concave along any line of direction $d \geq 0$, and is convex along any line of direction $\mathbf{1}_v - \mathbf{1}_w$ for any $v, w \in V$.*

- Another connection between submodularity and convexity/concavity
- but note, unlike the Lovász extension, this function is neither.

Submodular Max and polyhedral approaches

- Basic idea: Given a set of constraints $\mathcal{I}$, we form a polytope $P_{\mathcal{I}}$ such that $\{\mathbf{1}_I : I \in \mathcal{I}\} \subseteq P_{\mathcal{I}}$
- We find $\max_{x \in P_{\mathcal{I}}} F(x)$ where $F(x)$ is the multi-linear extension of f, to find a fractional solution x^*
- We then round x^* to a point on the hypercube, thus giving us a solution to the discrete problem.

Submodular Max and polyhedral approaches

- In the recent paper by Chekuri, Vondrak, and Zenklusen, they show:

Submodular Max and polyhedral approaches

- In the recent paper by Chekuri, Vondrak, and Zenklusen, they show:
- 1) constant factor approximation algorithm for $\max\{F(x) : x \in P\}$ for any down-monotone solvable polytope P and F multilinear extension of any non-negative submodular function.

Submodular Max and polyhedral approaches

- In the recent paper by Chekuri, Vondrak, and Zenklusen, they show:
- 1) constant factor approximation algorithm for $\max \{F(x) : x \in P\}$ for any down-monotone solvable polytope P and F multilinear extension of any non-negative submodular function.
- 2) A randomized rounding (pipage rounding) scheme to obtain an integer solution

Submodular Max and polyhedral approaches

- In the recent paper by Chekuri, Vondrak, and Zenklusen, they show:
- 1) constant factor approximation algorithm for $\max\{F(x) : x \in P\}$ for any down-monotone solvable polytope P and F multilinear extension of any non-negative submodular function.
- 2) A randomized rounding (pipage rounding) scheme to obtain an integer solution
- 3) An optimal $(1 - 1/e)$ instance of their rounding scheme that can be used for a variety of interesting independence systems, including $O(1)$ knapsacks, k matroids and $O(1)$ knapsacks, a k-matchoid and ℓ sparse packing integer programs, and unsplittable flow in paths and trees.

Submodular Max and polyhedral approaches

- In the recent paper by Chekuri, Vondrak, and Zenklusen, they show:
- 1) constant factor approximation algorithm for $\max\{F(x) : x \in P\}$ for any down-monotone solvable polytope P and F multilinear extension of any non-negative submodular function.
- 2) A randomized rounding (pipage rounding) scheme to obtain an integer solution
- 3) An optimal $(1 - 1/e)$ instance of their rounding scheme that can be used for a variety of interesting independence systems, including $O(1)$ knapsacks, k matroids and $O(1)$ knapsacks, a k-matchoid and ℓ sparse packing integer programs, and unsplittable flow in paths and trees.
- Also, Vondrak showed that this scheme achieves the $\frac{1}{c}(1 - e^{-c})$ curvature based bound for any matroid, which matches the bound we had earlier for uniform matroids with standard greedy.

Submodular Max and polyhedral approaches

- In the recent paper by Chekuri, Vondrak, and Zenklusen, they show:
- 1) constant factor approximation algorithm for $\max\{F(x) : x \in P\}$ for any down-monotone solvable polytope P and F multilinear extension of any non-negative submodular function.
- 2) A randomized rounding (pipage rounding) scheme to obtain an integer solution
- 3) An optimal $(1 - 1/e)$ instance of their rounding scheme that can be used for a variety of interesting independence systems, including $O(1)$ knapsacks, k matroids and $O(1)$ knapsacks, a k-matchoid and ℓ sparse packing integer programs, and unsplittable flow in paths and trees.
- Also, Vondrak showed that this scheme achieves the $\frac{1}{c}(1 - e^{-c})$ curvature based bound for any matroid, which matches the bound we had earlier for uniform matroids with standard greedy.
- In practice, one needs to do Monte-Carlo methods to estimate the multilinear extension (so further approximations would apply).