

# EE595A – Submodular functions, their optimization and applications – Spring 2011

Prof. Jeff Bilmes

University of Washington, Seattle  
Department of Electrical Engineering  
Spring Quarter, 2011

[http://ssli.ee.washington.edu/~bilmes/ee595a\\_spring\\_2011/](http://ssli.ee.washington.edu/~bilmes/ee595a_spring_2011/)

Lecture 20 - June 9th, 2011

# Announcements

- Final lecture.

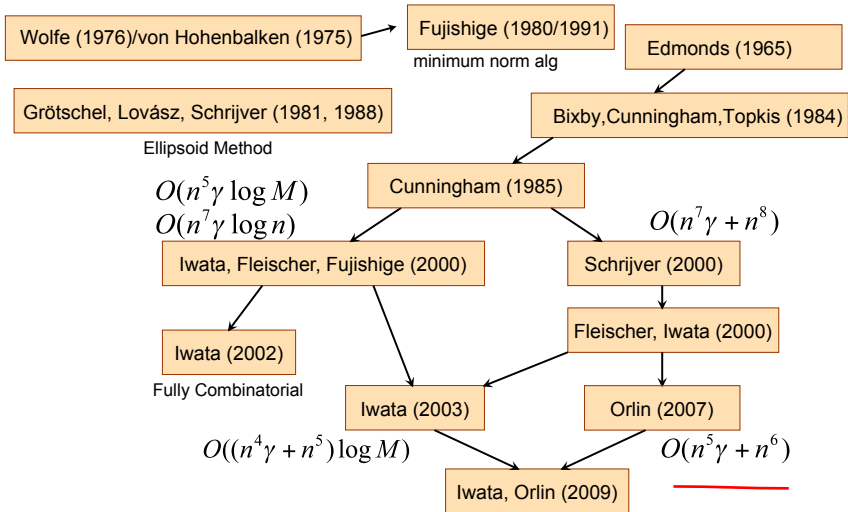
# Class Road Map

We need to find one makeup lecture this term.

- L1 (3/30):
- L2 (4/1):
- L3 (4/6):
- L4 (4/8):
- L5 (4/13):
- L6 (4/15):
- L7 (4/20):
- L8 (4/27):
- L9 (4/29):
- L10 (5/4):
- L11 (5/6): On SFM, polymatroid member & greedy, Lovász ext.
- L12 (5/11): Lovász ext. + polymatroid props.
- L13 (5/13): More polymatroids, start lattices
- L14 (5/18): lattices/submodular
- L15 (5/20): lattices,  $\rightarrow$  SFM.
- L16 (5/25):  $\rightarrow$  SFM
- L17 (5/27): dep/sat
- L18 (6/1): exchange capacities
- L19 (6/3): SFM algorithm
- L20: (6/9): Sym SFM, sub. max,

# SFM Summary (modified from S. Iwata's slides)

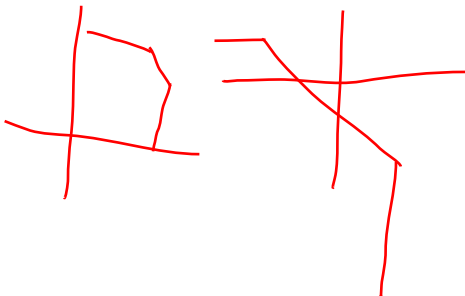
## Submodular Function Minimization



# SFM

- Recall extended polymatroid which applies to any normalized submodular function

$$EP_f = \{x : x(A) \leq f(A), \forall A \subseteq E\} \tag{1}$$



# SFM

- Recall extended polymatroid which applies to any normalized submodular function

$$EP_f = \{x : x(A) \leq f(A), \forall A \subseteq E\} \tag{1}$$

- Let  $f$  be submodular and let  $a \in \mathbb{R}^E$ . Define function  $f_a$  on  $E$  as

$$f_a(B) = \min_{A \subseteq B} (f(A) + a(B \setminus A)) \tag{2}$$

for any  $B \subseteq E$ .

# SFM

- Recall extended polymatroid which applies to any normalized submodular function

$$EP_f = \{x : x(A) \leq f(A), \forall A \subseteq E\} \tag{1}$$

- Let  $f$  be submodular and let  $a \in \mathbb{R}^E$ . Define function  $f_a$  on  $E$  as

$$f_a(B) = \min_{A \subseteq B} (f(A) + a(B \setminus A)) \tag{2}$$

for any  $B \subseteq E$ .

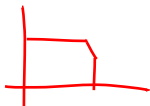
- Then  $f_a$  is submodular. Moreover,

$$EP_{f_a} = \{x \in EP_f : x \leq a\} \tag{3}$$



and

$$P_{f_a} = \{x \in P_f : x \leq a\} \tag{4}$$



## SFM

- Recall extended polymatroid which applies to any normalized submodular function

$$EP_f = \{x : x(A) \leq f(A), \forall A \subseteq E\} \quad (1)$$

- Let  $f$  be submodular and let  $a \in \mathbb{R}^E$ . Define function  $f_a$  on  $E$  as

$$f_a(B) = \min_{A \subseteq B} (f(A) + a(B \setminus A)) \quad (2)$$

for any  $B \subseteq E$ .

- Then  $f_a$  is submodular. Moreover,

$$EP_{f_a} = \{x \in EP_f : x \leq a\} \quad (3)$$

and

$$P_{f_a} = \{x \in P_f : x \leq a\} \quad (4)$$

- Therefore, if  $\underline{P}$  is an extended polymatroid, then so is  $\underline{P} \cap \{x : x \leq a\}$ .

## SFM

- Recall extended polymatroid which applies to any normalized submodular function

$$EP_f = \{x : x(A) \leq f(A), \forall A \subseteq E\} \quad (1)$$

- Let  $f$  be submodular and let  $a \in \mathbb{R}^E$ . Define function  $f_a$  on  $E$  as

$$f_a(B) = \min_{A \subseteq B} (f(A) + a(B \setminus A)) \quad (2)$$

for any  $B \subseteq E$ .

- Then  $f_a$  is submodular. Moreover,

$$EP_{f_a} = \{x \in EP_f : x \leq a\} \quad (3)$$

and

$$P_{f_a} = \{x \in P_f : x \leq a\} \quad (4)$$

- Therefore, if  $P$  is an extended polymatroid, then so is  $P \cap \{x : x \leq a\}$ .
- We also saw earlier that for any  $EP_f$ ,

$$f(A) = \max \{x(A) : x \in EP_f\} \quad (5)$$

for any  $A \subseteq E$ .

## SFM

- Now, take  $a = 0$ , and we get

$$f_0(E) = \min_{A \subseteq E} f(A) \quad (6)$$

$$EP_{f_0} = EP_f \cap \{x : x \leq 0\} \quad (7)$$

# SFM

- Now, take  $a = 0$ , and we get

$$f_0(E) = \min_{A \subseteq E} f(A) \tag{6}$$

$$EP_{f_0} = EP_f \cap \{x : x \leq 0\} \tag{7}$$

- This gives

$$\min_{A \subseteq E} f(A) = \max \{x(E) : x \in EP_f, x \leq 0\} \tag{8}$$

# SFM

- Now, take  $a = 0$ , and we get

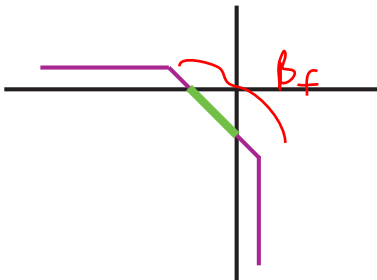
$$f_0(E) = \min_{A \subseteq E} f(A) \tag{6}$$

$$EP_{f_0} = EP_f \cap \{x : x \leq 0\} \tag{7}$$

- This gives

$$\min_{A \subseteq E} f(A) = \max \{x(E) : x \in EP_f, x \leq 0\} \tag{8}$$

- We see



# SFM

- Fujishige showed, based on above, SFM is done via

$$\min \left\{ \sum_{e \in E} x(e)^2 : x \in B_f \right\} \tag{9}$$

where  $B_f$  is the base polytope of submodular  $f$ .

# SFM

- Fujishige showed, based on above, SFM is done via

$$\min \left\{ \sum_{e \in E} x(e)^2 : x \in B_f \right\} \tag{9}$$

where  $B_f$  is the base polytope of submodular  $f$ .

- Given solution  $\hat{x}$  to the above, minimal/maximal solution becomes  $A_- = \{e : \hat{x}(e) < 0\}$  and  $A_0 = \{e : \hat{x}(e) \leq 0\}$ .

# SFM

- Fujishige showed, based on above, SFM is done via

$$\min \left\{ \sum_{e \in E} x(e)^2 : x \in B_f \right\} \tag{9}$$

where  $B_f$  is the base polytope of submodular  $f$ .

- Given solution  $\hat{x}$  to the above, minimal/maximal solution becomes  $A_- = \{e : \hat{x}(e) < 0\}$  and  $A_0 = \{e : \hat{x}(e) \leq 0\}$ .
- The optimization can be solved by Wolfe's numeric procedure, 1974.

# SFM

- Fujishige showed, based on above, SFM is done via

$$\min \left\{ \sum_{e \in E} x(e)^2 : x \in B_f \right\} \tag{9}$$

where  $B_f$  is the base polytope of submodular  $f$ .

- Given solution  $\hat{x}$  to the above, minimal/maximal solution becomes  $A_- = \{e : \hat{x}(e) < 0\}$  and  $A_0 = \{e : \hat{x}(e) \leq 0\}$ .
- The optimization can be solved by Wolfe's numeric procedure, 1974.
- This is called the “minimum norm” algorithm for SFM.

# SFM

- Fujishige showed, based on above, SFM is done via

$$\min \left\{ \sum_{e \in E} x(e)^2 : x \in B_f \right\} \tag{9}$$

where  $B_f$  is the base polytope of submodular  $f$ .

- Given solution  $\hat{x}$  to the above, minimal/maximal solution becomes  $A_- = \{e : \hat{x}(e) < 0\}$  and  $A_0 = \{e : \hat{x}(e) \leq 0\}$ .
- The optimization can be solved by Wolfe's numeric procedure, 1974.
- This is called the "minimum norm" algorithm for SFM.
- It is often good in practice, currently go-to algorithm when needing submodular function minimization

# SFM

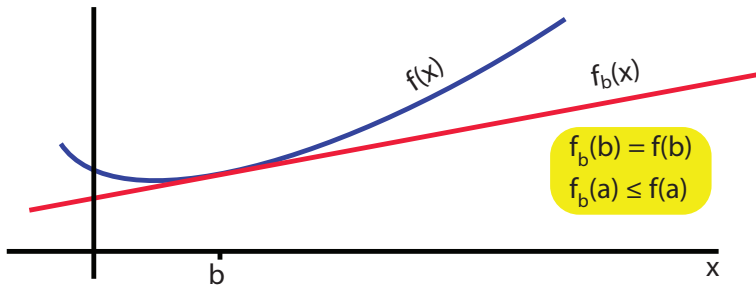
- Fujishige showed, based on above, SFM is done via

$$\min \left\{ \sum_{e \in E} x(e)^2 : x \in B_f \right\} \tag{9}$$

where  $B_f$  is the base polytope of submodular  $f$ .

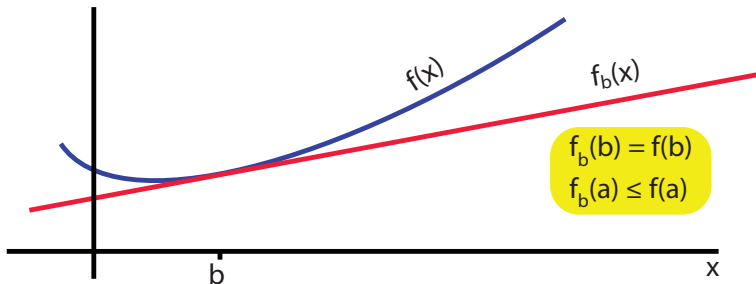
- Given solution  $\hat{x}$  to the above, minimal/maximal solution becomes  $A_- = \{e : \hat{x}(e) < 0\}$  and  $A_0 = \{e : \hat{x}(e) \leq 0\}$ .
- The optimization can be solved by Wolfe's numeric procedure, 1974.
- This is called the "minimum norm" algorithm for SFM.
- It is often good in practice, currently go-to algorithm when needing submodular function minimization
- although complexity is currently unknown.

# Convex Functions and Tight Linear Lower Bounds



- We have that  $f(x)$  is convex,  $f_b(x)$  is linear, and is a tight lower bound on  $f(x)$ .

# Convex Functions and Tight Linear Lower Bounds



- We have that  $f(x)$  is convex,  $f_b(x)$  is linear, and is a tight lower bound on  $f(x)$ .
- Can there be both a tight linear upper bound and tight linear lower bound on a convex function, where each bound is tight at the same point?

# Submodular Functions and Tight Linear Lower Bounds

- We saw that given any submodular function  $f$ , and a base  $x \in B_f$  generated by the greedy algorithm on the ordered set  $E_n = (e_1, e_2, \dots, e_n)$ , we have
- $x$  is tight at each of  $E_i$ . That is  $x(E_i) = f(E_i)$
- Also since  $x \in P_f$ , then  $x(A) \leq f(A) \quad \forall A$ .


# Submodular Functions and Tight Linear Upper Bounds

- Recall from lecture 2 (equation 63)

$$f(T) \leq f(S) + \sum_{j \in T \setminus S} \rho_j(S) - \sum_{j \in S \setminus T} \rho_j(S \cup T - \{j\}), \forall S, T \subseteq E \quad (10)$$

# Submodular Functions and Tight Linear Upper Bounds

- Recall from lecture 2 (equation 63)

$$f(T) \leq f(S) + \sum_{j \in T \setminus S} \rho_j(S) - \sum_{j \in S \setminus T} \rho_j(S \cup T - \{j\}), \forall S, T \subseteq E \quad (10)$$


- Using submodular diminishing returns, we can weaken this to

$$f(T) \leq f(S) + \sum_{j \in T \setminus S} \rho_j(S) - \sum_{j \in S \setminus T} \rho_j(E - \{j\}), \forall S, T \subseteq E \quad (11)$$

# Submodular Functions and Tight Linear Upper Bounds

- Recall from lecture 2 (equation 63)

$$f(T) \leq f(S) + \sum_{j \in T \setminus S} \rho_j(S) - \sum_{j \in S \setminus T} \rho_j(S \cup T - \{j\}), \forall S, T \subseteq E \quad (10)$$

- Using submodular diminishing returns, we can weaken this to

$$f(T) \leq f(S) + \sum_{j \in T \setminus S} \rho_j(S) - \sum_{j \in S \setminus T} \rho_j(E - \{j\}), \forall S, T \subseteq E \quad (11)$$

- Thus, we can define a tight modular upper bound. Given  $S \subseteq E$ ,

$$\underline{h_S(T)} \stackrel{\text{def}}{=} f(S) + \sum_{j \in T \setminus S} \rho_j(S) - \sum_{j \in S \setminus T} \rho_j(E - \{j\}) \quad (12)$$

$$= \underline{f(S)} + \sum_{j \in T \setminus S} \rho_j(S) - \sum_{j \in S} \rho_j(E - \{j\}) + \sum_{j \in S \cap T} \rho_j(E - \{j\}) \quad \checkmark$$

$$= \text{const}_S + \text{modular}(T) \quad (13)$$

# Submodular Functions and Tight Linear Upper Bounds

- Recall from lecture 2 (equation 63)

$$f(T) \leq f(S) + \sum_{j \in T \setminus S} \rho_j(S) - \sum_{j \in S \setminus T} \rho_j(S \cup T - \{j\}), \quad \forall S, T \subseteq E \quad (10)$$

- Using submodular diminishing returns, we can weaken this to

$$f(T) \leq f(S) + \sum_{j \in T \setminus S} \rho_j(S) - \sum_{j \in S \setminus T} \rho_j(E - \{j\}), \quad \forall S, T \subseteq E \quad (11)$$

- Thus, we can define a tight modular upper bound. Given  $S \subseteq E$ ,

$$h_S(T) \stackrel{\text{def}}{=} f(S) + \sum_{j \in T \setminus S} \rho_j(S) - \sum_{j \in S \setminus T} \rho_j(E - \{j\}) \quad (12)$$

$$= f(S) + \sum_{j \in T \setminus S} \rho_j(S) - \sum_{j \in S} \rho_j(E - \{j\}) + \sum_{j \in S \cap T} \rho_j(E - \{j\})$$

$$= \text{const}_S + \text{modular}(T) \quad (13)$$

- Then  $h_S$  is modular,  $h_S(S) = f(S)$ , and  $f(T) \leq h_S(T) \quad \forall T$ .

# Minimization of a Symmetric Submodular Functions

- If  $f$  is both submodular, and also we have that  $f(A) = f(E \setminus A)$  for all  $A$ , then  $f$  is said to be **symmetric submodular**

ex:  $f(A) = I(\underbrace{X_A}; \underbrace{X_{V \setminus A}})$

# Minimization of a Symmetric Submodular Functions

- If  $f$  is both submodular, and also we have that  $f(A) = f(E \setminus A)$  for all  $A$ , then  $f$  is said to be **symmetric submodular**
- Given any non-symmetric submodular function  $g$ , we can always symmetrize it,  $f(A) = g(A) + g(E \setminus A) - g(E)$ . Then  $f$  is symmetric and normalized  $f(\emptyset) = 0$ .

# Minimization of a Symmetric Submodular Functions

- If  $f$  is both submodular, and also we have that  $f(A) = f(E \setminus A)$  for all  $A$ , then  $f$  is said to be **symmetric submodular**
- Given any non-symmetric submodular function  $g$ , we can always symmetrize it,  $f(A) = g(A) + g(E \setminus A) - g(E)$ . Then  $f$  is symmetric and normalized  $f(\emptyset) = 0$ .
- Such an  $f$  is also non-negative since

$$2f(A) = f(A) + f(E \setminus A) \geq f(\emptyset) + f(E) = 2f(\emptyset) \geq 0 \quad (14)$$

$$H(x_A) + H(x_{V \setminus A}) - H(x_V) = I(x_A; x_{V \setminus A})$$

# Minimization of a Symmetric Submodular Functions

- If  $f$  is both submodular, and also we have that  $f(A) = f(E \setminus A)$  for all  $A$ , then  $f$  is said to be **symmetric submodular**
- Given any non-symmetric submodular function  $g$ , we can always symmetrize it,  $f(A) = g(A) + g(E \setminus A) - g(E)$ . Then  $f$  is symmetric and normalized  $f(\emptyset) = 0$ .

- Such an  $f$  is also non-negative since

$$2f(A) = f(A) + f(E \setminus A) \geq f(\emptyset) + f(E) = 2f(\emptyset) \geq 0 \quad (14)$$

- Such a symmetrized submodular function measures the “dependence” between  $A$  and  $E \setminus A$ . If  $f(A) = 0$  then  $g$  is decomposable as  $g(B) = \underbrace{g(B \cap A)} + \underbrace{g(B \cap (E \setminus A))}$ .

# Minimization of a Symmetric Submodular Functions

- If  $f$  is both submodular, and also we have that  $f(A) = f(E \setminus A)$  for all  $A$ , then  $f$  is said to be **symmetric submodular**
- Given any non-symmetric submodular function  $g$ , we can always symmetrize it,  $f(A) = g(A) + g(E \setminus A) - g(E)$ . Then  $f$  is symmetric and normalized  $f(\emptyset) = 0$ .
- Such an  $f$  is also non-negative since

$$2f(A) = f(A) + f(E \setminus A) \geq f(\emptyset) + f(E) = 2f(\emptyset) \geq 0 \quad (14)$$

- Such a symmetrized submodular function measures the “dependence” between  $A$  and  $E \setminus A$ . If  $f(A) = 0$  then  $g$  is decomposable as  $g(B) = g(B \cap A) + g(B \cap (E \setminus A))$ .
- Example:  $g = \text{entropy}$ , then  $f = \text{mutual information}$ .

# Minimization of a Symmetric Submodular Functions

- If  $f$  is both submodular, and also we have that  $f(A) = f(E \setminus A)$  for all  $A$ , then  $f$  is said to be **symmetric submodular**
- Given any non-symmetric submodular function  $g$ , we can always symmetrize it,  $f(A) = g(A) + g(E \setminus A) - g(E)$ . Then  $f$  is symmetric and normalized  $f(\emptyset) = 0$ .

- Such an  $f$  is also non-negative since

$$2f(A) = f(A) + f(E \setminus A) \geq f(\emptyset) + f(E) = 2f(\emptyset) \geq 0 \quad (14)$$

- Such a symmetrized submodular function measures the “dependence” between  $A$  and  $E \setminus A$ . If  $f(A) = 0$  then  $g$  is decomposable as  $g(B) = g(B \cap A) + g(B \cap (E \setminus A))$ .
- Example:  $g =$  entropy, then  $f =$  mutual information.
- In the following slides,  $f$  will be presumed to be the symmetrized version of  $g$ .

# Minimization of a Symmetric Submodular Functions

- Minimizing symmetric submodular functions can be done in strongly polynomial time  $O(n^3)$ . The algorithm is by Nagamochi & Ibaracki ~~2002~~ <sup>1992</sup> for graph cuts and was 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 is by Nagamochi & Ibaracki 2002 for graph cuts and was shown by Queyranne in 1995 to work for sym. SFM.
- The algorithm first finds a MA (maximum adjacency) or a maximum back order.

- 
- 1 Choose  $v_1$  arbitrarily ;
  - 2  $W_1 \leftarrow (v_1)$  ;
  - 3 **for**  $i \leftarrow 1 \dots |V| - 1$  **do**
  - 4     Choose  $v_{i+1} \in \operatorname{argmin}_{u \in V \setminus W_i} (g(W_i \cup \{u\}) - g(u))$  ;
  - 5      $W_{i+1} \leftarrow (W_i, v_{i+1})$  ; /\* 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 is by Nagamochi & Ibaracki 2002 for graph cuts and was shown by Queyranne in 1995 to work for sym. SFM.
- The algorithm first finds a MA (maximum adjacency) or a maximum back order.

- 
- 1 Choose  $v_1$  arbitrarily ;
  - 2  $W_1 \leftarrow (v_1)$  ;
  - 3 **for**  $i \leftarrow 1 \dots |V| - 1$  **do**
  - 4     Choose  $v_{i+1} \in \operatorname{argmin}_{u \in V \setminus W_i} (g(W_i \cup \{u\}) - g(u))$  ;
  - 5      $W_{i+1} \leftarrow (W_i, v_{i+1})$  ; /\* Append  $v_{i+1}$  to end of  $W_i$

- Note algorithm operates on non-symmetric function  $g$ . If  $g$  is already symmetric, then  $f = g$ .

# Minimization of a Symmetric Submodular Functions

- Minimizing symmetric submodular functions can be done in strongly polynomial time  $O(n^3)$ . The algorithm is by Nagamochi & Ibaracki 2002 for graph cuts and was shown by Queyranne in 1995 to work for sym. SFM.
- The algorithm first finds a MA (maximum adjacency) or a maximum back order.

- 
- 1 Choose  $v_1$  arbitrarily ;
  - 2  $W_1 \leftarrow (v_1)$  ;
  - 3 **for**  $i \leftarrow 1 \dots |V| - 1$  **do**
  - 4     Choose  $v_{i+1} \in \operatorname{argmin}_{u \in V \setminus W_i} (g(W_i \cup \{u\}) - g(u))$  ;
  - 5      $W_{i+1} \leftarrow (W_i, v_{i+1})$  ; /\* Append  $v_{i+1}$  to end of  $W_i$

- 
- Note algorithm operates on non-symmetric function  $g$ . If  $g$  is already symmetric, then  $f = g$ .
  - 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

$$\{u\} \in \operatorname{argmin} \{f(A) : u \in A, t \notin A\} \quad (15)$$

# 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

$$\{u\} \in \operatorname{argmin} \{f(A) : u \in A, t \notin A\} \quad (15)$$

- That is,

$$f(\{u\}) \leq f(A) \quad \forall A \text{ s.t. } t \notin A \ni u \quad (16)$$

## 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

$$\{u\} \in \operatorname{argmin} \{f(A) : u \in A, t \notin A\} \quad (15)$$

- That is,

$$f(\{u\}) \leq f(A) \quad \forall A \text{ s.t. } t \notin A \ni u \quad (16)$$

### Theorem 5.1

*In the ordered set  $W = (v_1, \dots, v_n)$  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

$$\{u\} \in \operatorname{argmin} \{f(A) : u \in A, t \notin A\} \quad (15)$$

- That is,

$$f(\{u\}) \leq f(A) \quad \forall A \text{ s.t. } t \notin A \ni u \quad (16)$$

## Theorem 5.1

*In the ordered set  $W = (v_1, \dots, v_n)$  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 minimizer, say  $X^*$  of  $f$  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 minimizer, say  $X^*$  of  $f$  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 of the first case, and consider a new element  $tu$  and clustered ground set  $V' = V \setminus \{t, u\} \cup \{tu\}$ , and new symmetric submodular function  $f' : 2^{V'} \rightarrow \mathbb{R}$  with

$$f'(X) = \begin{cases} f(X) & \text{if } tu \notin X \\ f(X \cup \{t, u\} \setminus \{tu\}) & \text{if } tu \in X \end{cases} \quad (17)$$

# Minimization of a Symmetric Submodular Functions

- Now, given a pendent pair  $(t, u)$  there are two cases.
- Either: The minimizer, say  $X^*$  of  $f$  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 of the first case, and consider a new element  $tu$  and clustered ground set  $V' = V \setminus \{t, u\} \cup \{tu\}$ , and new symmetric submodular function  $f' : 2^{V'} \rightarrow \mathbb{R}$  with

$$f'(X) = \begin{cases} f(X) & \text{if } tu \notin X \\ f(X \cup \{t, u\} \setminus \{tu\}) & \text{if } tu \in X \end{cases} \quad (17)$$

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

# Minimization of a Symmetric Submodular Functions

- Now, given a pendent pair  $(t, u)$  there are two cases.
- Either: The minimizer, say  $X^*$  of  $f$  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 of the first case, and consider a new element  $tu$  and clustered ground set  $V' = V \setminus \{t, u\} \cup \{tu\}$ , and new symmetric submodular function  $f' : 2^{V'} \rightarrow \mathbb{R}$  with

$$f'(X) = \begin{cases} f(X) & \text{if } tu \notin X \\ f(X \cup \{t, u\} \setminus \{tu\}) & \text{if } tu \in X \end{cases} \quad (17)$$

- We then find a new pendent pair on  $f'$  using the above algorithm, store the min value, and merge.
- 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 minimizer, say  $X^*$  of  $f$  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 of the first case, and consider a new element  $tu$  and clustered ground set  $V' = V \setminus \{t, u\} \cup \{tu\}$ , and new symmetric submodular function  $f' : 2^{V'} \rightarrow \mathbb{R}$  with

$$f'(X) = \begin{cases} f(X) & \text{if } tu \notin X \\ f(X \cup \{t, u\} \setminus \{tu\}) & \text{if } tu \in X \end{cases} \quad (17)$$

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

# Minimization of a Symmetric Submodular Functions

## Theorem 5.2

*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.

# 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 a polymatroid function, maximization is trivial (take the 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 a polymatroid function, maximization is trivial (take the ground set).
- Thus, when we do submodular max, 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 a polymatroid function, maximization is trivial (take the ground set).
- Thus, when we do submodular max, 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 a polymatroid function, maximization is trivial (take the ground set).
- Thus, when we do submodular max, 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 a polymatroid function, maximization is trivial (take the ground set).
- Thus, when we do submodular max, we either
  - Find the maximum under some constraint
  - Find the maximum for a non-polymatroid submodular function
  - Perhaps 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 a polymatroid function, maximization is trivial (take the ground set).
- Thus, when we do submodular max, we either
  - Find the maximum under some constraint
  - Find the maximum for a non-polymatroid submodular function
  - Perhaps 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 round set and let  $E_1, E_2, \dots, E_m$  be a set of subsets.

# The Set Cover Problem

- Let  $E$  be a round 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.

# The Set Cover Problem

- Let  $E$  be a round 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  $f : 2^V \rightarrow \mathbb{Z}_+$  as  $f(X) = |\bigcup_{v \in X} E_v|$

# The Set Cover Problem

- Let  $E$  be a round 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  $f : 2^V \rightarrow \mathbb{Z}_+$  as  $f(X) = |\bigcup_{v \in V} E_v|$
- Then  $f$  is the set cover function. As we say,  $f$  is monotone submodular (a polymatroid).

# The Set Cover Problem

- Let  $E$  be a round 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  $f : 2^V \rightarrow \mathbb{Z}_+$  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  $f(X) = m$  (smallest subset of the subsets of  $E$  where  $E$  is still covered).  
i.e.,

$$\min |X| \text{ subject to } f(X) \geq |E| \tag{18}$$

# The Set Cover Problem

- Let  $E$  be a round 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  $f : 2^V \rightarrow \mathbb{Z}_+$  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  $f(X) = n$  (smallest subset of the subsets of  $E$  where  $E$  is still covered).  
 i.e.,  $n = |E|$

$$\min |X| \text{ subject to } f(X) \geq |E| \tag{18}$$

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

# The Set Cover Problem

- Let  $E$  be a round 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  $f : 2^V \rightarrow \mathbb{Z}_+$  as  $f(X) = |\bigcup_{v \in V} E_v|$
- 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  $f(X) = m$  (smallest subset of the subsets of  $E$  where  $E$  is still covered).  
I.e.,

$$\min |X| \text{ subject to } f(X) \geq |E| \quad (18)$$

- 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 ( $n^{O(\log \log n)}$ ).

# The Max $k$ -Cover Problem

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

# The Max $k$ -Cover Problem

- Let  $E$  be a round 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.

# The Max $k$ -Cover Problem

- Let  $E$  be a round 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  $f : 2^V \rightarrow \mathbb{Z}_+$  as  $f(X) = |\bigcup_{v \in V} E_v|$

# The Max $k$ -Cover Problem

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

$$\max f(X) \text{ subject to } |X| \leq k \tag{19}$$

# 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  $f : 2^V \rightarrow \mathbb{Z}_+$  as  $f(X) = |\bigcup_{v \in X} E_v|$
- 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:
 
$$\max f(X) \text{ subject to } |X| \leq k \tag{19}$$
- 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

$$S_{i+1} = S_i \cup \left\{ \operatorname{argmax}_{v \in V \setminus S_i} f(S_i \cup \{v\}) \right\} \quad (20)$$

# Cardinality Constrained Max. of Polymatroid Functions

- This algorithm has a guarantee

## Theorem 6.1

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  $k = i + 1$ :
  - $\approx 0.63$
- 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  $\epsilon > 0$ .

# Minimum Submodular Cover

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

$$S^* \in \operatorname{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \quad (21)$$

where  $\alpha$  is a “cover” requirement.

# Minimum Submodular Cover

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

$$S^* \in \underset{S \subseteq V}{\operatorname{argmin}} |S| \text{ such that } f(S) \geq \alpha \tag{21}$$

where  $\alpha$  is a “cover” requirement.

- Normally take  $\alpha = f(V)$  but defining  $f'(A) = \min \{f(A), \alpha\}$  we can take any  $\alpha$ .

$$\alpha = .75 \cdot f(V)$$

# Minimum Submodular Cover

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

$$S^* \in \operatorname{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \tag{21}$$

where  $\alpha$  is a “cover” requirement.

- Normally take  $\alpha = f(V)$  but defining  $f'(A) = \min \{f(A), \alpha\}$  we can take any  $\alpha$ .
- 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. That is:

$$S^* \in \operatorname{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \quad (21)$$

where  $\alpha$  is a “cover” requirement.

- Normally take  $\alpha = f(V)$  but defining  $f'(A) = \min \{f(A), \alpha\}$  we can take any  $\alpha$ .
- 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. That is:

$$S^* \in \operatorname{argmin}_{S \subseteq V} |S| \text{ such that } f(S) \geq \alpha \quad (21)$$

where  $\alpha$  is a “cover” requirement.

- Normally take  $\alpha = f(V)$  but defining  $f'(A) = \min \{f(A), \alpha\}$  we can take any  $\alpha$ .
- 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  $O(\log(\max_{S \subseteq V} f(\{s\})))$  approximation. Set cover is hard to approximate with a factor better than  $(1 - \epsilon) \log \alpha$ , where  $\alpha$  is the desired cover constraint.

# Generalizations

- There are a number of ways of generalizing these results.

h

# Generalizations

- There are a number of ways of generalizing these results.
- First, we can use more general constraints, rather than just cardinality constraints.

h

# Generalizations

- There are a number of ways of generalizing these results.
- First, we can use more general constraints, rather than just cardinality constraints.
- 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.

h

# Generalizations

- There are a number of ways of generalizing these results.
- First, we can use more general constraints, rather than just cardinality constraints.
- 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.
- We may wish to do both.

h

# Generalizations

- There are a number of ways of generalizing these results.
- First, we can use more general constraints, rather than just cardinality constraints.
- 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.
- We may wish to do both.
- In either case, 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 unlikely event were to be true, such as  $P=NP$ ).

h

# Submodular max with constraints

- Consider a set of sets  $\mathcal{I}$  and one wishes to find  $\max \{f(A) : A \in \mathcal{I}\}$

# Submodular max with constraints

- Consider a set of sets  $\mathcal{I}$  and one wishes to find  $\max \{f(A) : A \in \mathcal{I}\}$
- In fact, if  $M = (E, \mathcal{I})$  is a  $k$ -uniform matroid we saw in Lecture 3 ( $\mathcal{I} = \{A \subseteq E : |A| \leq k\}$ ), we have the cardinality constrained submodular max we just encountered.

# Submodular max with constraints

- Consider a set of sets  $\mathcal{I}$  and one wishes to find  $\max \{f(A) : A \in \mathcal{I}\}$
- In fact, if  $M = (E, \mathcal{I})$  is a  $k$ -uniform matroid we saw in Lecture 3 ( $\mathcal{I} = \{A \subseteq E : |A| \leq k\}$ ), we have the cardinality constrained submodular max we just encountered.
- Might be useful to allow an arbitrary matroid (e.g., partition matroid  $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \leq k_i \text{ for all } i = 1, \dots, \ell\}$ ., or a transversal, etc).

# Submodular max with constraints

- Consider a set of sets  $\mathcal{I}$  and one wishes to find  $\max \{f(A) : A \in \mathcal{I}\}$
- In fact, if  $M = (E, \mathcal{I})$  is a  $k$ -uniform matroid we saw in Lecture 3 ( $\mathcal{I} = \{A \subseteq E : |A| \leq k\}$ ), we have the cardinality constrained submodular max we just encountered.
- Might be useful to allow an arbitrary matroid (e.g., partition matroid  $\mathcal{I} = \{X \subseteq V : |X \cap V_i| \leq k_i \text{ for all } i = 1, \dots, \ell\}$ , or a transversal, etc).
- In fact, one could consider multiple matroids  $M_1, M_2, \dots, M_p$  and one may want a solution that is independent in all matroids. I.e., the constraint set is that  $A \in \mathcal{I}_1 \cap \mathcal{I}_2 \cap \dots \cap \mathcal{I}_p$

# 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

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i : S_i + v \in \bigcap_{i=1}^p \mathcal{I}_i}{\operatorname{argmax}} f(S_i \cup \{v\}) \right\} \quad (22)$$

# 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

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i : S_i + v \in \bigcap_{i=1}^p \mathcal{I}_i}{\operatorname{argmax}} f(S_i \cup \{v\}) \right\} \quad (22)$$

- 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

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i : S_i + v \in \bigcap_{i=1}^p \mathcal{I}_i}{\operatorname{argmax}} f(S_i \cup \{v\}) \right\} \quad (22)$$

- 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

$$\rightarrow S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i : S_i + v \in \bigcap_{i=1}^p \mathcal{I}_i}{\operatorname{argmax}} f(S_i \cup \{v\}) \right\} \quad (22)$$

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

### Theorem 6.2

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 a set exists.

$S_i$

## 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

$$S_{i+1} = S_i \cup \left\{ \underset{v \in V \setminus S_i : S_i + v \in \bigcap_{i=1}^p \mathcal{I}_i}{\operatorname{argmax}} f(S_i \cup \{v\}) \right\} \quad (22)$$

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

### Theorem 6.2

Given a polymatroid function  $f$ , and set of matroids  $\{M_i = (E, \mathcal{I}_i)\}_{i=1}^p$ , 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 a set exists.

- So this is a very easy algorithm to solve multiple matroid constraints, but the bound is not that good when there are many matroids.

# 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  $c : E \rightarrow \mathbb{Z}_+$ .

# Monotone Submodular over Knapsack Constraint

- The constraint  $|A| \leq k$  is a simple cardinality constraint.
- Consider a non-negative integral modular function  $c : E \rightarrow \mathbb{Z}_+$ .
- 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  $c : E \rightarrow \mathbb{Z}_+$ .
- 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  $\max \{f(A) : A \subseteq V, c(A) \leq b\}$ .
- $c(e)$  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

$$S_{i+1} = S_i \cup \left\{ \operatorname{argmax}_{v \in V \setminus S_i} \left( f(S_i \cup \{v\}) - f(S_i) \right) \right\} \quad (23)$$

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

$$S_{i+1} = S_i \cup \left\{ \operatorname{argmax}_{v \in V \setminus S_i} \left( f(S_i \cup \{v\}) - f(S_i) \right) \right\} \quad (23)$$

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

$$S_{i+1} = S_i \cup \left\{ \operatorname{argmax}_{v \in V \setminus S_i} \frac{f(S_i \cup \{v\}) - f(S_i)}{c(v)} \right\} \quad (24)$$

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.
- 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 this to  $d$  simultaneous knapsack constraints, it's possible (via continuous extension), to

# Accelerated Greedy

- Greedy can be made much faster by a simple strategy made possible, once again, via the use of submodularity.

# Accelerated Greedy

- Greedy can be made much faster by a simple strategy made possible, once again, via the use of submodularity.
- 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  $\alpha_v = f(v|S_i)$ .

# Accelerated Greedy

- Greedy can be made much faster by a simple strategy made possible, once again, via the use of submodularity.
- 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  $\alpha_v = f(v|S_i)$ .
- Once we choose a  $v$ , then  $S_{i+1} = S_i + v$ .

# Accelerated Greedy

- Greedy can be made much faster by a simple strategy made possible, once again, via the use of submodularity.
- 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  $\alpha_v = \underline{f(v|S_i)}$ .  $\hat{=} f(S_i + v) - f(S_i)$
- Once we choose a  $v$ , then  $S_{i+1} = S_i + v$ .
- For  $v \notin S_{i+1}$  we have  $f(v|S_{i+1}) \leq f(v|S_i)$ .

# Accelerated Greedy

- Greedy can be made much faster by a simple strategy made possible, once again, via the use of submodularity.
- 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  $\alpha_v = f(v|S_i)$ .
- Once we choose a  $v$ , then  $S_{i+1} = S_i + v$ .
- For  $v \notin S_{i+1}$  we have  $f(v|S_{i+1}) \leq f(v|S_i)$ .
- Therefore, if we find a  $v'$  such that  $f(v'|S_{i+1}) \geq \alpha_v$  for all  $v \neq v'$ , then since  $f(v|S_{i+1}) \leq \alpha_v$ , we need not re-evaluate the gain.

$$f(v'|S_{i+1}) \geq f(v|S_{i+1}) \quad v \neq v'$$

# Accelerated Greedy

- Greedy can be made much faster by a simple strategy made possible, once again, via the use of submodularity.
- 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  $\alpha_v = f(v|S_i)$ .
- Once we choose a  $v$ , then  $S_{i+1} = S_i + v$ .
- For  $v \notin S_{i+1}$  we have  $f(v|S_{i+1}) \leq f(v|S_i)$ .
- Therefore, if we find a  $v'$  such that  $f(v'|S_{i+1}) \geq \alpha_v$  for all  $v \neq v'$ , then since  $f(v|S_{i+1}) \leq \alpha_v$ , we need not re-evaluate the gain.
- Strategy is: find the  $\max \alpha_{v'}$ , and then compute the real  $f(v'|S_{i+1})$ . If it is greater than all other  $\alpha_v$ 's then that's the next greedy step. Otherwise, replace  $\alpha_{v'}$  and repeat.

# Accelerated Greedy

- Greedy can be made much faster by a simple strategy made possible, once again, via the use of submodularity.
- 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  $\alpha_v = f(v|S_i)$ .
- Once we choose a  $v$ , then  $S_{i+1} = S_i + v$ .
- For  $v \notin S_{i+1}$  we have  $f(v|S_{i+1}) \leq f(v|S_i)$ .
- Therefore, if we find a  $v'$  such that  $f(v'|S_{i+1}) \geq \alpha_v$  for all  $v \neq v'$ , then since  $f(v|S_{i+1}) \leq \alpha_v$ , we need not re-evaluate the gain.
- Strategy is: find the  $\max \alpha_{v'}$ , and then compute the real  $f(v'|S_{i+1})$ . If it is greater than all other  $\alpha_v$ 's then that's the next greedy step. Otherwise, replace  $\alpha_{v'}$  and repeat.
- In practice, this results in enormous speedups in the greedy procedure (and has been used for selecting blogs of greatest influence).

# What About Non-monotone

- 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

- 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

- 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

- 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.
- It is possible to get a  $(\frac{1}{3} - \frac{\epsilon}{n})$  approximation for maximizing non-monotone non-negative submodular functions, using an algorithm that uses at most  $O(\frac{1}{\epsilon} n^3 \log n)$  function calls.

# 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 6.3

*Given a submodular function  $f$ , if  $S$  is a local optimum 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 6.3

Given a submodular function  $f$ , if  $S$  is a local optimum 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 optimum, 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 6.3

*Given a submodular function  $f$ , if  $S$  is a local optimum 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 optimum, 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 optimum is already hard (PLS-complete), but it is possible to find an approximate local optimum 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 6.3

*Given a submodular function  $f$ , if  $S$  is a local optimum 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 optimum, 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 optimum is already hard (PLS-complete), but it is possible to find an approximate local optimum relatively efficiently.
- This is the approach that yields the  $(\frac{1}{3} - \frac{\epsilon}{n})$  approximation algorithm.

## 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.

# 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.
- Very recently, a paper by Chekuri, Vondrak, and Zenklusen (2011, appeared yesterday in fact) make some progress on this front using multilinear extensions.

# Multilinear extension

## Definition 6.4

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

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \quad (25)$$

# Multilinear extension

## Definition 6.4

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

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \quad (25)$$

- 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 6.4

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

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \tag{25}$$

- 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 6.4

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

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \quad (25)$$

- 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 6.5

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

# Multilinear extension

## Definition 6.4

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

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \quad (25)$$

- 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 6.5

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  $\frac{\partial F}{\partial x_i} \geq 0$  for all  $i \in V$ ,  $x \in [0, 1]^V$ .

# Multilinear extension

## Definition 6.4

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

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{j \in V \setminus S} (1 - x_j) \quad (25)$$

- 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 6.5

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  $\frac{\partial F}{\partial x_i} \geq 0$  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  $i, j \in V$ ,  $x \in [0, 1]^V$ .

# Multilinear extension

- Moreover, we have

# Multilinear extension

- Moreover, we have

## Lemma 6.6

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

# Multilinear extension

- Moreover, we have

## Lemma 6.6

*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 6.6

*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  $d \geq 0$*
- *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 6.6

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  $d \geq 0$
- 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 6.6

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  $d \geq 0$
- 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 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 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  $\ell$  sparse packing integer programs, and unsplittable flow in paths and trees.

## Where next?

- Submodular functions are not new, but on the other hand are gaining in popularity.

# Where next?

- Submodular functions are not new, but on the other hand are gaining in popularity.
- Many of the papers on submodular max with constraints have occurred within the last 3 years.

## Where next?

- Submodular functions are not new, but on the other hand are gaining in popularity.
- Many of the papers on submodular max with constraints have occurred within the last 3 years.
- There have been many SFM papers in the past 10 years.

## Where next?

- Submodular functions are not new, but on the other hand are gaining in popularity.
- Many of the papers on submodular max with constraints have occurred within the last 3 years.
- There have been many SFM papers in the past 10 years.
- Many new problems associated with submodularity are becoming popular.

# Where next?

- Submodular functions are not new, but on the other hand are gaining in popularity.
- Many of the papers on submodular max with constraints have occurred within the last 3 years.
- There have been many SFM papers in the past 10 years.
- Many new problems associated with submodularity are becoming popular.
- Combinatorial algorithms with submodular rather than modular costs (e.g., spanning trees, matchings, paths, cuts, etc.) are becoming popular.

# Where next?

- Submodular functions are not new, but on the other hand are gaining in popularity.
- Many of the papers on submodular max with constraints have occurred within the last 3 years.
- There have been many SFM papers in the past 10 years.
- Many new problems associated with submodularity are becoming popular.
- Combinatorial algorithms with submodular rather than modular costs (e.g., spanning trees, matchings, paths, cuts, etc.) are becoming popular.
- Many new applications of submodular functions in machine learning.

# Where next?

- Submodular functions are not new, but on the other hand are gaining in popularity.
- Many of the papers on submodular max with constraints have occurred within the last 3 years.
- There have been many SFM papers in the past 10 years.
- Many new problems associated with submodularity are becoming popular.
- Combinatorial algorithms with submodular rather than modular costs (e.g., spanning trees, matchings, paths, cuts, etc.) are becoming popular.
- Many new applications of submodular functions in machine learning.
- This course has served as a thorough introduction to many important aspects of submodular functions.

# Scratch Paper

# Scratch Paper

# Scratch Paper

# Sources for Today's Lecture

- 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.
- Jegelka & Bilmes, "Approximate Probabilistic Inference via Generalized Graph Cuts", 2011.
- Lin & Bilmes, "Approximate Probabilistic Inference via Generalized Graph Cuts", 2011.