Introduction to Submodular Functions

S. Thomas McCormick   Satoru Iwata

Sauder School of Business, UBC
Cargese Workshop on Combinatorial Optimization,
Sept–Oct 2013

Teaching plan

- First hour: Tom McCormick on submodular functions

Teaching plan

- First hour: Tom McCormick on submodular functions
- Next half hour: Satoru Iwata on Lovàsz extension

Teaching plan

- First hour: Tom McCormick on submodular functions
- Next half hour: Satoru Iwata on Lovàsz extension
- Later: Tom, Satoru, Francis, Seffi on more advanced topics

Contents

Contents

Outline

Motivating "business school" example

- Suppose that you manage a factory that is capable of making any one of a large finite set E of products.

Motivating “business school” example

- Suppose that you manage a factory that is capable of making any one of a large finite set E of products.
- In order to produce product $e \in E$ it is necessary to set up the machines needed to manufacture e, and this costs money.

Motivating "business school" example

- Suppose that you manage a factory that is capable of making any one of a large finite set E of products.
- In order to produce product $e \in E$ it is necessary to set up the machines needed to manufacture e, and this costs money.
- The setup cost is non-linear, and it depends on which other products you choose to produce.

Motivating “business school” example

- Suppose that you manage a factory that is capable of making any one of a large finite set E of products.
- In order to produce product $e \in E$ it is necessary to set up the machines needed to manufacture e, and this costs money.
- The setup cost is non-linear, and it depends on which other products you choose to produce.
  - For example, if you are already producing iPhones, then the setup cost for also producing iPads is small, but if you are not producing iPhones, the setup cost for producing iPads is large.

Motivating "business school" example

- Suppose that you manage a factory that is capable of making any one of a large finite set E of products.
- In order to produce product $e \in E$ it is necessary to set up the machines needed to manufacture e, and this costs money.
- The setup cost is non-linear, and it depends on which other products you choose to produce.
  - For example, if you are already producing iPhones, then the setup cost for also producing iPads is small, but if you are not producing iPhones, the setup cost for producing iPads is large.
- Suppose that we choose to produce the subset of products $S \subseteq E$. Then we write the setup cost of subset S as $c(S)$.

Set Functions

- Notice that $c(S)$ is a function from 2^E (the family of all subsets of E) to $\mathbb{R}$.

Set Functions

- Notice that $c(S)$ is a function from 2^E (the family of all subsets of E) to $\mathbb{R}$.
- If f is a function $f : 2^E \to \mathbb{R}$ then we call f a set function.

Set Functions

- Notice that $c(S)$ is a function from 2^E (the family of all subsets of E) to $\mathbb{R}$.
- If f is a function $f : 2^E \rightarrow \mathbb{R}$ then we call f a set function.
- We globally use n to denote $|E|$. Thus a set function f on E is determined by its 2^n values $f(S)$ for $S \subseteq E$.

Set Functions

- Notice that $c(S)$ is a function from 2^E (the family of all subsets of E) to $\mathbb{R}$.
- If f is a function $f : 2^E \rightarrow \mathbb{R}$ then we call f a set function.
- We globally use n to denote $|E|$. Thus a set function f on E is determined by its 2^n values $f(S)$ for $S \subseteq E$.
- This is *a lot* of data. We typically have some more compact representation of f that allows us to efficiently compute $f(S)$ for a given S.

Set Functions

- Notice that $c(S)$ is a function from 2^E (the family of all subsets of E) to $\mathbb{R}$.
- If f is a function $f : 2^E \to \mathbb{R}$ then we call f a set function.
- We globally use n to denote $|E|$. Thus a set function f on E is determined by its 2^n values $f(S)$ for $S \subseteq E$.
- This is *a lot* of data. We typically have some more compact representation of f that allows us to efficiently compute $f(S)$ for a given S.
- Because of this, we talk about set functions using an value oracle model: we assume that we have an algorithm $\mathcal{E}$ whose input is some $S \subseteq E$, and whose output is $f(S)$. We denote the running time of $\mathcal{E}$ by EO.

Set Functions

- Notice that $c(S)$ is a function from 2^E (the family of all subsets of E) to $\mathbb{R}$.
- If f is a function $f : 2^E \to \mathbb{R}$ then we call f a set function.
- We globally use n to denote $|E|$. Thus a set function f on E is determined by its 2^n values $f(S)$ for $S \subseteq E$.
- This is *a lot* of data. We typically have some more compact representation of f that allows us to efficiently compute $f(S)$ for a given S.
- Because of this, we talk about set functions using an value oracle model: we assume that we have an algorithm $\mathcal{E}$ whose input is some $S \subseteq E$, and whose output is $f(S)$. We denote the running time of $\mathcal{E}$ by EO.
  - We typically think that $\mathrm{EO} = \Omega(n)$, i.e., that it takes at least linear time to evaluate f on S.

Back to the motivating example

- We have setup cost set function $c : 2^E \to \mathbb{R}$.

Back to the motivating example

- We have setup cost set function $c : 2^E \to \mathbb{R}$.
- Imagine that we are currently producing subset S, and we are considering also producing product e for $e \notin S$.

Back to the motivating example

- We have setup cost set function $c : 2^E \to \mathbb{R}$.
- Imagine that we are currently producing subset S, and we are considering also producing product e for $e \notin S$.
- The marginal setup cost for adding e to S is $c(S \cup \{e\}) - c(S)$.

Back to the motivating example

- We have setup cost set function $c : 2^E \to \mathbb{R}$.
- Imagine that we are currently producing subset S, and we are considering also producing product e for $e \notin S$.
- The marginal setup cost for adding e to S is $c(S \cup \{e\}) - c(S)$.
  - To simplify notation we often write $c(S \cup \{e\})$ as $c(S + e)$.

Back to the motivating example

- We have setup cost set function $c : 2^E \to \mathbb{R}$.
- Imagine that we are currently producing subset S, and we are considering also producing product e for $e \notin S$.
- The marginal setup cost for adding e to S is $c(S \cup \{e\}) - c(S)$.
  - To simplify notation we often write $c(S \cup \{e\})$ as $c(S + e)$.
- In this notation the marginal setup cost is $c(S + e) - c(S)$.

Back to the motivating example

- We have setup cost set function $c : 2^E \to \mathbb{R}$.
- Imagine that we are currently producing subset S, and we are considering also producing product e for $e \notin S$.
- The marginal setup cost for adding e to S is $c(S \cup \{e\}) - c(S)$.
  - To simplify notation we often write $c(S \cup \{e\})$ as $c(S + e)$.
- In this notation the marginal setup cost is $c(S + e) - c(S)$.
- Suppose that $S \subset T$ and that $e \notin T$. Since T includes everything in S and more, it is reasonable to guess that the marginal setup cost of adding e to T is not larger than the marginal setup cost of adding e to S. That is,

$$\forall S \subset T \subset T + e, \; c(T + e) - c(T) \leq c(S + e) - c(S). \tag{1}$$

Back to the motivating example

- We have setup cost set function $c : 2^E \to \mathbb{R}$.
- Imagine that we are currently producing subset S, and we are considering also producing product e for $e \notin S$.
- The marginal setup cost for adding e to S is $c(S \cup \{e\}) - c(S)$.
  - To simplify notation we often write $c(S \cup \{e\})$ as $c(S + e)$.
- In this notation the marginal setup cost is $c(S + e) - c(S)$.
- Suppose that $S \subset T$ and that $e \notin T$. Since T includes everything in S and more, it is reasonable to guess that the marginal setup cost of adding e to T is not larger than the marginal setup cost of adding e to S. That is,

$$\forall S \subset T \subset T + e, \; c(T + e) - c(T) \leq c(S + e) - c(S). \tag{1}$$

- When a set function satisfies (1) we say that it is submodular.

Outline

Submodularity definitions

- In general, if f is a set function on E, we say that f is submodular if

$$\forall S \subset T \subset T + e,\ f(T+e) - f(T) \leq f(S+e) - f(S). \quad (2)$$

Submodularity definitions

- In general, if f is a set function on E, we say that f is submodular if

$$\forall S \subset T \subset T + e, \; f(T + e) - f(T) \leq f(S + e) - f(S). \quad (2)$$

- The classic definition of submodularity looks quite different. We also say that set function f is submodular if

$$\text{for all } S, T \subseteq E, \; f(S) + f(T) \geq f(S \cup T) + f(S \cap T). \quad (3)$$

Submodularity definitions

- In general, if f is a set function on E, we say that f is submodular if

$$\forall S \subset T \subset T + e,\ f(T+e) - f(T) \le f(S+e) - f(S). \quad (2)$$

- The classic definition of submodularity looks quite different. We also say that set function f is submodular if

$$\text{for all } S, T \subseteq E,\ f(S) + f(T) \ge f(S \cup T) + f(S \cap T). \quad (3)$$

Lemma

Definitions (2) and (3) are equivalent.

Submodularity definitions

- In general, if f is a set function on E, we say that f is submodular if

$$\forall S \subset T \subset T + e,\ f(T+e) - f(T) \leq f(S+e) - f(S). \quad (2)$$

- The classic definition of submodularity looks quite different. We also say that set function f is submodular if

$$\text{for all } S, T \subseteq E,\ f(S) + f(T) \geq f(S \cup T) + f(S \cap T). \quad (3)$$

Lemma
Definitions (2) and (3) are equivalent.

Proof.
Homework. □

More definitions

- We say that set function f is monotone if $S \subseteq T$ implies that $f(S) \leq f(T)$.

More definitions

- We say that set function f is monotone if $S \subseteq T$ implies that $f(S) \leq f(T)$.
  - Many set functions arising in applications are monotone, but not all of them.

More definitions

- We say that set function f is monotone if $S \subseteq T$ implies that $f(S) \leq f(T)$.
  - Many set functions arising in applications are monotone, but not all of them.
- A set function that is both submodular and monotone is called a polymatroid.

More definitions

- We say that set function f is monotone if $S \subseteq T$ implies that $f(S) \leq f(T)$.
  - Many set functions arising in applications are monotone, but not all of them.
- A set function that is both submodular and monotone is called a polymatroid.
  - Polymatroids generalize matroids, and are a special case of the submodular polyhedra we'll see later.

Even more definitions

- We say that set function f is supermodular if it satisfies these definitions with the inequalities reversed, i.e., if

$$\forall S \subset T \subset T + e, \; f(T + e) - f(T) \geq f(S + e) - f(S). \tag{4}$$

Thus f is supermodular iff $-f$ is submodular.

Even more definitions

- We say that set function f is supermodular if it satisfies these definitions with the inequalities reversed, i.e., if

$$\forall S \subset T \subset T + e, \; f(T + e) - f(T) \geq f(S + e) - f(S). \tag{4}$$

Thus f is supermodular iff $-f$ is submodular.

- We say that set function f is modular if it satisfies these definitions with equality, i.e., if

$$\forall S \subset T \subset T + e, \; f(T + e) - f(T) = f(S + e) - f(S). \tag{5}$$

Thus f is modular iff it is both sub- and supermodular.

Even more definitions

- We say that set function f is supermodular if it satisfies these definitions with the inequalities reversed, i.e., if

$$\forall S \subset T \subset T + e,\ f(T+e) - f(T) \geq f(S+e) - f(S). \quad (4)$$

Thus f is supermodular iff $-f$ is submodular.

- We say that set function f is modular if it satisfies these definitions with equality, i.e., if

$$\forall S \subset T \subset T + e,\ f(T+e) - f(T) = f(S+e) - f(S). \quad (5)$$

Thus f is modular iff it is both sub- and supermodular.

Lemma

Set function f is modular iff there is some vector $a \in \mathbb{R}^E$ such that $f(S) = f(\emptyset) + \sum_{e \in S} a_e$.

Even more definitions

- We say that set function f is supermodular if it satisfies these definitions with the inequalities reversed, i.e., if

$$\forall S \subset T \subset T + e, \; f(T + e) - f(T) \geq f(S + e) - f(S). \quad (4)$$

  Thus f is supermodular iff $-f$ is submodular.

- We say that set function f is modular if it satisfies these definitions with equality, i.e., if

$$\forall S \subset T \subset T + e, \; f(T + e) - f(T) = f(S + e) - f(S). \quad (5)$$

  Thus f is modular iff it is both sub- and supermodular.

Lemma

Set function f is modular iff there is some vector $a \in \mathbb{R}^E$ such that $f(S) = f(\emptyset) + \sum_{e \in S} a_e$.

Proof.

Homework. $\square$

Motivating example again

- The lemma suggest a natural way to extend a vector $a \in \mathbb{R}^E$ to a modular set function: Define $a(S) = \sum_{e \in S} a_e$. Note that $a(\emptyset) = 0$. (Queyranne: "$a \cdot S$" is better notation?)

Motivating example again

- The lemma suggest a natural way to extend a vector $a \in \mathbb{R}^E$ to a modular set function: Define $a(S) = \sum_{e \in S} a_e$. Note that $a(\emptyset) = 0$. (Queyranne: "$a \cdot S$" is better notation?)
- For example, let's suppose that the profit from producing product $e \in E$ is p_e, i.e., $p \in \mathbb{R}^E$.

Motivating example again

- The lemma suggest a natural way to extend a vector $a \in \mathbb{R}^E$ to a modular set function: Define $a(S) = \sum_{e \in S} a_e$. Note that $a(\emptyset) = 0$. (Queyranne: "$a \cdot S$" is better notation?)
- For example, let's suppose that the profit from producing product $e \in E$ is p_e, i.e., $p \in \mathbb{R}^E$.
- We assume that these profits add up linearly, so that the profit from producing subset S is $p(S) = \sum_{e \in E} p_e$.

Motivating example again

- The lemma suggest a natural way to extend a vector $a \in \mathbb{R}^E$ to a modular set function: Define $a(S) = \sum_{e \in S} a_e$. Note that $a(\emptyset) = 0$. (Queyranne: "$a \cdot S$" is better notation?)
- For example, let's suppose that the profit from producing product $e \in E$ is p_e, i.e., $p \in \mathbb{R}^E$.
- We assume that these profits add up linearly, so that the profit from producing subset S is $p(S) = \sum_{e \in E} p_e$.
- Therefore our net revenue from producing subset S is $p(S) - c(S)$, which is a supermodular set function (why?).

Motivating example again

- The lemma suggest a natural way to extend a vector $a \in \mathbb{R}^E$ to a modular set function: Define $a(S) = \sum_{e \in S} a_e$. Note that $a(\emptyset) = 0$. (Queyranne: "$a \cdot S$" is better notation?)
- For example, let's suppose that the profit from producing product $e \in E$ is p_e, i.e., $p \in \mathbb{R}^E$.
- We assume that these profits add up linearly, so that the profit from producing subset S is $p(S) = \sum_{e \in E} p_e$.
- Therefore our net revenue from producing subset S is $p(S) - c(S)$, which is a supermodular set function (why?).
  - Notice that the similar notations "$c(S)$" and "$p(S)$" mean different things here: $c(S)$ really is a set function, whereas $p(S)$ is an artificial set function derived from a vector $p \in \mathbb{R}^E$.

Motivating example again

- The lemma suggest a natural way to extend a vector $a \in \mathbb{R}^E$ to a modular set function: Define $a(S) = \sum_{e \in S} a_e$. Note that $a(\emptyset) = 0$. (Queyranne: "$a \cdot S$" is better notation?)
- For example, let's suppose that the profit from producing product $e \in E$ is p_e, i.e., $p \in \mathbb{R}^E$.
- We assume that these profits add up linearly, so that the profit from producing subset S is $p(S) = \sum_{e \in E} p_e$.
- Therefore our net revenue from producing subset S is $p(S) - c(S)$, which is a supermodular set function (why?).
  - Notice that the similar notations "$c(S)$" and "$p(S)$" mean different things here: $c(S)$ really is a set function, whereas $p(S)$ is an artificial set function derived from a vector $p \in \mathbb{R}^E$.
- In this example we naturally want to find a subset to produce that maximizes our net revenue, i.e, to solve $\max_{S \subseteq E}(p(S) - c(S))$, or equivalently

$$\min_{S \subseteq E}(c(S) - p(S)).$$

More examples of submodularity

- Let $G = (N, A)$ be a directed graph. For $S \subseteq N$ define $\delta^+(S) = \{i \to j \in A \mid i \in S,\ j \notin S\}$, $\delta^-(S) = \{i \to j \in A \mid i \notin S,\ j \in S\}$. Then $|\delta^+(S)|$ and $|\delta^-(S)|$ are submodular.

More examples of submodularity

- Let $G = (N, A)$ be a directed graph. For $S \subseteq N$ define $\delta^+(S) = \{i \to j \in A \mid i \in S,\ j \notin S\}$, $\delta^-(S) = \{i \to j \in A \mid i \notin S,\ j \in S\}$. Then $|\delta^+(S)|$ and $|\delta^-(S)|$ are submodular.
- More generally, suppose that $w \in \mathbb{R}^A$ are weights on the arcs. If $w \geq 0$, then $w(\delta^+(S))$ and $w(\delta^-(S))$ are submodular, and if $w \not\geq 0$ then they are not necessarily submodular (homework).

More examples of submodularity

- Let $G = (N, A)$ be a directed graph. For $S \subseteq N$ define $\delta^+(S) = \{i \to j \in A \mid i \in S,\ j \notin S\}$, $\delta^-(S) = \{i \to j \in A \mid i \notin S,\ j \in S\}$. Then $|\delta^+(S)|$ and $|\delta^-(S)|$ are submodular.
- More generally, suppose that $w \in \mathbb{R}^A$ are weights on the arcs. If $w \geq 0$, then $w(\delta^+(S))$ and $w(\delta^-(S))$ are submodular, and if $w \not\geq 0$ then they are not necessarily submodular (homework).
  - The same is true for *undirected* graphs where we consider $\delta(S) = \{i — j \mid i \in S,\ j \notin S\}$.

More examples of submodularity

- Let $G = (N, A)$ be a directed graph. For $S \subseteq N$ define $\delta^+(S) = \{i \to j \in A \mid i \in S,\ j \notin S\}$, $\delta^-(S) = \{i \to j \in A \mid i \notin S,\ j \in S\}$. Then $|\delta^+(S)|$ and $|\delta^-(S)|$ are submodular.
- More generally, suppose that $w \in \mathbb{R}^A$ are weights on the arcs. If $w \geq 0$, then $w(\delta^+(S))$ and $w(\delta^-(S))$ are submodular, and if $w \not\geq 0$ then they are not necessarily submodular (homework).
  - The same is true for *undirected* graphs where we consider $\delta(S) = \{i - j \mid i \in S,\ j \notin S\}$.
  - Here, e.g., $w(\delta^+(\emptyset)) = 0$.

More examples of submodularity

- Let $G = (N, A)$ be a directed graph. For $S \subseteq N$ define $\delta^+(S) = \{i \to j \in A \mid i \in S,\ j \notin S\}$, $\delta^-(S) = \{i \to j \in A \mid i \notin S,\ j \in S\}$. Then $|\delta^+(S)|$ and $|\delta^-(S)|$ are submodular.
- More generally, suppose that $w \in \mathbb{R}^A$ are weights on the arcs. If $w \geq 0$, then $w(\delta^+(S))$ and $w(\delta^-(S))$ are submodular, and if $w \not\geq 0$ then they are not necessarily submodular (homework).
  - The same is true for *undirected* graphs where we consider $\delta(S) = \{i \,—\, j \mid i \in S,\ j \notin S\}$.
  - Here, e.g., $w(\delta^+(\emptyset)) = 0$.
- Now specialize the previous example slightly to Max Flow / Min Cut: Let $N = \{s\} \cup \{t\} \cup E$ be the node set with source s and sink t. We have arc capacities $u \in \mathbb{R}^A_+$, i.e., arc $i \to j$ has capacity $u_{ij} \geq 0$. An s–t cut is some $S \subseteq E$, and the capacity of cut S is $\text{cap}(S) = u(\delta^+(S + s))$, which is submodular.

More examples of submodularity

- Let $G = (N, A)$ be a directed graph. For $S \subseteq N$ define $\delta^+(S) = \{i \to j \in A \mid i \in S,\ j \notin S\}$, $\delta^-(S) = \{i \to j \in A \mid i \notin S,\ j \in S\}$. Then $|\delta^+(S)|$ and $|\delta^-(S)|$ are submodular.
- More generally, suppose that $w \in \mathbb{R}^A$ are weights on the arcs. If $w \geq 0$, then $w(\delta^+(S))$ and $w(\delta^-(S))$ are submodular, and if $w \not\geq 0$ then they are not necessarily submodular (homework).
  - The same is true for *undirected* graphs where we consider $\delta(S) = \{i - j \mid i \in S,\ j \notin S\}$.
  - Here, e.g., $w(\delta^+(\emptyset)) = 0$.
- Now specialize the previous example slightly to Max Flow / Min Cut: Let $N = \{s\} \cup \{t\} \cup E$ be the node set with source s and sink t. We have arc capacities $u \in \mathbb{R}^A_+$, i.e., arc $i \to j$ has capacity $u_{ij} \geq 0$. An s–t cut is some $S \subseteq E$, and the capacity of cut S is $\text{cap}(S) = u(\delta^+(S + s))$, which is submodular.
  - Here $\text{cap}(\emptyset) = \sum_{e \in E} u_{se}$ is usually positive.

Outline

Max Flow / Min Cut

- Review: Vector $x \in \mathbb{R}^A$ is a feasible flow if it satisfies

Max Flow / Min Cut

- Review: Vector $x \in \mathbb{R}^A$ is a feasible flow if it satisfies
  1. Conservation: $x(\delta^+(\{i\}) = x(\delta^-(\{i\})$ for all $i \in E$, i.e., flow out = flow in.

Max Flow / Min Cut

- Review: Vector $x \in \mathbb{R}^A$ is a feasible flow if it satisfies
  1. Conservation: $x(\delta^+(\{i\}) = x(\delta^-(\{i\})$ for all $i \in E$, i.e., flow out $=$ flow in.
  2. Boundedness: $0 \leq x_{ij} \leq u_{ij}$ for all $i \to j \in A$.

Max Flow / Min Cut

- Review: Vector $x \in \mathbb{R}^A$ is a feasible flow if it satisfies
  1. Conservation: $x(\delta^+(\{i\}) = x(\delta^-(\{i\})$ for all $i \in E$, i.e., flow out = flow in.
  2. Boundedness: $0 \leq x_{ij} \leq u_{ij}$ for all $i \to j \in A$.
- The value of flow f is $\text{val}(x) = x(\delta^+(\{s\})) - x(\delta^-(\{s\}))$.

Max Flow / Min Cut

- Review: Vector $x \in \mathbb{R}^A$ is a feasible flow if it satisfies
  1. Conservation: $x(\delta^+(\{i\}) = x(\delta^-(\{i\})$ for all $i \in E$, i.e., flow out $=$ flow in.
  2. Boundedness: $0 \le x_{ij} \le u_{ij}$ for all $i \to j \in A$.
- The value of flow f is $\mathrm{val}(x) = x(\delta^+(\{s\})) - x(\delta^-(\{s\}))$.

Theorem (Ford & Fulkerson)

For any capacities u, $\mathrm{val}^* \equiv \max_x \mathrm{val}(x) = \min_S \mathrm{cap}(S) \equiv \mathrm{cap}^*$, *i.e., the value of a max flow equals the capacity of a min cut.*

Max Flow / Min Cut

- Review: Vector $x \in \mathbb{R}^A$ is a feasible flow if it satisfies
  1. Conservation: $x(\delta^+(\{i\}) = x(\delta^-(\{i\})$ for all $i \in E$, i.e., flow out $=$ flow in.
  2. Boundedness: $0 \le x_{ij} \le u_{ij}$ for all $i \to j \in A$.
- The value of flow f is $\mathrm{val}(x) = x(\delta^+(\{s\})) - x(\delta^-(\{s\}))$.

Theorem (Ford & Fulkerson)

For any capacities u, $\mathrm{val}^* \equiv \max_x \mathrm{val}(x) = \min_S \mathrm{cap}(S) \equiv \mathrm{cap}^*$, *i.e., the value of a max flow equals the capacity of a min cut.*

- Now we want to sketch part of the proof of this, since some later proofs will use the same technique.

Algorithmic proof of Max Flow / Min Cut

- First, weak duality. For any feasible flow x and cut S:

$$\begin{aligned}
\mathrm{val}(x) &= x(\delta^+(\{s\})) - x(\delta^-(\{s\})) \\
&\qquad + \textstyle\sum_{i\in S}[x(\delta^+(\{i\})) - x(\delta^-(\{i\}))] \\
&= x(\delta^+(S+s)) - x(\delta^-(S+s)) \\
&\le u(\delta^+(S+s)) - 0 = \mathrm{cap}(S).
\end{aligned}$$

Algorithmic proof of Max Flow / Min Cut

- First, weak duality. For any feasible flow x and cut S:

$$\begin{aligned}
\mathrm{val}(x) &= x(\delta^+(\{s\})) - x(\delta^-(\{s\})) \\
&\qquad + \textstyle\sum_{i\in S}[x(\delta^+(\{i\})) - x(\delta^-(\{i\}))] \\
&= x(\delta^+(S+s)) - x(\delta^-(S+s)) \\
&\le u(\delta^+(S+s)) - 0 = \mathrm{cap}(S).
\end{aligned}$$

- An augmenting path w.r.t. feasible flow x is a directed path P such that $i \to j \in P$ implies either (i) $i \to j \in A$ and $x_{ij} < u_{ij}$, or (ii) $j \to i \in A$ and $x_{ji} > 0$.

Algorithmic proof of Max Flow / Min Cut

- First, weak duality. For any feasible flow x and cut S:

$$\begin{aligned}
\mathrm{val}(x) &= x(\delta^+(\{s\})) - x(\delta^-(\{s\})) \\
&\qquad + \textstyle\sum_{i \in S}[x(\delta^+(\{i\})) - x(\delta^-(\{i\}))] \\
&= x(\delta^+(S+s)) - x(\delta^-(S+s)) \\
&\leq u(\delta^+(S+s)) - 0 = \mathrm{cap}(S).
\end{aligned}$$

- An augmenting path w.r.t. feasible flow x is a directed path P such that $i \to j \in P$ implies either (i) $i \to j \in A$ and $x_{ij} < u_{ij}$, or (ii) $j \to i \in A$ and $x_{ji} > 0$.
- If there is an augmenting path P from s to t w.r.t. x, then clearly we can push some flow $\alpha > 0$ through P and increase $\mathrm{val}(x)$ by α, proving that x is not maximum.

Algorithmic proof of Max Flow / Min Cut

- First, weak duality. For any feasible flow x and cut S:

$$\begin{aligned}
\mathrm{val}(x) &= x(\delta^+(\{s\})) - x(\delta^-(\{s\})) \\
&\quad\quad + \textstyle\sum_{i\in S}[x(\delta^+(\{i\})) - x(\delta^-(\{i\}))] \\
&= x(\delta^+(S+s)) - x(\delta^-(S+s)) \\
&\leq u(\delta^+(S+s)) - 0 = \mathrm{cap}(S).
\end{aligned}$$

- An augmenting path w.r.t. feasible flow x is a directed path P such that $i \to j \in P$ implies either (i) $i \to j \in A$ and $x_{ij} < u_{ij}$, or (ii) $j \to i \in A$ and $x_{ji} > 0$.
- If there is an augmenting path P from s to t w.r.t. x, then clearly we can push some flow $\alpha > 0$ through P and increase $\mathrm{val}(x)$ by α, proving that x is not maximum.
- Conversely, suppose $\nexists$ aug. path P from s to t w.r.t. x. Define $S = \{i \in E \mid \exists$ aug. path from s to i w.r.t. $x\}$.

Algorithmic proof of Max Flow / Min Cut

- First, weak duality. For any feasible flow x and cut S:

$$\begin{aligned}\mathrm{val}(x) &= x(\delta^+(\{s\})) - x(\delta^-(\{s\})) \\ &\qquad + \textstyle\sum_{i\in S}[x(\delta^+(\{i\})) - x(\delta^-(\{i\}))] \\ &= x(\delta^+(S+s)) - x(\delta^-(S+s)) \\ &\leq u(\delta^+(S+s)) - 0 = \mathrm{cap}(S).\end{aligned}$$

- An augmenting path w.r.t. feasible flow x is a directed path P such that $i \to j \in P$ implies either (i) $i \to j \in A$ and $x_{ij} < u_{ij}$, or (ii) $j \to i \in A$ and $x_{ji} > 0$.
- If there is an augmenting path P from s to t w.r.t. x, then clearly we can push some flow $\alpha > 0$ through P and increase $\mathrm{val}(x)$ by α, proving that x is not maximum.
- Conversely, suppose $\nexists$ aug. path P from s to t w.r.t. x. Define $S = \{i \in E \mid \exists$ aug. path from s to i w.r.t. $x\}$.
- For $i \in S + s$ and $j \notin S + s$ we must have $x_{ij} = u_{ij}$ and $x_{ji} = 0$, and so $\mathrm{val}(x) = x(\delta^+(S+s)) - x(\delta^-(S+s)) = u(\delta^+(S+s)) - 0 = \mathrm{cap}(S)$.

More Max Flow / Min Cut observations

- This proof suggests an algorithm: find and push flow on augmenting paths until none exist, and then we're optimal.

More Max Flow / Min Cut observations

- This proof suggests an algorithm: find and push flow on augmenting paths until none exist, and then we're optimal.
  - The trick is to *bound* the number of iterations (augmenting paths).

More Max Flow / Min Cut observations

- This proof suggests an algorithm: find and push flow on augmenting paths until none exist, and then we're optimal.
  - The trick is to *bound* the number of iterations (augmenting paths).
- The generic proof idea we'll use later: push flow until you can't push any more, and then the cut that blocks further pushes must be a min cut.

More Max Flow / Min Cut observations

- This proof suggests an algorithm: find and push flow on augmenting paths until none exist, and then we're optimal.
  - The trick is to *bound* the number of iterations (augmenting paths).
- The generic proof idea we'll use later: push flow until you can't push any more, and then the cut that blocks further pushes must be a min cut.
- There are Max Flow algorithms *not* based on augmenting paths, such as Push-Relabel.

More Max Flow / Min Cut observations

- This proof suggests an algorithm: find and push flow on augmenting paths until none exist, and then we're optimal.
  - The trick is to *bound* the number of iterations (augmenting paths).
- The generic proof idea we'll use later: push flow until you can't push any more, and then the cut that blocks further pushes must be a min cut.
- There are Max Flow algorithms *not* based on augmenting paths, such as Push-Relabel.
  - Push-Relabel allows some violations of conservation, and pushes flow on individual arcs instead of paths, using distance labels (that estimate how far node i is from t via an augmenting path) as a guide.

More Max Flow / Min Cut observations

- This proof suggests an algorithm: find and push flow on augmenting paths until none exist, and then we're optimal.
  - The trick is to *bound* the number of iterations (augmenting paths).
- The generic proof idea we'll use later: push flow until you can't push any more, and then the cut that blocks further pushes must be a min cut.
- There are Max Flow algorithms *not* based on augmenting paths, such as Push-Relabel.
  - Push-Relabel allows some violations of conservation, and pushes flow on individual arcs instead of paths, using distance labels (that estimate how far node i is from t via an augmenting path) as a guide.
  - Many SFMin algorithms are based on Push-Relabel.

More Max Flow / Min Cut observations

- This proof suggests an algorithm: find and push flow on augmenting paths until none exist, and then we're optimal.
  - The trick is to *bound* the number of iterations (augmenting paths).
- The generic proof idea we'll use later: push flow until you can't push any more, and then the cut that blocks further pushes must be a min cut.
- There are Max Flow algorithms *not* based on augmenting paths, such as Push-Relabel.
  - Push-Relabel allows some violations of conservation, and pushes flow on individual arcs instead of paths, using distance labels (that estimate how far node i is from t via an augmenting path) as a guide.
  - Many SFMin algorithms are based on Push-Relabel.
- Min Cut is a canonical example of minimizing a submodular function, and many of the algorithms are based on analogies with Max Flow / Min Cut.

Further examples which are all submodular (Krause)

- Matroids: The rank function of a matroid.

Further examples which are all submodular (Krause)

- Matroids: The rank function of a matroid.
- Coverage: There is a set F a facilities we can open, and a set C of clients we want to service. There is a bipartite graph $B = (F \cup C, A)$ from F to C such that if we open $S \subseteq F$, we serve the set of clients $\Gamma(S) \equiv \{j \in C \mid i \to j \in A$, some $i \in S\}$. If $w \geq 0$ then $w(\Gamma(S))$ is submodular.

Further examples which are all submodular (Krause)

- Matroids: The rank function of a matroid.
- Coverage: There is a set F a facilities we can open, and a set C of clients we want to service. There is a bipartite graph $B = (F \cup C, A)$ from F to C such that if we open $S \subseteq F$, we serve the set of clients $\Gamma(S) \equiv \{j \in C \mid i \rightarrow j \in A$, some $i \in S\}$. If $w \geq 0$ then $w(\Gamma(S))$ is submodular.
- Queues: If a system E of queues satisfies a "conservation law" then the amount of work that can be done by queues in $S \subseteq E$ is submodular.

Further examples which are all submodular (Krause)

- Matroids: The rank function of a matroid.
- Coverage: There is a set F a facilities we can open, and a set C of clients we want to service. There is a bipartite graph $B = (F \cup C, A)$ from F to C such that if we open $S \subseteq F$, we serve the set of clients $\Gamma(S) \equiv \{j \in C \mid i \to j \in A$, some $i \in S\}$. If $w \geq 0$ then $w(\Gamma(S))$ is submodular.
- Queues: If a system E of queues satisfies a "conservation law" then the amount of work that can be done by queues in $S \subseteq E$ is submodular.
- Entropy: The *Shannon entropy* of a random vector.

Further examples which are all submodular (Krause)

- Matroids: The rank function of a matroid.
- Coverage: There is a set F a facilities we can open, and a set C of clients we want to service. There is a bipartite graph $B = (F \cup C, A)$ from F to C such that if we open $S \subseteq F$, we serve the set of clients $\Gamma(S) \equiv \{j \in C \mid i \to j \in A, \text{ some } i \in S\}$. If $w \geq 0$ then $w(\Gamma(S))$ is submodular.
- Queues: If a system E of queues satisfies a "conservation law" then the amount of work that can be done by queues in $S \subseteq E$ is submodular.
- Entropy: The *Shannon entropy* of a random vector.
- Sensor location: If we have a joint probability distribution over two random vectors $P(X, Y)$ indexed by E and the X variables are *conditionally independent* given Y, then the expected reduction in the uncertainty of about Y given the values of X on subset S is submodular. Think of placing sensors at a subset S of locations in the ground set E in order to measure Y; a sort of stochastic coverage.

Outline

Optimizing submodular functions

- In our motivating example we wanted to $\min_{S \subseteq E} c(S) - p(S)$.

Optimizing submodular functions

- In our motivating example we wanted to $\min_{S \subseteq E} c(S) - p(S)$.
- This is a specific example of the generic problem of Submodular Function Minimization (SFMin):

$$\text{Given submodular } f\text{, solve } \min_{S \subseteq E} f(S).$$

Optimizing submodular functions

- In our motivating example we wanted to $\min_{S \subseteq E} c(S) - p(S)$.
- This is a specific example of the generic problem of Submodular Function Minimization (SFMin):

$$\text{Given submodular } f\text{, solve } \min_{S \subseteq E} f(S).$$

- By contrast, in other contexts we want to *maximize*. For example, in an undirected graph with weights $w \geq 0$ on the edges, the Max Cut problem is to $\max_{S \subseteq E} w(\delta(S))$.

Optimizing submodular functions

- In our motivating example we wanted to $\min_{S \subseteq E} c(S) - p(S)$.
- This is a specific example of the generic problem of Submodular Function Minimization (SFMin):

$$\text{Given submodular } f\text{, solve } \min_{S \subseteq E} f(S).$$

- By contrast, in other contexts we want to *maximize*. For example, in an undirected graph with weights $w \geq 0$ on the edges, the Max Cut problem is to $\max_{S \subseteq E} w(\delta(S))$.
- Generically, Submodular Function Maximization (SFMax) is:

$$\text{Given submodular } f\text{, solve } \max_{S \subseteq E} f(S).$$

Constrained SFMax

- More generally, in the sensor location example, we want to find a subset that *maximizes* uncertainty reduction.

Constrained SFMax

- More generally, in the sensor location example, we want to find a subset that *maximizes* uncertainty reduction.
  - The function is monotone, i.e., $S \subseteq T \implies f(S) \leq f(T)$.

Constrained SFMax

- More generally, in the sensor location example, we want to find a subset that *maximizes* uncertainty reduction.
  - The function is monotone, i.e., $S \subseteq T \implies f(S) \leq f(T)$.
  - So we should just choose $S = E$ to maximize???

Constrained SFMax

- More generally, in the sensor location example, we want to find a subset that *maximizes* uncertainty reduction.
  - The function is monotone, i.e., $S \subseteq T \implies f(S) \leq f(T)$.
  - So we should just choose $S = E$ to maximize???
  - But in such problems we typically have a budget B, and want to maximize subject to the budget.

Constrained SFMax

- More generally, in the sensor location example, we want to find a subset that *maximizes* uncertainty reduction.
  - The function is monotone, i.e., $S \subseteq T \implies f(S) \leq f(T)$.
  - So we should just choose $S = E$ to maximize???
  - But in such problems we typically have a budget B, and want to maximize subject to the budget.
- This leads to considering Constrained SFMax:

Given submodular f and budget B, solve $\max_{S \subseteq E : |S| \leq B} f(S)$.

Constrained SFMax

- More generally, in the sensor location example, we want to find a subset that *maximizes* uncertainty reduction.
  - The function is monotone, i.e., $S \subseteq T \implies f(S) \leq f(T)$.
  - So we should just choose $S = E$ to maximize???
  - But in such problems we typically have a budget B, and want to maximize subject to the budget.
- This leads to considering Constrained SFMax:

  Given submodular f and budget B, solve $\max_{S \subseteq E : |S| \leq B} f(S)$.

- There are also variants of this with more general budgets.

Constrained SFMax

- More generally, in the sensor location example, we want to find a subset that *maximizes* uncertainty reduction.
  - The function is monotone, i.e., $S \subseteq T \implies f(S) \leq f(T)$.
  - So we should just choose $S = E$ to maximize???
  - But in such problems we typically have a budget B, and want to maximize subject to the budget.
- This leads to considering Constrained SFMax:

  Given submodular f and budget B, solve $\max_{S \subseteq E : |S| \leq B} f(S)$.

- There are also variants of this with more general budgets.
  - E.g., if a sensor in location i costs $c_i \geq 0$, then our constraint would be $c(S) \leq B$ (a *knapsack* constraint).

Constrained SFMax

- More generally, in the sensor location example, we want to find a subset that *maximizes* uncertainty reduction.
  - The function is monotone, i.e., $S \subseteq T \implies f(S) \leq f(T)$.
  - So we should just choose $S = E$ to maximize???
  - But in such problems we typically have a budget B, and want to maximize subject to the budget.
- This leads to considering Constrained SFMax:

Given submodular f and budget B, solve $\displaystyle \max_{S \subseteq E : |S| \leq B} f(S)$.

- There are also variants of this with more general budgets.
  - E.g., if a sensor in location i costs $c_i \geq 0$, then our constraint would be $c(S) \leq B$ (a *knapsack* constraint).
  - Or we could have multiple budgets, or ...

Complexity of submodular optimization

- The canonical example of SFMin is Min Cut, which has many polynomial algorithms, so there is some hope that SFMin is also polynomial.

Complexity of submodular optimization

- The canonical example of SFMin is Min Cut, which has many polynomial algorithms, so there is some hope that SFMin is also polynomial.
- The canonical example of SFMax is Max Cut, which is know to be NP Hard, and so SFMax is NP Hard.

Complexity of submodular optimization

- The canonical example of SFMin is Min Cut, which has many polynomial algorithms, so there is some hope that SFMin is also polynomial.
- The canonical example of SFMax is Max Cut, which is know to be NP Hard, and so SFMax is NP Hard.
  - Constrained SFMax is also NP Hard.

Complexity of submodular optimization

- The canonical example of SFMin is Min Cut, which has many polynomial algorithms, so there is some hope that SFMin is also polynomial.
- The canonical example of SFMax is Max Cut, which is know to be NP Hard, and so SFMax is NP Hard.
  - Constrained SFMax is also NP Hard.
  - Thus for the SFMax problems, we will be interested in approximation algorithms.

Complexity of submodular optimization

- The canonical example of SFMin is Min Cut, which has many polynomial algorithms, so there is some hope that SFMin is also polynomial.
- The canonical example of SFMax is Max Cut, which is know to be NP Hard, and so SFMax is NP Hard.
  - Constrained SFMax is also NP Hard.
  - Thus for the SFMax problems, we will be interested in approximation algorithms.
  - An algorithm for an maximization problem is a α-approximation if it always produces a feasible solution with objective value at least $\alpha \cdot \mathrm{OPT}$.

Complexity of submodular optimization

- Recall that our algorithms interact with f via calls to the value oracle $\mathcal{E}$, and one call costs $\mathrm{EO} = \Omega(n)$.

Complexity of submodular optimization

- Recall that our algorithms interact with f via calls to the value oracle $\mathcal{E}$, and one call costs $\mathrm{EO} = \Omega(n)$.
- As is usual in computational complexity, we have to think about how the running time varies as a function of the size of the problem.

Complexity of submodular optimization

- Recall that our algorithms interact with f via calls to the value oracle $\mathcal{E}$, and one call costs $\mathrm{EO} = \Omega(n)$.
- As is usual in computational complexity, we have to think about how the running time varies as a function of the size of the problem.
  - One clear measure of size is $n = |E|$.

Complexity of submodular optimization

- Recall that our algorithms interact with f via calls to the value oracle $\mathcal{E}$, and one call costs $\mathrm{EO} = \Omega(n)$.
- As is usual in computational complexity, we have to think about how the running time varies as a function of the size of the problem.
  - One clear measure of size is $n = |E|$.
  - But we might also need to think about the sizes of the values $f(S)$.

Complexity of submodular optimization

- Recall that our algorithms interact with f via calls to the value oracle $\mathcal{E}$, and one call costs $\mathrm{EO} = \Omega(n)$.
- As is usual in computational complexity, we have to think about how the running time varies as a function of the size of the problem.
  - One clear measure of size is $n = |E|$.
  - But we might also need to think about the sizes of the values $f(S)$.
  - When f is integer-valued, define $M = \max_{S \subseteq E} |f(S)|$.

Complexity of submodular optimization

- Recall that our algorithms interact with f via calls to the value oracle $\mathcal{E}$, and one call costs $\mathrm{EO} = \Omega(n)$.
- As is usual in computational complexity, we have to think about how the running time varies as a function of the size of the problem.
  - One clear measure of size is $n = |E|$.
  - But we might also need to think about the sizes of the values $f(S)$.
  - When f is integer-valued, define $M = \max_{S \subseteq E} |f(S)|$.
  - Unfortunately, exactly computing M is NP Hard (SFMax), but we can compute a good enough bound on M in $O(n\mathrm{EO})$ time.

Types of polynomial algorithms for SFMin/Max

- Assume for the moment that all data are integers.

Types of polynomial algorithms for SFMin/Max

- Assume for the moment that all data are integers.
  - An algorithm is pseudo-polynomial if it is polynomial in n, M, and EO.

Types of polynomial algorithms for SFMin/Max

- Assume for the moment that all data are integers.
  - An algorithm is pseudo-polynomial if it is polynomial in n, M, and EO.
    - Allowing M is not polynomial, as the real size of M is $O(\log M)$, and M is exponential in $\log M$.

Types of polynomial algorithms for SFMin/Max

- Assume for the moment that all data are integers.
  - An algorithm is pseudo-polynomial if it is polynomial in n, M, and EO.
    - Allowing M is not polynomial, as the real size of M is $O(\log M)$, and M is exponential in $\log M$.
  - An algorithm is (weakly) polynomial if it is polynomial in n, $\log M$, and EO.

Types of polynomial algorithms for SFMin/Max

- Assume for the moment that all data are integers.
  - An algorithm is pseudo-polynomial if it is polynomial in n, M, and EO.
    - Allowing M is not polynomial, as the real size of M is $O(\log M)$, and M is exponential in $\log M$.
  - An algorithm is (weakly) polynomial if it is polynomial in n, $\log M$, and EO.
- If non-integral data is allowed, then the running time cannot depend on M at all.

Types of polynomial algorithms for SFMin/Max

- Assume for the moment that all data are integers.
  - An algorithm is pseudo-polynomial if it is polynomial in n, M, and EO.
    - Allowing M is not polynomial, as the real size of M is $O(\log M)$, and M is exponential in $\log M$.
  - An algorithm is (weakly) polynomial if it is polynomial in n, $\log M$, and EO.
- If non-integral data is allowed, then the running time cannot depend on M at all.
  - An algorithm is strongly polynomial if it is polynomial in n and EO.

Types of polynomial algorithms for SFMin/Max

- Assume for the moment that all data are integers.
  - An algorithm is pseudo-polynomial if it is polynomial in n, M, and EO.
    - Allowing M is not polynomial, as the real size of M is $O(\log M)$, and M is exponential in $\log M$.
  - An algorithm is (weakly) polynomial if it is polynomial in n, $\log M$, and EO.
- If non-integral data is allowed, then the running time cannot depend on M at all.
  - An algorithm is strongly polynomial if it is polynomial in n and EO.
  - There is no apparent reason why an SFMin/Max algorithm needs multiplication or division, so we call an algorithm fully combinatorial if it is strongly polynomial, and uses only addition/subtraction and comparisons.

Is submodularity concavity or convexity?

- Submodular functions are sort of *concave*: Suppose that set function f has $f(S) = g(|S|)$ for some $g : \mathbb{R} \to \mathbb{R}$. Then f is submodular iff g is concave (homework). This is the "decreasing returns to scale" point of view.

Is submodularity concavity or convexity?

- Submodular functions are sort of *concave*: Suppose that set function f has $f(S) = g(|S|)$ for some $g : \mathbb{R} \to \mathbb{R}$. Then f is submodular iff g is concave (homework). This is the "decreasing returns to scale" point of view.
- Submodular functions are sort of *convex*: Set function f induces values on $\{0,1\}^E$ via $\hat{f}(\chi(S)) = f(S)$, where $\chi(S)_e = 1$ if $e \in S$, 0 otherwise. There is a canonical piecewise linear way to extend $\hat{f}$ to $[0,1]^E$ called the Lovász extension. Then f is submodular iff $\hat{f}$ is convex.

Is submodularity concavity or convexity?

- Submodular functions are sort of *concave*: Suppose that set function f has $f(S) = g(|S|)$ for some $g : \mathbb{R} \to \mathbb{R}$. Then f is submodular iff g is concave (homework). This is the "decreasing returns to scale" point of view.
- Submodular functions are sort of *convex*: Set function f induces values on $\{0,1\}^E$ via $\hat{f}(\chi(S)) = f(S)$, where $\chi(S)_e = 1$ if $e \in S$, 0 otherwise. There is a canonical piecewise linear way to extend $\hat{f}$ to $[0,1]^E$ called the Lovász extension. Then f is submodular iff $\hat{f}$ is convex.
- Continuous convex functions are easy to minimize, hard to maximize; SFMin looks easy, SFMax is hard. Thus the convex view looks better.

Is submodularity concavity or convexity?

- Submodular functions are sort of *concave*: Suppose that set function f has $f(S) = g(|S|)$ for some $g : \mathbb{R} \to \mathbb{R}$. Then f is submodular iff g is concave (homework). This is the "decreasing returns to scale" point of view.
- Submodular functions are sort of *convex*: Set function f induces values on $\{0,1\}^E$ via $\hat{f}(\chi(S)) = f(S)$, where $\chi(S)_e = 1$ if $e \in S$, 0 otherwise. There is a canonical piecewise linear way to extend $\hat{f}$ to $[0,1]^E$ called the Lovász extension. Then f is submodular iff $\hat{f}$ is convex.
- Continuous convex functions are easy to minimize, hard to maximize; SFMin looks easy, SFMax is hard. Thus the convex view looks better.
- There is a whole theory of discrete convexity starting from the Lovász extension that parallels continuous convex analysis, see Murota's book.

Outline

Submodular polyhedra

- Let's associate submodular functions with polyhedra.

Submodular polyhedra

- Let's associate submodular functions with polyhedra.
- It turns out that the right thing to do is to think about vectors $x \in \mathbb{R}^E$, and so polyhedra in $\mathbb{R}^E$.

Submodular polyhedra

- Let's associate submodular functions with polyhedra.
- It turns out that the right thing to do is to think about vectors $x \in \mathbb{R}^E$, and so polyhedra in $\mathbb{R}^E$.
- The key constraint for us is for some subset $S \subseteq E$

$$x(S) \leq f(S).$$

Submodular polyhedra

- Let's associate submodular functions with polyhedra.
- It turns out that the right thing to do is to think about vectors $x \in \mathbb{R}^E$, and so polyhedra in $\mathbb{R}^E$.
- The key constraint for us is for some subset $S \subseteq E$

$$x(S) \leq f(S).$$

- We can think of this as a sort of generalized upper bound on sums over subsets of components of x.

Submodular polyhedra

- Let's associate submodular functions with polyhedra.
- It turns out that the right thing to do is to think about vectors $x \in \mathbb{R}^E$, and so polyhedra in $\mathbb{R}^E$.
- The key constraint for us is for some subset $S \subseteq E$

$$x(S) \leq f(S).$$

- We can think of this as a sort of generalized upper bound on sums over subsets of components of x.
- What about when $S = \emptyset$? We get $x(\emptyset) \equiv 0 \leq f(\emptyset)$???

Submodular polyhedra

- Let's associate submodular functions with polyhedra.
- It turns out that the right thing to do is to think about vectors $x \in \mathbb{R}^E$, and so polyhedra in $\mathbb{R}^E$.
- The key constraint for us is for some subset $S \subseteq E$

$$x(S) \leq f(S).$$

- We can think of this as a sort of generalized upper bound on sums over subsets of components of x.
- What about when $S = \emptyset$? We get $x(\emptyset) \equiv 0 \leq f(\emptyset)$???
  - To get this to make sense we will *normalize* all our submodular functions via $f(S) \leftarrow f(S) - f(\emptyset)$ in order to be able to assume that $f(\emptyset) = 0$.

Submodular polyhedra

- Let's associate submodular functions with polyhedra.
- It turns out that the right thing to do is to think about vectors $x \in \mathbb{R}^E$, and so polyhedra in $\mathbb{R}^E$.
- The key constraint for us is for some subset $S \subseteq E$

$$x(S) \leq f(S).$$

- We can think of this as a sort of generalized upper bound on sums over subsets of components of x.
- What about when $S = \emptyset$? We get $x(\emptyset) \equiv 0 \leq f(\emptyset)$???
  - To get this to make sense we will *normalize* all our submodular functions via $f(S) \leftarrow f(S) - f(\emptyset)$ in order to be able to assume that $f(\emptyset) = 0$.
  - Notice that this normalization does not change the optimal subset for SFMin and SFMax.

Submodular polyhedra

- Let's associate submodular functions with polyhedra.
- It turns out that the right thing to do is to think about vectors $x \in \mathbb{R}^E$, and so polyhedra in $\mathbb{R}^E$.
- The key constraint for us is for some subset $S \subseteq E$

$$x(S) \leq f(S).$$

- We can think of this as a sort of generalized upper bound on sums over subsets of components of x.
- What about when $S = \emptyset$? We get $x(\emptyset) \equiv 0 \leq f(\emptyset)$???
  - To get this to make sense we will *normalize* all our submodular functions via $f(S) \leftarrow f(S) - f(\emptyset)$ in order to be able to assume that $f(\emptyset) = 0$.
  - Notice that this normalization does not change the optimal subset for SFMin and SFMax.
  - It further implies that the optimal value for SFMin is non-positive, and the optimal value for SFMax is non-negative, since we can always get 0 by choosing $S = \emptyset$.

Submodular polyhedra

- Let's associate submodular functions with polyhedra.
- It turns out that the right thing to do is to think about vectors $x \in \mathbb{R}^E$, and so polyhedra in $\mathbb{R}^E$.
- The key constraint for us is for some subset $S \subseteq E$

$$x(S) \leq f(S).$$

- We can think of this as a sort of generalized upper bound on sums over subsets of components of x.
- What about when $S = \emptyset$? We get $x(\emptyset) \equiv 0 \leq f(\emptyset)$???
  - To get this to make sense we will *normalize* all our submodular functions via $f(S) \leftarrow f(S) - f(\emptyset)$ in order to be able to assume that $f(\emptyset) = 0$.
  - Notice that this normalization does not change the optimal subset for SFMin and SFMax.
  - It further implies that the optimal value for SFMin is non-positive, and the optimal value for SFMax is non-negative, since we can always get 0 by choosing $S = \emptyset$.
  - This normalization is non-trivial for Min Cut.

The submodular polyhedron

- Now that we've normalized s.t. $f(\emptyset) = 0$, define the submodular polyhedron associated with set function f by

$$P(f) \equiv \{x \in \mathbb{R}^E \mid x(S) \leq f(S) \; \forall S \subseteq E\}.$$

The submodular polyhedron

- Now that we've normalized s.t. $f(\emptyset) = 0$, define the submodular polyhedron associated with set function f by

$$P(f) \equiv \{x \in \mathbb{R}^E \mid x(S) \leq f(S) \; \forall S \subseteq E\}.$$

  - When f is submodular and monotone (a polymatroid rank function), $P(f)$ is just the polymatroid.

The submodular polyhedron

- Now that we've normalized s.t. $f(\emptyset) = 0$, define the submodular polyhedron associated with set function f by

$$P(f) \equiv \{x \in \mathbb{R}^E \mid x(S) \leq f(S) \ \forall S \subseteq E\}.$$

  - When f is submodular and monotone (a polymatroid rank function), $P(f)$ is just the polymatroid.

- It turns out to be convenient to also consider the face of $P(f)$ induced by the constraint $x(E) \leq f(E)$, called the base polyhedron of f:

$$B(f) \equiv \{x \in \mathbb{R}^E \mid x(S) \leq f(S) \forall S \subset E, \ x(E) = f(E)\}.$$

The submodular polyhedron

- Now that we've normalized s.t. $f(\emptyset) = 0$, define the submodular polyhedron associated with set function f by

$$P(f) \equiv \{x \in \mathbb{R}^E \mid x(S) \leq f(S) \ \forall S \subseteq E\}.$$

  - When f is submodular and monotone (a polymatroid rank function), $P(f)$ is just the polymatroid.

- It turns out to be convenient to also consider the face of $P(f)$ induced by the constraint $x(E) \leq f(E)$, called the base polyhedron of f:

$$B(f) \equiv \{x \in \mathbb{R}^E \mid x(S) \leq f(S) \forall S \subset E, \ x(E) = f(E)\}.$$

- We will soon show that $B(f)$ is always non-empty when f is submodular.

Optimizing over $B(f)$

- Now that we have a polyhedron it is natural to want to optimize over it.

Optimizing over $B(f)$

- Now that we have a polyhedron it is natural to want to optimize over it.
- Consider $\max w^T x$ s.t. $x \in P(f)$. Notice that $y \leq x$ and $x \in P(f)$ imply that $y \in P(f)$. Thus if some $w_e < 0$ the optimum is unbounded below. So let's assume that $w \geq 0$.

Optimizing over $B(f)$

- Now that we have a polyhedron it is natural to want to optimize over it.
- Consider $\max w^T x$ s.t. $x \in P(f)$. Notice that $y \le x$ and $x \in P(f)$ imply that $y \in P(f)$. Thus if some $w_e < 0$ the optimum is unbounded below. So let's assume that $w \ge 0$.
- Intuitively, with $w \ge 0$ a maximum solution will be forced up against the $x(E) \le f(E)$ constraint, and so it will become tight, and so an optimal solution will be in $B(f)$. So we consider $\max_{x \in \mathbb{R}^E} w^T x$ s.t. $x \in B(f)$.

Optimizing over $B(f)$

- Now that we have a polyhedron it is natural to want to optimize over it.
- Consider $\max w^T x$ s.t. $x \in P(f)$. Notice that $y \leq x$ and $x \in P(f)$ imply that $y \in P(f)$. Thus if some $w_e < 0$ the optimum is unbounded below. So let's assume that $w \geq 0$.
- Intuitively, with $w \geq 0$ a maximum solution will be forced up against the $x(E) \leq f(E)$ constraint, and so it will become tight, and so an optimal solution will be in $B(f)$. So we consider $\max_{x \in \mathbb{R}^E} w^T x$ s.t. $x \in B(f)$.
- The naive thing to do is to try to solve this *greedily*: Order the elements such that $w_1 \geq w_2 \geq \cdots \geq w_n$.

Outline

The Greedy Algorithm (Edmonds)

- Order the elements such that $w_1 \geq w_2 \geq \cdots \geq w_n$.

The Greedy Algorithm (Edmonds)

- Order the elements such that $w_1 \geq w_2 \geq \cdots \geq w_n$.
  1. Make x_1 as large as possible: $x_1 \leftarrow f(\{e_1\}) - f(\emptyset)$.

The Greedy Algorithm (Edmonds)

- Order the elements such that $w_1 \geq w_2 \geq \cdots \geq w_n$.
  1. Make x_1 as large as possible: $x_1 \leftarrow f(\{e_1\}) - f(\emptyset)$.
  2. Make x_2 as large as possible: $x_2 \leftarrow f(\{e_1, e_2\}) - f(\{e_1\})$.

The Greedy Algorithm (Edmonds)

- Order the elements such that $w_1 \geq w_2 \geq \cdots \geq w_n$.
  1. Make x_1 as large as possible: $x_1 \leftarrow f(\{e_1\}) - f(\emptyset)$.
  2. Make x_2 as large as possible: $x_2 \leftarrow f(\{e_1, e_2\}) - f(\{e_1\})$.
  3. Make x_3 as large as possible: $x_3 \leftarrow f(\{e_1, e_2, e_3\}) - f(\{e_1, e_2\})$

The Greedy Algorithm (Edmonds)

- Order the elements such that $w_1 \geq w_2 \geq \cdots \geq w_n$.
  1. Make x_1 as large as possible: $x_1 \leftarrow f(\{e_1\}) - f(\emptyset)$.
  2. Make x_2 as large as possible: $x_2 \leftarrow f(\{e_1, e_2\}) - f(\{e_1\})$.
  3. Make x_3 as large as possible: $x_3 \leftarrow f(\{e_1, e_2, e_3\}) - f(\{e_1, e_2\})$
  4. Etc, etc ...

The Greedy Algorithm (Edmonds)

- Order the elements such that $w_1 \geq w_2 \geq \cdots \geq w_n$.
  1. Make x_1 as large as possible: $x_1 \leftarrow f(\{e_1\}) - f(\emptyset)$.
  2. Make x_2 as large as possible: $x_2 \leftarrow f(\{e_1, e_2\}) - f(\{e_1\})$.
  3. Make x_3 as large as possible: $x_3 \leftarrow f(\{e_1, e_2, e_3\}) - f(\{e_1, e_2\})$
  4. Etc, etc ...
- Notice that this Greedy Algorithm depends only on the input linear order. We derived the order from w, but we could apply the same algorithm to any order $\prec$.

The Greedy Algorithm (Edmonds)

- Order the elements such that $w_1 \geq w_2 \geq \cdots \geq w_n$.
  1. Make x_1 as large as possible: $x_1 \leftarrow f(\{e_1\}) - f(\emptyset)$.
  2. Make x_2 as large as possible: $x_2 \leftarrow f(\{e_1, e_2\}) - f(\{e_1\})$.
  3. Make x_3 as large as possible: $x_3 \leftarrow f(\{e_1, e_2, e_3\}) - f(\{e_1, e_2\})$
  4. Etc, etc ...
- Notice that this Greedy Algorithm depends only on the input linear order. We derived the order from w, but we could apply the same algorithm to any order $\prec$.
- Given linear order $\prec$ and $e \in E$, define $e^{\prec} = \{g \in E \mid g \prec e\}$.

The Greedy Algorithm (Edmonds)

- Order the elements such that $w_1 \geq w_2 \geq \cdots \geq w_n$.
  1. Make x_1 as large as possible: $x_1 \leftarrow f(\{e_1\}) - f(\emptyset)$.
  2. Make x_2 as large as possible: $x_2 \leftarrow f(\{e_1, e_2\}) - f(\{e_1\})$.
  3. Make x_3 as large as possible: $x_3 \leftarrow f(\{e_1, e_2, e_3\}) - f(\{e_1, e_2\})$
  4. Etc, etc ...
- Notice that this Greedy Algorithm depends only on the input linear order. We derived the order from w, but we could apply the same algorithm to any order $\prec$.
- Given linear order $\prec$ and $e \in E$, define $e^{\prec} = \{g \in E \mid g \prec e\}$.
  - E.g., suppose that
    $\prec_1$ is $3 \prec_1 1 \prec_1 4 \prec_1 5 \prec_1 2$ and
    $\prec_2$ is $1 \prec_2 2 \prec_2 3 \prec_2 4 \prec_2 5$.

The Greedy Algorithm (Edmonds)

- Order the elements such that $w_1 \geq w_2 \geq \cdots \geq w_n$.
  1. Make x_1 as large as possible: $x_1 \leftarrow f(\{e_1\}) - f(\emptyset)$.
  2. Make x_2 as large as possible: $x_2 \leftarrow f(\{e_1, e_2\}) - f(\{e_1\})$.
  3. Make x_3 as large as possible: $x_3 \leftarrow f(\{e_1, e_2, e_3\}) - f(\{e_1, e_2\})$
  4. Etc, etc ...
- Notice that this Greedy Algorithm depends only on the input linear order. We derived the order from w, but we could apply the same algorithm to any order $\prec$.
- Given linear order $\prec$ and $e \in E$, define $e^{\prec} = \{g \in E \mid g \prec e\}$.
  - E.g., suppose that
    $\prec_1$ is $3 \prec_1 1 \prec_1 4 \prec_1 5 \prec_1 2$ and
    $\prec_2$ is $1 \prec_2 2 \prec_2 3 \prec_2 4 \prec_2 5$.
  - Then $3^{\prec_1} = \emptyset$, $3^{\prec_2} = \{1, 2\}$,
    and $2^{\prec_1} = \{1, 3, 4, 5\}$, $2^{\prec_2} = \{1\}$.

The Greedy Algorithm (Edmonds)

- Order the elements such that $w_1 \geq w_2 \geq \cdots \geq w_n$.
  1. Make x_1 as large as possible: $x_1 \leftarrow f(\{e_1\}) - f(\emptyset)$.
  2. Make x_2 as large as possible: $x_2 \leftarrow f(\{e_1, e_2\}) - f(\{e_1\})$.
  3. Make x_3 as large as possible: $x_3 \leftarrow f(\{e_1, e_2, e_3\}) - f(\{e_1, e_2\})$
  4. Etc, etc ...
- Notice that this Greedy Algorithm depends only on the input linear order. We derived the order from w, but we could apply the same algorithm to any order $\prec$.
- Given linear order $\prec$ and $e \in E$, define $e^{\prec} = \{g \in E \mid g \prec e\}$.
  - E.g., suppose that
    $\prec_1$ is $3 \prec_1 1 \prec_1 4 \prec_1 5 \prec_1 2$ and
    $\prec_2$ is $1 \prec_2 2 \prec_2 3 \prec_2 4 \prec_2 5$.
  - Then $3^{\prec_1} = \emptyset$, $3^{\prec_2} = \{1, 2\}$,
    and $2^{\prec_1} = \{1, 3, 4, 5\}$, $2^{\prec_2} = \{1\}$.
- In this notation we can re-express the main step of Greedy on the ith element in $\prec$ as
  "Make $x_{e_i} \leftarrow f(e_i^{\prec} + e_i) - f(e_i^{\prec})$."

The Greedy Algorithm produces a feasible x

- We now prove that the x computed by Greedy belongs to $B(f)$ as follows:

The Greedy Algorithm produces a feasible x

- We now prove that the x computed by Greedy belongs to $B(f)$ as follows:
  - Index the elements such that $\prec$ is $e_1 \prec e_2 \prec \cdots \prec e_n$. First, $x(E) = \sum_{e_i \in E}[f(e_i^{\prec} + e_i) - f(e_i^{\prec})] = f(E) - f(\emptyset) = f(E)$.

The Greedy Algorithm produces a feasible x

- We now prove that the x computed by Greedy belongs to $B(f)$ as follows:
  - Index the elements such that $\prec$ is $e_1 \prec e_2 \prec \cdots \prec e_n$. First, $x(E) = \sum_{e_i \in E}[f(e_i^{\prec} + e_i) - f(e_i^{\prec})] = f(E) - f(\emptyset) = f(E)$.
  - Now for any $\emptyset \subset S \subset E$ we need to verify that $x(S) \leq f(S)$. Define k as the largest index such that $e_k \in S$, and use induction on k.

The Greedy Algorithm produces a feasible x

- We now prove that the x computed by Greedy belongs to $B(f)$ as follows:
  - Index the elements such that $\prec$ is $e_1 \prec e_2 \prec \cdots \prec e_n$. First, $x(E) = \sum_{e_i \in E}[f(e_i^{\prec} + e_i) - f(e_i^{\prec})] = f(E) - f(\emptyset) = f(E)$.
  - Now for any $\emptyset \subset S \subset E$ we need to verify that $x(S) \leq f(S)$. Define k as the largest index such that $e_k \in S$, and use induction on k.
  - If $k = 1$ then $S = \{e_1\}$ and $x_1 = f(e_1^{\prec} + e_1) - f(e_1^{\prec}) = f(\{e_1\}) - f(\emptyset) = f(S)$.

The Greedy Algorithm produces a feasible x

- We now prove that the x computed by Greedy belongs to $B(f)$ as follows:
  - Index the elements such that $\prec$ is $e_1 \prec e_2 \prec \cdots \prec e_n$. First, $x(E) = \sum_{e_i \in E}[f(e_i^{\prec} + e_i) - f(e_i^{\prec})] = f(E) - f(\emptyset) = f(E)$.
  - Now for any $\emptyset \subset S \subset E$ we need to verify that $x(S) \leq f(S)$. Define k as the largest index such that $e_k \in S$, and use induction on k.
  - If $k = 1$ then $S = \{e_1\}$ and $x_1 = f(e_1^{\prec} + e_1) - f(e_1^{\prec}) = f(\{e_1\}) - f(\emptyset) = f(S)$.
  - If $k > 1$, then $S \cup e_k^{\prec} = e_{k+1}^{\prec}$ and $S \cap e_k^{\prec} = S - e_k$. Then submodularity implies that $f(S) \geq f(S \cup e_k^{\prec}) + f(S \cap e_k^{\prec}) - f(e_k^{\prec}) = f(e_{k+1}^{\prec}) + f(S - e_k) - f(e_k^{\prec})$.

The Greedy Algorithm produces a feasible x

- We now prove that the x computed by Greedy belongs to $B(f)$ as follows:
  - Index the elements such that $\prec$ is $e_1 \prec e_2 \prec \cdots \prec e_n$. First, $x(E) = \sum_{e_i \in E}[f(e_i^{\prec} + e_i) - f(e_i^{\prec})] = f(E) - f(\emptyset) = f(E)$.
  - Now for any $\emptyset \subset S \subset E$ we need to verify that $x(S) \leq f(S)$. Define k as the largest index such that $e_k \in S$, and use induction on k.
  - If $k = 1$ then $S = \{e_1\}$ and $x_1 = f(e_1^{\prec} + e_1) - f(e_1^{\prec}) = f(\{e_1\}) - f(\emptyset) = f(S)$.
  - If $k > 1$, then $S \cup e_k^{\prec} = e_{k+1}^{\prec}$ and $S \cap e_k^{\prec} = S - e_k$. Then submodularity implies that $f(S) \geq f(S \cup e_k^{\prec}) + f(S \cap e_k^{\prec}) - f(e_k^{\prec}) = f(e_{k+1}^{\prec}) + f(S - e_k) - f(e_k^{\prec})$.
  - The largest e_i in $S - e_k$ is smaller than k, so induction applies to $S - e_k$ and we get $x(S) - x_{e_k} = x(S - e_k) \leq f(S - e_k)$, or $x(S) \leq f(S - e_k) + x_{e_k} = f(S - e_k) + (f(e_k^{\prec} + e_k) - f(e_k^{\prec}))$.

The Greedy Algorithm produces a feasible x

- We now prove that the x computed by Greedy belongs to $B(f)$ as follows:
  - Index the elements such that $\prec$ is $e_1 \prec e_2 \prec \cdots \prec e_n$. First, $x(E) = \sum_{e_i \in E}[f(e_i^{\prec} + e_i) - f(e_i^{\prec})] = f(E) - f(\emptyset) = f(E)$.
  - Now for any $\emptyset \subset S \subset E$ we need to verify that $x(S) \leq f(S)$. Define k as the largest index such that $e_k \in S$, and use induction on k.
  - If $k = 1$ then $S = \{e_1\}$ and $x_1 = f(e_1^{\prec} + e_1) - f(e_1^{\prec}) = f(\{e_1\}) - f(\emptyset) = f(S)$.
  - If $k > 1$, then $S \cup e_k^{\prec} = e_{k+1}^{\prec}$ and $S \cap e_k^{\prec} = S - e_k$. Then submodularity implies that $f(S) \geq f(S \cup e_k^{\prec}) + f(S \cap e_k^{\prec}) - f(e_k^{\prec}) = f(e_{k+1}^{\prec}) + f(S - e_k) - f(e_k^{\prec})$.
  - The largest e_i in $S - e_k$ is smaller than k, so induction applies to $S - e_k$ and we get $x(S) - x_{e_k} = x(S - e_k) \leq f(S - e_k)$, or $x(S) \leq f(S - e_k) + x_{e_k} = f(S - e_k) + (f(e_k^{\prec} + e_k) - f(e_k^{\prec}))$.
  - Thus $x(S) \leq f(S - e_k) + (f(e_k^{\prec} + e_k) - f(e_k^{\prec})) = f(e_{k+1}^{\prec}) + f(S - e_k) - f(e_k^{\prec}) \leq f(S)$.

Is Greedy's solution optimal?

- Recall that we are trying to solve $\max_{x \in \mathbb{R}^E} w^T x$ s.t. $x \in B(f)$.

Is Greedy's solution optimal?

- Recall that we are trying to solve $\max_{x \in \mathbb{R}^E} w^T x$ s.t. $x \in B(f)$.
- This is a linear program (LP):

$$
\begin{array}{rcll}
\max w^T x & & & \\
\text{s.t. } x(S) & \leq & f(S) & \text{for all } \emptyset \subset S \subset E \\
x(E) & = & f(E) & \\
x & & \text{free.} &
\end{array}
$$

Is Greedy's solution optimal?

- Recall that we are trying to solve $\max_{x \in \mathbb{R}^E} w^T x$ s.t. $x \in B(f)$.
- This is a linear program (LP):

$$
\begin{array}{rcll}
\max w^T x & & & \\
\text{s.t. } x(S) & \leq & f(S) & \text{for all } \emptyset \subset S \subset E \\
x(E) & = & f(E) & \\
x & & \text{free.} &
\end{array}
$$

- This LP has 2^n constraints, one for each S.

Is Greedy's solution optimal?

- Recall that we are trying to solve $\max_{x \in \mathbb{R}^E} w^T x$ s.t. $x \in B(f)$.
- This is a linear program (LP):

$$
\begin{array}{rcll}
\max w^T x & & & \\
\text{s.t. } x(S) & \leq & f(S) & \text{for all } \emptyset \subset S \subset E \\
x(E) & = & f(E) & \\
x & & \text{free.} &
\end{array}
$$

- This LP has 2^n constraints, one for each S.
- Optimality is proven via duality. Put dual variable π_S on constraint $x(S) \leq f(S)$ to get the dual:

$$
\begin{array}{rcll}
\min \sum_{S \subseteq E} f(S) \pi_S & & & \\
\text{s.t. } \sum_{S \ni e} \pi_S & = & w_e & \text{for all } e \in E \\
\pi_S & \geq & 0 & \text{for all } S \subset E \\
\pi_E & & \text{free.} &
\end{array}
$$

Is Greedy's solution optimal?

- Recall that we are trying to solve $\max_{x \in \mathbb{R}^E} w^T x$ s.t. $x \in B(f)$.
- This is a linear program (LP):

$$
\begin{array}{rcll}
\max w^T x & & & \\
\text{s.t. } x(S) & \leq & f(S) & \text{for all } \emptyset \subset S \subset E \\
x(E) & = & f(E) & \\
x & & \text{free.} &
\end{array}
$$

- This LP has 2^n constraints, one for each S.
- Optimality is proven via duality. Put dual variable π_S on constraint $x(S) \leq f(S)$ to get the dual:

$$
\begin{array}{rcll}
\min \sum_{S \subseteq E} f(S) \pi_S & & & \\
\text{s.t. } \sum_{S \ni e} \pi_S & = & w_e & \text{for all } e \in E \\
\pi_S & \geq & 0 & \text{for all } S \subset E \\
\pi_E & & \text{free.} &
\end{array}
$$

- In order to show optimality of the x coming from Greedy, we construct a dual optimal solution.

Dual feasibility

- Here are the dual LPs:

$$
\begin{array}{rcll}
\max w^T x & & & \\
\text{s.t. } x(S) & \leq & f(S) & \forall S \\
x(E) & = & f(E) & \\
x & & \text{free.} &
\end{array}
\qquad
\begin{array}{rcll}
\min \sum_{S \subseteq E} f(S) \pi_S & & & \\
\text{s.t. } \sum_{S \ni e} \pi_S & = & w_e & \\
\pi_S & \geq & 0 & S \neq E \\
\pi_E & & \text{free.} &
\end{array}
$$

Dual feasibility

- Here are the dual LPs:

$$\begin{array}{rcll} \max w^T x & & & \\ \text{s.t. } x(S) & \leq & f(S) & \forall S \\ x(E) & = & f(E) & \\ x & & \text{free.} & \end{array} \qquad \begin{array}{rcll} \min \sum_{S \subseteq E} f(S) \pi_S & & & \\ \text{s.t. } \sum_{S \ni e} \pi_S & = & w_e & \\ \pi_S & \geq & 0 & S \neq E \\ \pi_E & & \text{free.} & \end{array}$$

- Define π_S like this: Put $\pi_S = w_{e_{i-1}} - w_{e_i}$ if $S = e_i^{\prec}$, $\pi_E = w_{e_n} - 0$ (using "$w_{e_{n+1}} = 0$"), and $\pi_S = 0$ otherwise.

Dual feasibility

- Here are the dual LPs:

$$\begin{array}{rcll} \max w^T x & & & \\ \text{s.t. } x(S) & \leq & f(S) & \forall S \\ x(E) & = & f(E) & \\ x & & \text{free.} & \end{array} \qquad \begin{array}{rcll} \min \sum_{S \subseteq E} f(S) \pi_S & & & \\ \text{s.t. } \sum_{S \ni e} \pi_S & = & w_e & \\ \pi_S & \geq & 0 & S \neq E \\ \pi_E & & \text{free.} & \end{array}$$

- Define π_S like this: Put $\pi_S = w_{e_{i-1}} - w_{e_i}$ if $S = e_i^{\prec}$, $\pi_E = w_{e_n} - 0$ (using "$w_{e_{n+1}} = 0$"), and $\pi_S = 0$ otherwise.
- First, note that this π_S is feasible for the dual LP:

Dual feasibility

- Here are the dual LPs:

$$\begin{array}{rcll} \max w^T x & & & \\ \text{s.t. } x(S) & \leq & f(S) & \forall S \\ x(E) & = & f(E) & \\ x & & \text{free.} & \end{array} \qquad \begin{array}{rcll} \min \sum_{S \subseteq E} f(S) \pi_S & & & \\ \text{s.t. } \sum_{S \ni e} \pi_S & = & w_e & \\ \pi_S & \geq & 0 & S \neq E \\ \pi_E & & \text{free.} & \end{array}$$

- Define π_S like this: Put $\pi_S = w_{e_{i-1}} - w_{e_i}$ if $S = e_i^{\prec}$, $\pi_E = w_{e_n} - 0$ (using "$w_{e_{n+1}} = 0$"), and $\pi_S = 0$ otherwise.
- First, note that this π_S is feasible for the dual LP:
  - We chose $\prec$ s.t. $w_{e_{i-1}} - w_{e_i} \geq 0$, and so $\pi_S \geq 0$.

Dual feasibility

- Here are the dual LPs:

$$\begin{array}{rcll} \max w^T x & & & \\ \text{s.t. } x(S) & \leq & f(S) & \forall S \\ x(E) & = & f(E) & \\ x & & \text{free.} & \end{array} \qquad \begin{array}{rcll} \min \sum_{S \subseteq E} f(S)\pi_S & & & \\ \text{s.t. } \sum_{S \ni e} \pi_S & = & w_e & \\ \pi_S & \geq & 0 & S \neq E \\ \pi_E & & \text{free.} & \end{array}$$

- Define π_S like this: Put $\pi_S = w_{e_{i-1}} - w_{e_i}$ if $S = e_i^{\prec}$, $\pi_E = w_{e_n} - 0$ (using "$w_{e_{n+1}} = 0$"), and $\pi_S = 0$ otherwise.
- First, note that this π_S is feasible for the dual LP:
  - We chose $\prec$ s.t. $w_{e_{i-1}} - w_{e_i} \geq 0$, and so $\pi_S \geq 0$.
  - Now $\sum_{S \ni e_k} \pi_S = \sum_{i=k+1}^{n+1} (w_{e_{i-1}} - w_{e_i})$
    $= w_{e_k} - w_{e_{n+1}} = w_{e_k}$, as desired.

Optimality from duality

- For any $x \in B(f)$ and π feasible for the dual, note that

$$\begin{array}{rcl} w^T x & = & \sum_{e \in E} (\sum_{S \ni e} \pi_S) x_e \\ & = & \sum_{S \subseteq E} \pi_S \sum_{e \in S} x_e \\ & = & \sum_{S \subseteq E} \pi_S x(S) \\ & \leq & \sum_{S \subseteq E} \pi_S f(S). \end{array}$$

Optimality from duality

- For any $x \in B(f)$ and π feasible for the dual, note that

$$\begin{array}{rcl} w^T x & = & \sum_{e \in E} (\sum_{S \ni e} \pi_S) x_e \\ & = & \sum_{S \subseteq E} \pi_S \sum_{e \in S} x_e \\ & = & \sum_{S \subseteq E} \pi_S x(S) \\ & \leq & \sum_{S \subseteq E} \pi_S f(S). \end{array}$$

- Since we already proved that the Greedy output $x \in B(f)$ and our π is feasible, we only need to show that $w^T x = \sum_{S \subseteq E} \pi_S f(S)$.

Optimality from duality

- For any $x \in B(f)$ and π feasible for the dual, note that

$$\begin{array}{rcl} w^T x & = & \sum_{e \in E} (\sum_{S \ni e} \pi_S) x_e \\ & = & \sum_{S \subseteq E} \pi_S \sum_{e \in S} x_e \\ & = & \sum_{S \subseteq E} \pi_S x(S) \\ & \leq & \sum_{S \subseteq E} \pi_S f(S). \end{array}$$

- Since we already proved that the Greedy output $x \in B(f)$ and our π is feasible, we only need to show that $w^T x = \sum_{S \subseteq E} \pi_S f(S)$.
- Consider the above display. The only place there's an inequality is $\sum_{S \subseteq E} \pi_S x(S) \leq \sum_{S \subseteq E} \pi_S f(S)$.

Optimality from duality

- For any $x \in B(f)$ and π feasible for the dual, note that

$$\begin{array}{rcl} w^T x & = & \sum_{e \in E} (\sum_{S \ni e} \pi_S) x_e \\ & = & \sum_{S \subseteq E} \pi_S \sum_{e \in S} x_e \\ & = & \sum_{S \subseteq E} \pi_S x(S) \\ & \leq & \sum_{S \subseteq E} \pi_S f(S). \end{array}$$

- Since we already proved that the Greedy output $x \in B(f)$ and our π is feasible, we only need to show that $w^T x = \sum_{S \subseteq E} \pi_S f(S)$.
- Consider the above display. The only place there's an inequality is $\sum_{S \subseteq E} \pi_S x(S) \leq \sum_{S \subseteq E} \pi_S f(S)$.
  - If $\pi_S = 0$ then both sides are zero.

Optimality from duality

- For any $x \in B(f)$ and π feasible for the dual, note that

$$\begin{array}{rcl} w^T x & = & \sum_{e \in E} (\sum_{S \ni e} \pi_S) x_e \\ & = & \sum_{S \subseteq E} \pi_S \sum_{e \in S} x_e \\ & = & \sum_{S \subseteq E} \pi_S x(S) \\ & \leq & \sum_{S \subseteq E} \pi_S f(S). \end{array}$$

- Since we already proved that the Greedy output $x \in B(f)$ and our π is feasible, we only need to show that $w^T x = \sum_{S \subseteq E} \pi_S f(S)$.
- Consider the above display. The only place there's an inequality is $\sum_{S \subseteq E} \pi_S x(S) \leq \sum_{S \subseteq E} \pi_S f(S)$.
  - If $\pi_S = 0$ then both sides are zero.
  - If $\pi_S \neq 0$, then S is $e_k^{\prec}$ for some k.

Optimality from duality

- For any $x \in B(f)$ and π feasible for the dual, note that

$$\begin{array}{rcl} w^T x & = & \sum_{e \in E} (\sum_{S \ni e} \pi_S) x_e \\ & = & \sum_{S \subseteq E} \pi_S \sum_{e \in S} x_e \\ & = & \sum_{S \subseteq E} \pi_S x(S) \\ & \leq & \sum_{S \subseteq E} \pi_S f(S). \end{array}$$

- Since we already proved that the Greedy output $x \in B(f)$ and our π is feasible, we only need to show that $w^T x = \sum_{S \subseteq E} \pi_S f(S)$.
- Consider the above display. The only place there's an inequality is $\sum_{S \subseteq E} \pi_S x(S) \leq \sum_{S \subseteq E} \pi_S f(S)$.
  - If $\pi_S = 0$ then both sides are zero.
  - If $\pi_S \neq 0$, then S is $e_k^{\prec}$ for some k.
  - But then $x(S) = \sum_{i<k} x_{e_i} = \sum_{i<k} (f(e_i^{\prec} + e_i) - f(e_i^{\prec})) = f(e_{k-1}^{\prec} + e_{k-1}) - f(\emptyset) = f(e_k^{\prec}) = f(S)$.

Optimality from duality

- For any $x \in B(f)$ and π feasible for the dual, note that

$$\begin{array}{rcl} w^T x & = & \sum_{e \in E} (\sum_{S \ni e} \pi_S) x_e \\ & = & \sum_{S \subseteq E} \pi_S \sum_{e \in S} x_e \\ & = & \sum_{S \subseteq E} \pi_S x(S) \\ & \leq & \sum_{S \subseteq E} \pi_S f(S). \end{array}$$

- Since we already proved that the Greedy output $x \in B(f)$ and our π is feasible, we only need to show that $w^T x = \sum_{S \subseteq E} \pi_S f(S)$.
- Consider the above display. The only place there's an inequality is $\sum_{S \subseteq E} \pi_S x(S) \leq \sum_{S \subseteq E} \pi_S f(S)$.
  - If $\pi_S = 0$ then both sides are zero.
  - If $\pi_S \neq 0$, then S is $e_k^{\prec}$ for some k.
  - But then $x(S) = \sum_{i<k} x_{e_i} = \sum_{i<k} (f(e_i^{\prec} + e_i) - f(e_i^{\prec})) = f(e_{k-1}^{\prec} + e_{k-1}) - f(\emptyset) = f(e_k^{\prec}) = f(S)$.
  - Thus we get equality, and so x is (primal) optimal (and π is dual optimal).

Notes about the Greedy Algorithm

- The Greedy Algorithm takes $O(n\mathrm{EO} + n \log n)$ time:

Notes about the Greedy Algorithm

- The Greedy Algorithm takes $O(n\mathrm{EO} + n \log n)$ time:
  - It takes $O(n \log n)$ time to sort the w_e.

Notes about the Greedy Algorithm

- The Greedy Algorithm takes $O(n\mathrm{EO} + n \log n)$ time:
  - It takes $O(n \log n)$ time to sort the w_e.
  - There are n calls to $\mathcal{E}$ that cost $O(n\mathrm{EO})$.

Notes about the Greedy Algorithm

- The Greedy Algorithm takes $O(n\mathrm{EO} + n \log n)$ time:
  - It takes $O(n \log n)$ time to sort the w_e.
  - There are n calls to $\mathcal{E}$ that cost $O(n\mathrm{EO})$.
- It can be shown (see below) that the output x of Greedy is in fact a vertex of $B(f)$.

Notes about the Greedy Algorithm

- The Greedy Algorithm takes $O(n\mathrm{EO} + n \log n)$ time:
  - It takes $O(n \log n)$ time to sort the w_e.
  - There are n calls to $\mathcal{E}$ that cost $O(n\mathrm{EO})$.
- It can be shown (see below) that the output x of Greedy is in fact a vertex of $B(f)$.
  - When the input to Greedy is linear order $\prec$, we denote the output x by $v^{\prec}$.

Notes about the Greedy Algorithm

- The Greedy Algorithm takes $O(n\mathrm{EO} + n \log n)$ time:
  - It takes $O(n \log n)$ time to sort the w_e.
  - There are n calls to $\mathcal{E}$ that cost $O(n\mathrm{EO})$.
- It can be shown (see below) that the output x of Greedy is in fact a vertex of $B(f)$.
  - When the input to Greedy is linear order $\prec$, we denote the output x by $v^{\prec}$.
  - We have shown that $w^T x$ is maximized at $v^{\prec}$ for an order $\prec$ consistent with w, and so in fact these Greedy vertices are *all* the vertices of $B(f)$. Thus there are at most $n!$ vertices of $B(f)$.

Notes about the Greedy Algorithm

- The Greedy Algorithm takes $O(n\mathrm{EO} + n \log n)$ time:
  - It takes $O(n \log n)$ time to sort the w_e.
  - There are n calls to $\mathcal{E}$ that cost $O(n\mathrm{EO})$.
- It can be shown (see below) that the output x of Greedy is in fact a vertex of $B(f)$.
  - When the input to Greedy is linear order $\prec$, we denote the output x by $v^{\prec}$.
  - We have shown that $w^T x$ is maximized at $v^{\prec}$ for an order $\prec$ consistent with w, and so in fact these Greedy vertices are *all* the vertices of $B(f)$. Thus there are at most $n!$ vertices of $B(f)$.
  - Although $B(f)$ has 2^n constraints, the linear order $\prec$ is a succinct certificate that $v^{\prec} \in B(f)$.

Notes about the Greedy Algorithm

- The Greedy Algorithm takes $O(n\mathrm{EO} + n \log n)$ time:
  - It takes $O(n \log n)$ time to sort the w_e.
  - There are n calls to $\mathcal{E}$ that cost $O(n\mathrm{EO})$.
- It can be shown (see below) that the output x of Greedy is in fact a vertex of $B(f)$.
  - When the input to Greedy is linear order $\prec$, we denote the output x by $v^{\prec}$.
  - We have shown that $w^T x$ is maximized at $v^{\prec}$ for an order $\prec$ consistent with w, and so in fact these Greedy vertices are *all* the vertices of $B(f)$. Thus there are at most $n!$ vertices of $B(f)$.
  - Although $B(f)$ has 2^n constraints, the linear order $\prec$ is a succinct certificate that $v^{\prec} \in B(f)$.
  - This proves that $B(f) \neq \emptyset$.

Notes about the Greedy Algorithm

- The Greedy Algorithm takes $O(n\mathrm{EO} + n \log n)$ time:
  - It takes $O(n \log n)$ time to sort the w_e.
  - There are n calls to $\mathcal{E}$ that cost $O(n\mathrm{EO})$.
- It can be shown (see below) that the output x of Greedy is in fact a vertex of $B(f)$.
  - When the input to Greedy is linear order $\prec$, we denote the output x by $v^{\prec}$.
  - We have shown that $w^T x$ is maximized at $v^{\prec}$ for an order $\prec$ consistent with w, and so in fact these Greedy vertices are *all* the vertices of $B(f)$. Thus there are at most $n!$ vertices of $B(f)$.
  - Although $B(f)$ has 2^n constraints, the linear order $\prec$ is a succinct certificate that $v^{\prec} \in B(f)$.
  - This proves that $B(f) \neq \emptyset$.
  - Greedy works on $B(f)$ for *any* w; it works on $P(f)$ if $w \geq 0$.

Understanding the basis matrix for Greedy

- The basis matrix M for an LP is the submatrix induced by the columns of the variables not at their bounds, and the rows whose constraints are tight (satisfied with equality).

Understanding the basis matrix for Greedy

- The basis matrix M for an LP is the submatrix induced by the columns of the variables not at their bounds, and the rows whose constraints are tight (satisfied with equality).
  - Here all the x_e are free (do not have bounds) and so M includes columns for every $e \in E$.

Understanding the basis matrix for Greedy

- The basis matrix M for an LP is the submatrix induced by the columns of the variables not at their bounds, and the rows whose constraints are tight (satisfied with equality).
  - Here all the x_e are free (do not have bounds) and so M includes columns for every $e \in E$.
  - As we saw in the proof, the constraint for $S = e_k^{\prec}$ is tight for each $e_k \in E$.

Understanding the basis matrix for Greedy

- The basis matrix M for an LP is the submatrix induced by the columns of the variables not at their bounds, and the rows whose constraints are tight (satisfied with equality).
  - Here all the x_e are free (do not have bounds) and so M includes columns for every $e \in E$.
  - As we saw in the proof, the constraint for $S = e_k^{\prec}$ is tight for each $e_k \in E$.
- Therefore M is the lower triangular matrix:

$$M = \begin{array}{c} \\ e_2^{\prec} \\ e_3^{\prec} \\ \vdots \\ e_{n+1}^{\prec} \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & \dots & e_n \end{array} \\ \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix} \end{array}$$

More Greedy basis matrix

- Recall that M is the lower triangular matrix:

$$M = \begin{array}{c c} & \begin{array}{cccc} e_1 & e_2 & \dots & e_n \end{array} \\ \begin{array}{c} e_2^{\prec} \\ e_3^{\prec} \\ \vdots \\ e_{n+1}^{\prec} \end{array} & \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix} \end{array}$$

More Greedy basis matrix

- Recall that M is the lower triangular matrix:

$$M = \begin{array}{c c} & \begin{array}{cccc} e_1 & e_2 & \dots & e_n \end{array} \\ \begin{array}{c} e_2^{\prec} \\ e_3^{\prec} \\ \vdots \\ e_{n+1}^{\prec} \end{array} & \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix} \end{array}$$

- Let $b^{\prec}$ be the RHS $(f(e_2^{\prec}), f(e_3^{\prec}), \dots, f(e_{n+1}^{\prec}))$.

More Greedy basis matrix

- Recall that M is the lower triangular matrix:

$$M = \begin{array}{c c} & \begin{array}{cccc} e_1 & e_2 & \dots & e_n \end{array} \\ \begin{array}{c} e_2^{\prec} \\ e_3^{\prec} \\ \vdots \\ e_{n+1}^{\prec} \end{array} & \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix} \end{array}$$

- Let $b^{\prec}$ be the RHS $(f(e_2^{\prec}), f(e_3^{\prec}), \dots, f(e_{n+1}^{\prec}))$.
- Then our Greedy primal vector $v^{\prec}$ solves $Mv^{\prec} = b^{\prec}$.

More Greedy basis matrix

- Recall that M is the lower triangular matrix:

$$M = \begin{array}{c c} & \begin{array}{cccc} e_1 & e_2 & \dots & e_n \end{array} \\ \begin{array}{c} e_2^{\prec} \\ e_3^{\prec} \\ \vdots \\ e_{n+1}^{\prec} \end{array} & \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix} \end{array}$$

- Let $b^{\prec}$ be the RHS $(f(e_2^{\prec}), f(e_3^{\prec}), \dots, f(e_{n+1}^{\prec}))$.
- Then our Greedy primal vector $v^{\prec}$ solves $Mv^{\prec} = b^{\prec}$.
- Triangular systems like this are easy to solve, and indeed gives that $x_{e_i} = f(e_i^{\prec} + e_i) - f(e_i^{\prec})$.

More Greedy basis matrix

- Recall that M is the lower triangular matrix:

$$M = \begin{array}{c} \\ e_2^{\prec} \\ e_3^{\prec} \\ \vdots \\ e_{n+1}^{\prec} \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & \dots & e_n \end{array} \\ \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix} \end{array}$$

- Let $b^{\prec}$ be the RHS $(f(e_2^{\prec}), f(e_3^{\prec}), \dots, f(e_{n+1}^{\prec}))$.
- Then our Greedy primal vector $v^{\prec}$ solves $Mv^{\prec} = b^{\prec}$.
- Triangular systems like this are easy to solve, and indeed gives that $x_{e_i} = f(e_i^{\prec} + e_i) - f(e_i^{\prec})$.
- Duality says that the dual has the same basis matrix, and π restricted to the $e_i^{\prec}$ solves $\pi^T M = w^T$.

More Greedy basis matrix

- Recall that M is the lower triangular matrix:

$$M = \begin{array}{c c} & \begin{array}{cccc} e_1 & e_2 & \dots & e_n \end{array} \\ \begin{array}{c} e_2^{\prec} \\ e_3^{\prec} \\ \vdots \\ e_{n+1}^{\prec} \end{array} & \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix} \end{array}$$

- Let $b^{\prec}$ be the RHS $(f(e_2^{\prec}), f(e_3^{\prec}), \dots, f(e_{n+1}^{\prec}))$.
- Then our Greedy primal vector $v^{\prec}$ solves $Mv^{\prec} = b^{\prec}$.
- Triangular systems like this are easy to solve, and indeed gives that $x_{e_i} = f(e_i^{\prec} + e_i) - f(e_i^{\prec})$.
- Duality says that the dual has the same basis matrix, and π restricted to the $e_i^{\prec}$ solves $\pi^T M = w^T$.
- Again this triangular system easily solves to $\pi_{e_i^{\prec}} = w_{i-1} - w_i$.

More Greedy basis matrix

- Recall that M is the lower triangular matrix:

$$M = \begin{array}{c c} & \begin{array}{cccc} e_1 & e_2 & \dots & e_n \end{array} \\ \begin{array}{c} e_2^{\prec} \\ e_3^{\prec} \\ \vdots \\ e_{n+1}^{\prec} \end{array} & \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix} \end{array}$$

- Let $b^{\prec}$ be the RHS $(f(e_2^{\prec}), f(e_3^{\prec}), \dots, f(e_{n+1}^{\prec}))$.
- Then our Greedy primal vector $v^{\prec}$ solves $Mv^{\prec} = b^{\prec}$.
- Triangular systems like this are easy to solve, and indeed gives that $x_{e_i} = f(e_i^{\prec} + e_i) - f(e_i^{\prec})$.
- Duality says that the dual has the same basis matrix, and π restricted to the $e_i^{\prec}$ solves $\pi^T M = w^T$.
- Again this triangular system easily solves to $\pi_{e_i^{\prec}} = w_{i-1} - w_i$.
- This also shows that $v^{\prec}$ is a vertex, as it follows from M being nonsingular.