Short Course on Submodular Functions

S. Thomas McCormick   Maurice Queyranne

Sauder School of Business, UBC

JPOC Summer School, June 2013

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).
- Second half of the course: Maurice Queyranne on submodular function maximization (SFMax).

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).
- Second half of the course: Maurice Queyranne on submodular function maximization (SFMax).
- There will be three hours for students to work on homework problems and present solutions.

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).
- Second half of the course: Maurice Queyranne on submodular function maximization (SFMax).
- There will be three hours for students to work on homework problems and present solutions.
- The detailed schedule is:

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).
- Second half of the course: Maurice Queyranne on submodular function maximization (SFMax).
- There will be three hours for students to work on homework problems and present solutions.
- The detailed schedule is:
  1. TM lectures four hours on SFMin Monday afternoon.

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).
- Second half of the course: Maurice Queyranne on submodular function maximization (SFMax).
- There will be three hours for students to work on homework problems and present solutions.
- The detailed schedule is:
  1. TM lectures four hours on SFMin Monday afternoon.
     1.1 One hour problem session Tuesday morning.

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).
- Second half of the course: Maurice Queyranne on submodular function maximization (SFMax).
- There will be three hours for students to work on homework problems and present solutions.
- The detailed schedule is:
  1. TM lectures four hours on SFMin Monday afternoon.
     1.1 One hour problem session Tuesday morning.
  2. TM lectures three hours on SFMin Tuesday morning.

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).
- Second half of the course: Maurice Queyranne on submodular function maximization (SFMax).
- There will be three hours for students to work on homework problems and present solutions.
- The detailed schedule is:
  1. TM lectures four hours on SFMin Monday afternoon.
     1.1 One hour problem session Tuesday morning.
  2. TM lectures three hours on SFMin Tuesday morning.
     2.1 Lunch on Tuesday.

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).
- Second half of the course: Maurice Queyranne on submodular function maximization (SFMax).
- There will be three hours for students to work on homework problems and present solutions.
- The detailed schedule is:
  1. TM lectures four hours on SFMin Monday afternoon.
     1.1 One hour problem session Tuesday morning.
  2. TM lectures three hours on SFMin Tuesday morning.
     2.1 Lunch on Tuesday.
     2.2 One hour problem session Tuesday afternoon.

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).
- Second half of the course: Maurice Queyranne on submodular function maximization (SFMax).
- There will be three hours for students to work on homework problems and present solutions.
- The detailed schedule is:
  1. TM lectures four hours on SFMin Monday afternoon.
     - 1.1 One hour problem session Tuesday morning.
  2. TM lectures three hours on SFMin Tuesday morning.
     - 2.1 Lunch on Tuesday.
     - 2.2 One hour problem session Tuesday afternoon.
  3. MQ lectures three hours on SFMax Tuesday afternoon.

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).
- Second half of the course: Maurice Queyranne on submodular function maximization (SFMax).
- There will be three hours for students to work on homework problems and present solutions.
- The detailed schedule is:
  1. TM lectures four hours on SFMin Monday afternoon.
     1.1 One hour problem session Tuesday morning.
  2. TM lectures three hours on SFMin Tuesday morning.
     2.1 Lunch on Tuesday.
     2.2 One hour problem session Tuesday afternoon.
  3. MQ lectures three hours on SFMax Tuesday afternoon.
     3.1 One hour problem session Wednesday morning.

Teaching plan

- First half of the course: Tom McCormick on submodular function minimization (SFMin).
- Second half of the course: Maurice Queyranne on submodular function maximization (SFMax).
- There will be three hours for students to work on homework problems and present solutions.
- The detailed schedule is:
  1. TM lectures four hours on SFMin Monday afternoon.
     - 1.1 One hour problem session Tuesday morning.
  2. TM lectures three hours on SFMin Tuesday morning.
     - 2.1 Lunch on Tuesday.
     - 2.2 One hour problem session Tuesday afternoon.
  3. MQ lectures three hours on SFMax Tuesday afternoon.
     - 3.1 One hour problem session Wednesday morning.
  4. MQ lectures three hours on SFMax Wednesday morning.

Contents

Contents

Contents

Contents

Contents

Outline

Motivating Example

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

Motivating 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 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 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 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 $c : 2^E \rightarrow \mathbb{R}$.
- 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,

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

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,

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

- 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

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

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

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

  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.

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. □

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 $a(\emptyset) = 0$.

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 $a(\emptyset) = 0$.
- 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 $a(\emptyset) = 0$.
- 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 $a(\emptyset) = 0$.
- 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 $a(\emptyset) = 0$.
- 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 $a(\emptyset) = 0$.
- 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 $\delta(S) = \{i \,\text{—}\, j \mid i \in S,\ j \notin S\}$.

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 $w \ngeq 0$ then they are not necessarily submodular (homework).
  - The same is true for *undirected* graphs where we consider $\delta(S) = \{i \text{---} j \mid i \in S,\ j \notin S\}$.
  - 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 $\operatorname{cap}(S) = u(\delta^+(S + s))$, 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 $\mathrm{val}(x) = x(\delta^+(\{s\})) - x(\delta^-(\{s\}))$.

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: $0 \leq x_{ij} \leq u_{ij}$ 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: $0 \leq x_{ij} \leq u_{ij}$ 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:

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

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 $x_{ji} > 0$.

Algorithmic proof of Max Flow / Min Cut

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

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

- 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:

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

- 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 $S = \{i \in E \mid \exists \text{ aug. path from } s \text{ to } i \text{ w.r.t. } x\}$.

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 $S = \{i \in E \mid \exists \text{ aug. path from } s \text{ to } i \text{ w.r.t. } x\}$.
- 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.
  - Our main SFMin algorithm will be 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.
  - Our main SFMin algorithm will be 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

- Matroids: The rank function of a matroid.

Further examples which are all submodular

- 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

- 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.

Further examples which are all submodular

- 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.
- Entropy: The *Shannon entropy* of a random vector.

Further examples which are all submodular

- 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):

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

- 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):

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

- 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:

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

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., $S \subseteq T \implies f(S) \le f(T)$.
  - 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 $\max_{S \subseteq E : |S| \le B} f(S)$.

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 $\max_{S \subseteq E : |S| \leq B} f(S)$.

- 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 $\alpha \cdot \text{OPT}$.

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 $y \leq x$ 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. $x \in B(f)$.

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

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

The Greedy Algorithm

- 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

- 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

- 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

- 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

- 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

- 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

- 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

- 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

- 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

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

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

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

- 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:

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

More Greedy basis matrix

- Recall that M is the lower triangular matrix:

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

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

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 $Mv^{\prec} = b^{\prec}$.

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 $x_{e_i} = f(e_i^{\prec} + e_i) - f(e_i^{\prec})$.

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:

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

- 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 $\pi^T M = w^T$.
- 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:

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

- 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 $\pi^T M = w^T$.
- 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.

From vertices of $B(f)$ to edges of $B(f)$

- We now understand the vertices of $B(f)$ via Greedy.

From vertices of $B(f)$ to edges of $B(f)$

- We now understand the vertices of $B(f)$ via Greedy.
- To be able to move around in $B(f)$ we also need to understand its edges.

From vertices of $B(f)$ to edges of $B(f)$

- We now understand the vertices of $B(f)$ via Greedy.
- To be able to move around in $B(f)$ we also need to understand its edges.
- Suppose that $\prec$ looks like

$$e_1 e_2 \dots e_i l k e_{i+3} \dots e_n,$$

and $\prec'$ looks like

$$e_1 e_2 \dots e_i k l e_{i+3} \dots e_n;$$

we say that (l, k) are *consecutive* in $\prec$.

From vertices of $B(f)$ to edges of $B(f)$

- We now understand the vertices of $B(f)$ via Greedy.
- To be able to move around in $B(f)$ we also need to understand its edges.
- Suppose that $\prec$ looks like

$$e_1 e_2 \dots e_i l k e_{i+3} \dots e_n,$$

  and $\prec'$ looks like

$$e_1 e_2 \dots e_i k l e_{i+3} \dots e_n;$$

  we say that (l, k) are *consecutive* in $\prec$.
- For $e \in E$ define $\chi(e) \in \{0,1\}^E$ by $\chi(e)_e = 1$ and $\chi(e)_g = 0$ for $g \neq e$.

From vertices of $B(f)$ to edges of $B(f)$

- We now understand the vertices of $B(f)$ via Greedy.
- To be able to move around in $B(f)$ we also need to understand its edges.
- Suppose that $\prec$ looks like

$$e_1 e_2 \dots e_i l k e_{i+3} \dots e_n,$$

  and $\prec'$ looks like

$$e_1 e_2 \dots e_i k l e_{i+3} \dots e_n;$$

  we say that (l,k) are *consecutive* in $\prec$.
- For $e \in E$ define $\chi(e) \in \{0,1\}^E$ by $\chi(e)_e = 1$ and $\chi(e)_g = 0$ for $g \neq e$.
- We are going to show that $v^{\prec'} - v^{\prec} = \alpha(\chi_k - \chi_l)$ for a step length α.

Stepping along an edge

- Recall that $v^{\prec}$ comes from $e_1 e_2 \ldots e_i {\color{red}l}{\color{blue}k} e_{i+3} \ldots e_n$, and $\prec'$ comes from $e_1 e_2 \ldots e_i {\color{blue}k}{\color{red}l} e_{i+3} \ldots e_n$.

Stepping along an edge

- Recall that $v^{\prec}$ comes from $e_1 e_2 \ldots e_i l k e_{i+3} \ldots e_n$, and $\prec'$ comes from $e_1 e_2 \ldots e_i k l e_{i+3} \ldots e_n$.
- Notice that for $e \neq k, l$ we have that $e^{\prec} = e^{\prec'}$.

Stepping along an edge

- Recall that $v^{\prec}$ comes from $e_1e_2\dots e_i{\color{red}l}{\color{blue}k}e_{i+3}\dots e_n$, and $\prec'$ comes from $e_1e_2\dots e_i{\color{blue}k}{\color{red}l}e_{i+3}\dots e_n$.
- Notice that for $e \neq k,l$ we have that $e^{\prec} = e^{\prec'}$.
  - Thus for $e \neq k,l$ we have that $v_e^{\prec} = f(e^{\prec} + e) - f(e^{\prec}) = f(e^{\prec'} + e) - f(e^{\prec'}) = v_e^{\prec'}$.

Stepping along an edge

- Recall that $v^{\prec}$ comes from $e_1 e_2 \ldots e_i l k e_{i+3} \ldots e_n$, and $\prec'$ comes from $e_1 e_2 \ldots e_i k l e_{i+3} \ldots e_n$.
- Notice that for $e \neq k, l$ we have that $e^{\prec} = e^{\prec'}$.
  - Thus for $e \neq k, l$ we have that $v_e^{\prec} = f(e^{\prec} + e) - f(e^{\prec}) = f(e^{\prec'} + e) - f(e^{\prec'}) = v_e^{\prec'}$.
- For $e = k$ we have $v_k^{\prec} = f(k^{\prec} + k) - f(k^{\prec}) = f(l^{\prec} + k + l) - f(l^{\prec} + l)$ and $v_k^{\prec'} = f(k^{\prec'} + k) - f(k^{\prec'}) = f(l^{\prec} + k) - f(l^{\prec})$.

Stepping along an edge

- Recall that $v^{\prec}$ comes from $e_1 e_2 \dots e_i l k e_{i+3} \dots e_n$, and $\prec'$ comes from $e_1 e_2 \dots e_i k l e_{i+3} \dots e_n$.
- Notice that for $e \neq k, l$ we have that $e^{\prec} = e^{\prec'}$.
  - Thus for $e \neq k, l$ we have that $v_e^{\prec} = f(e^{\prec} + e) - f(e^{\prec}) = f(e^{\prec'} + e) - f(e^{\prec'}) = v_e^{\prec'}$.
- For $e = k$ we have $v_k^{\prec} = f(k^{\prec} + k) - f(k^{\prec}) = f(l^{\prec} + k + l) - f(l^{\prec} + l)$ and $v_k^{\prec'} = f(k^{\prec'} + k) - f(k^{\prec'}) = f(l^{\prec} + k) - f(l^{\prec})$.
- For $e = l$ we have $v_l^{\prec} = f(l^{\prec} + l) - f(l^{\prec}) = f(l^{\prec} + l) - f(l^{\prec})$ and $v_l^{\prec'} = f(l^{\prec'} + l) - f(l^{\prec'}) = f(l^{\prec} + k + l) - f(l^{\prec} + k)$.

Stepping along an edge

- Recall that $v^{\prec}$ comes from $e_1 e_2 \dots e_i l k e_{i+3} \dots e_n$, and $\prec'$ comes from $e_1 e_2 \dots e_i k l e_{i+3} \dots e_n$.
- Notice that for $e \neq k, l$ we have that $e^{\prec} = e^{\prec'}$.
  - Thus for $e \neq k, l$ we have that $v_e^{\prec} = f(e^{\prec} + e) - f(e^{\prec}) = f(e^{\prec'} + e) - f(e^{\prec'}) = v_e^{\prec'}$.
- For $e = k$ we have $v_k^{\prec} = f(k^{\prec} + k) - f(k^{\prec}) = f(l^{\prec} + k + l) - f(l^{\prec} + l)$ and $v_k^{\prec'} = f(k^{\prec'} + k) - f(k^{\prec'}) = f(l^{\prec} + k) - f(l^{\prec})$.
- For $e = l$ we have $v_l^{\prec} = f(l^{\prec} + l) - f(l^{\prec}) = f(l^{\prec} + l) - f(l^{\prec})$ and $v_l^{\prec'} = f(l^{\prec'} + l) - f(l^{\prec'}) = f(l^{\prec} + k + l) - f(l^{\prec} + k)$.
  - Define $\alpha = [f(l^{\prec} + l) - f(l^{\prec})] - [f(l^{\prec} + k + l) - f(l^{\prec} + k)]$.

Stepping along an edge

- Recall that $v^{\prec}$ comes from $e_1 e_2 \dots e_i l k e_{i+3} \dots e_n$, and $\prec'$ comes from $e_1 e_2 \dots e_i k l e_{i+3} \dots e_n$.
- Notice that for $e \neq k, l$ we have that $e^{\prec} = e^{\prec'}$.
  - Thus for $e \neq k, l$ we have that $v_e^{\prec} = f(e^{\prec} + e) - f(e^{\prec}) = f(e^{\prec'} + e) - f(e^{\prec'}) = v_e^{\prec'}$.
- For $e = k$ we have $v_k^{\prec} = f(k^{\prec} + k) - f(k^{\prec}) = f(l^{\prec} + k + l) - f(l^{\prec} + l)$ and $v_k^{\prec'} = f(k^{\prec'} + k) - f(k^{\prec'}) = f(l^{\prec} + k) - f(l^{\prec})$.
- For $e = l$ we have $v_l^{\prec} = f(l^{\prec} + l) - f(l^{\prec}) = f(l^{\prec} + l) - f(l^{\prec})$ and $v_l^{\prec'} = f(l^{\prec'} + l) - f(l^{\prec'}) = f(l^{\prec} + k + l) - f(l^{\prec} + k)$.
  - Define $\alpha = [f(l^{\prec} + l) - f(l^{\prec})] - [f(l^{\prec} + k + l) - f(l^{\prec} + k)]$.
  - By submodularity, $\alpha \geq 0$.

Stepping along an edge

- Recall that $v^{\prec}$ comes from $e_1e_2\ldots e_i lk e_{i+3}\ldots e_n$, and $\prec'$ comes from $e_1e_2\ldots e_i kl e_{i+3}\ldots e_n$.
- Notice that for $e \neq k, l$ we have that $e^{\prec} = e^{\prec'}$.
  - Thus for $e \neq k, l$ we have that $v_e^{\prec} = f(e^{\prec} + e) - f(e^{\prec}) = f(e^{\prec'} + e) - f(e^{\prec'}) = v_e^{\prec'}$.
- For $e = k$ we have $v_k^{\prec} = f(k^{\prec} + k) - f(k^{\prec}) = f(l^{\prec} + k + l) - f(l^{\prec} + l)$ and $v_k^{\prec'} = f(k^{\prec'} + k) - f(k^{\prec'}) = f(l^{\prec} + k) - f(l^{\prec})$.
- For $e = l$ we have $v_l^{\prec} = f(l^{\prec} + l) - f(l^{\prec}) = f(l^{\prec} + l) - f(l^{\prec})$ and $v_l^{\prec'} = f(l^{\prec'} + l) - f(l^{\prec'}) = f(l^{\prec} + k + l) - f(l^{\prec} + k)$.
  - Define $\alpha = [f(l^{\prec} + l) - f(l^{\prec})] - [f(l^{\prec} + k + l) - f(l^{\prec} + k)]$.
  - By submodularity, $\alpha \geq 0$.
  - Then we see that $v_l^{\prec'} = v_l^{\prec} - \alpha$, and $v_k^{\prec'} = v_k^{\prec} + \alpha$.

Stepping along an edge

- Recall that $v^{\prec}$ comes from $e_1 e_2 \dots e_i l k e_{i+3} \dots e_n$, and $\prec'$ comes from $e_1 e_2 \dots e_i k l e_{i+3} \dots e_n$.
- Notice that for $e \neq k, l$ we have that $e^{\prec} = e^{\prec'}$.
  - Thus for $e \neq k, l$ we have that $v_e^{\prec} = f(e^{\prec} + e) - f(e^{\prec}) = f(e^{\prec'} + e) - f(e^{\prec'}) = v_e^{\prec'}$.
- For $e = k$ we have $v_k^{\prec} = f(k^{\prec} + k) - f(k^{\prec}) = f(l^{\prec} + k + l) - f(l^{\prec} + l)$ and $v_k^{\prec'} = f(k^{\prec'} + k) - f(k^{\prec'}) = f(l^{\prec} + k) - f(l^{\prec})$.
- For $e = l$ we have $v_l^{\prec} = f(l^{\prec} + l) - f(l^{\prec}) = f(l^{\prec} + l) - f(l^{\prec})$ and $v_l^{\prec'} = f(l^{\prec'} + l) - f(l^{\prec'}) = f(l^{\prec} + k + l) - f(l^{\prec} + k)$.
  - Define $\alpha = [f(l^{\prec} + l) - f(l^{\prec})] - [f(l^{\prec} + k + l) - f(l^{\prec} + k)]$.
  - By submodularity, $\alpha \geq 0$.
  - Then we see that $v_l^{\prec'} = v_l^{\prec} - \alpha$, and $v_k^{\prec'} = v_k^{\prec} + \alpha$.
  - Intuition: as we move k earlier in $\prec$, $v_k^{\prec}$ gets bigger; as we move k later in $\prec$, $v_k^{\prec}$ gets smaller.

Exchange capacities

- We call this step length $\alpha = [f(l^{\prec} + l) - f(l^{\prec})] - [f(l^{\prec} + k + l) - f(l^{\prec} + k)]$ the exchange capacity of the consecutive pair (l, k), and denote it as $c(k, l; v^{\prec})$.

Exchange capacities

- We call this step length $\alpha = [f(l^{\prec} + l) - f(l^{\prec})] - [f(l^{\prec} + k + l) - f(l^{\prec} + k)]$ the exchange capacity of the consecutive pair (l, k), and denote it as $c(k, l; v^{\prec})$.
  - Since $x(E) = f(E)$ is a constraint of $B(f)$, all $x \in B(f)$ have the constant sum $f(E)$. Thus it is not a surprise that $|v_k^{\prec} - v_k^{\prec'}| = |v_l^{\prec} - v_l^{\prec'}| = c(k, l; v^{\prec})$.

Exchange capacities

- We call this step length $\alpha = [f(l^{\prec} + l) - f(l^{\prec})] - [f(l^{\prec} + k + l) - f(l^{\prec} + k)]$ the exchange capacity of the consecutive pair (l, k), and denote it as $c(k, l; v^{\prec})$.
  - Since $x(E) = f(E)$ is a constraint of $B(f)$, all $x \in B(f)$ have the constant sum $f(E)$. Thus it is not a surprise that $|v_k^{\prec} - v_k^{\prec'}| = |v_l^{\prec} - v_l^{\prec'}| = c(k, l; v^{\prec})$.
  - We have indeed shown that when (l, k) is consecutive in $\prec$, then $v^{\prec'} - v^{\prec} = c(k, l; v^{\prec})(\chi_k - \chi_l)$.

Exchange capacities

- We call this step length $\alpha = [f(l^{\prec} + l) - f(l^{\prec})] - [f(l^{\prec} + k + l) - f(l^{\prec} + k)]$ the exchange capacity of the consecutive pair (l, k), and denote it as $c(k, l; v^{\prec})$.
  - Since $x(E) = f(E)$ is a constraint of $B(f)$, all $x \in B(f)$ have the constant sum $f(E)$. Thus it is not a surprise that $|v_k^{\prec} - v_k^{\prec'}| = |v_l^{\prec} - v_l^{\prec'}| = c(k, l; v^{\prec})$.
  - We have indeed shown that when (l, k) is consecutive in $\prec$, then $v^{\prec'} - v^{\prec} = c(k, l; v^{\prec})(\chi_k - \chi_l)$.
- It turns out that all the edges of $B(f)$ come from consecutive exchanges like this.

Exchange capacities

- We call this step length $\alpha = [f(l^{\prec} + l) - f(l^{\prec})] - [f(l^{\prec} + k + l) - f(l^{\prec} + k)]$ the exchange capacity of the consecutive pair (l, k), and denote it as $c(k, l; v^{\prec})$.
  - Since $x(E) = f(E)$ is a constraint of $B(f)$, all $x \in B(f)$ have the constant sum $f(E)$. Thus it is not a surprise that $|v_k^{\prec} - v_k^{\prec'}| = |v_l^{\prec} - v_l^{\prec'}| = c(k, l; v^{\prec})$.
  - We have indeed shown that when (l, k) is consecutive in $\prec$, then $v^{\prec'} - v^{\prec} = c(k, l; v^{\prec})(\chi_k - \chi_l)$.
- It turns out that all the edges of $B(f)$ come from consecutive exchanges like this.
- Given some $x \in B(f)$ and $k, l \in E$, it is natural to wonder if we can compute the more general exchange capacity $c(k, l; x)$, which is the largest α such that $x + \alpha(\chi_k - \chi_l) \in B(f)$.

Exchange capacities

- We call this step length $\alpha = [f(l^{\prec} + l) - f(l^{\prec})] - [f(l^{\prec} + k + l) - f(l^{\prec} + k)]$ the exchange capacity of the consecutive pair (l, k), and denote it as $c(k, l; v^{\prec})$.
  - Since $x(E) = f(E)$ is a constraint of $B(f)$, all $x \in B(f)$ have the constant sum $f(E)$. Thus it is not a surprise that $|v_k^{\prec} - v_k^{\prec'}| = |v_l^{\prec} - v_l^{\prec'}| = c(k, l; v^{\prec})$.
  - We have indeed shown that when (l, k) is consecutive in $\prec$, then $v^{\prec'} - v^{\prec} = c(k, l; v^{\prec})(\chi_k - \chi_l)$.
- It turns out that all the edges of $B(f)$ come from consecutive exchanges like this.
- Given some $x \in B(f)$ and $k, l \in E$, it is natural to wonder if we can compute the more general exchange capacity $c(k, l; x)$, which is the largest α such that $x + \alpha(\chi_k - \chi_l) \in B(f)$.
  - Unfortunately it turns out that computing $c(k, l; x)$ is provably as difficult as SFMin.

Outline

An algorithmic framework for SFMin

- We now start to develop a framework for algorithms for SFMin (due to Cunningham) that resembles the Max Flow / Min Cut algorithms.

An algorithmic framework for SFMin

- We now start to develop a framework for algorithms for SFMin (due to Cunningham) that resembles the Max Flow / Min Cut algorithms.
- The framework starts by showing that SFMin can be modeled using a dual pair of linear program (due to Edmonds).

An algorithmic framework for SFMin

- We now start to develop a framework for algorithms for SFMin (due to Cunningham) that resembles the Max Flow / Min Cut algorithms.
- The framework starts by showing that SFMin can be modeled using a dual pair of linear program (due to Edmonds).
- However, the first weakly and strongly polynomial algorithms for SFMin came from a very different viewpoint.

An algorithmic framework for SFMin

- We now start to develop a framework for algorithms for SFMin (due to Cunningham) that resembles the Max Flow / Min Cut algorithms.
- The framework starts by showing that SFMin can be modeled using a dual pair of linear program (due to Edmonds).
- However, the first weakly and strongly polynomial algorithms for SFMin came from a very different viewpoint.
  - There is an equivalence between Separation and Optimization via the Ellipsoid Algorithm due to Grötschel, Lovász, and Schrijver.

An algorithmic framework for SFMin

- We now start to develop a framework for algorithms for SFMin (due to Cunningham) that resembles the Max Flow / Min Cut algorithms.
- The framework starts by showing that SFMin can be modeled using a dual pair of linear program (due to Edmonds).
- However, the first weakly and strongly polynomial algorithms for SFMin came from a very different viewpoint.
  - There is an equivalence between Separation and Optimization via the Ellipsoid Algorithm due to Grötschel, Lovász, and Schrijver.
  - For a certain polymatroid, its Separation problem is equivalent to SFMin.

An algorithmic framework for SFMin

- We now start to develop a framework for algorithms for SFMin (due to Cunningham) that resembles the Max Flow / Min Cut algorithms.
- The framework starts by showing that SFMin can be modeled using a dual pair of linear program (due to Edmonds).
- However, the first weakly and strongly polynomial algorithms for SFMin came from a very different viewpoint.
  - There is an equivalence between Separation and Optimization via the Ellipsoid Algorithm due to Grötschel, Lovász, and Schrijver.
  - For a certain polymatroid, its Separation problem is equivalent to SFMin.
  - The polymatroid's Optimization problem is equivalent to the LP we solved via Greedy.

An algorithmic framework for SFMin

- We now start to develop a framework for algorithms for SFMin (due to Cunningham) that resembles the Max Flow / Min Cut algorithms.
- The framework starts by showing that SFMin can be modeled using a dual pair of linear program (due to Edmonds).
- However, the first weakly and strongly polynomial algorithms for SFMin came from a very different viewpoint.
  - There is an equivalence between Separation and Optimization via the Ellipsoid Algorithm due to Grötschel, Lovász, and Schrijver.
  - For a certain polymatroid, its Separation problem is equivalent to SFMin.
  - The polymatroid's Optimization problem is equivalent to the LP we solved via Greedy.
  - Therefore Ellipsoid says that SFMin is (weakly) polynomial.

An algorithmic framework for SFMin

- We now start to develop a framework for algorithms for SFMin (due to Cunningham) that resembles the Max Flow / Min Cut algorithms.
- The framework starts by showing that SFMin can be modeled using a dual pair of linear program (due to Edmonds).
- However, the first weakly and strongly polynomial algorithms for SFMin came from a very different viewpoint.
  - There is an equivalence between Separation and Optimization via the Ellipsoid Algorithm due to Grötschel, Lovász, and Schrijver.
  - For a certain polymatroid, its Separation problem is equivalent to SFMin.
  - The polymatroid's Optimization problem is equivalent to the LP we solved via Greedy.
  - Therefore Ellipsoid says that SFMin is (weakly) polynomial.
  - GLS then extend this to show a strongly polynomial running time.

Edmonds' LP formulation of SFMin

- Recall that SFMin is $\min_{S \subseteq E} f(S)$. It is very unclear whether this can be formulated as an LP.

Edmonds' LP formulation of SFMin

- Recall that SFMin is $\min_{S \subseteq E} f(S)$. It is very unclear whether this can be formulated as an LP.
- Let's modify the dual LPs we used for Greedy by relaxing $x(E) = f(E)$ to just $x(E) \le f(E)$, putting an upper bound u on x in the primal, and replacing w by the all-ones vector $\mathbb{1}$:

$$\begin{array}{rcl} \max \mathbb{1}^T x & & \\ \text{s.t. } x(S) & \le & f(S) \\ x & \le & u \\ x & & \text{free.} \end{array} \qquad \begin{array}{rcl} \min u^T \sigma + \sum_{S \subseteq E} f(S) \pi_S & & \\ \text{s.t. } \sigma_e + \sum_{S \ni e} \pi_S & = & 1 \\ \sigma, \pi & \ge & 0 \end{array}$$

Edmonds' LP formulation of SFMin

- Recall that SFMin is $\min_{S \subseteq E} f(S)$. It is very unclear whether this can be formulated as an LP.
- Let's modify the dual LPs we used for Greedy by relaxing $x(E) = f(E)$ to just $x(E) \leq f(E)$, putting an upper bound u on x in the primal, and replacing w by the all-ones vector $\mathbb{1}$:

$$
\begin{array}{rcl}
\max \mathbb{1}^T x & & \\
\text{s.t. } x(S) & \leq & f(S) \\
x & \leq & u \\
x & & \text{free.}
\end{array}
\qquad
\begin{array}{rcl}
\min u^T \sigma + \sum_{S \subseteq E} f(S) \pi_S & & \\
\text{s.t. } \sigma_e + \sum_{S \ni e} \pi_S & = & 1 \\
\sigma, \pi & \geq & 0
\end{array}
$$

- These kinds of "combinatorial" LPs often have 0–1 optimal solutions.

Edmonds' LP formulation of SFMin

- Recall that SFMin is $\min_{S \subseteq E} f(S)$. It is very unclear whether this can be formulated as an LP.
- Let's modify the dual LPs we used for Greedy by relaxing $x(E) = f(E)$ to just $x(E) \leq f(E)$, putting an upper bound u on x in the primal, and replacing w by the all-ones vector $\mathbb{1}$:

$$\begin{array}{rcl} \max \mathbb{1}^T x & & \\ \text{s.t. } x(S) & \leq & f(S) \\ x & \leq & u \\ x & & \text{free.} \end{array} \qquad \begin{array}{rcl} \min u^T \sigma + \sum_{S \subseteq E} f(S) \pi_S & & \\ \text{s.t. } \sigma_e + \sum_{S \ni e} \pi_S & = & 1 \\ \sigma, \pi & \geq & 0 \end{array}$$

- These kinds of "combinatorial" LPs often have 0–1 optimal solutions.
- Even better, we guess (see below) that there exists an optimal solution to the dual where only one π_S is positive, say $\pi_{S^*} = 1$.

LP optimal structure

- We believe that there exists an optimal solution to the dual where only one π_S is positive, say $\pi_{S^*} = 1$.

LP optimal structure

- We believe that there exists an optimal solution to the dual where only one π_S is positive, say $\pi_{S^*} = 1$.
- Then the constraints $\sigma_e + \sum_{S \ni e} \pi_S = 1$ would force that $\sigma = \chi(E - S^*)$, and so the dual objective value would be $u(E - S^*) + f(S^*)$. Let's prove this.

LP optimal structure

- We believe that there exists an optimal solution to the dual where only one π_S is positive, say $\pi_{S^*} = 1$.
- Then the constraints $\sigma_e + \sum_{S \ni e} \pi_S = 1$ would force that $\sigma = \chi(E - S^*)$, and so the dual objective value would be $u(E - S^*) + f(S^*)$. Let's prove this.
  - (Weak duality:) $\mathbb{1}^T x = x(E) = x(S) + x(E - S) \leq f(S) + u(E - S)$. Thus we only have to show that this is satisfied with equality.

LP optimal structure

- We believe that there exists an optimal solution to the dual where only one π_S is positive, say $\pi_{S^*} = 1$.
- Then the constraints $\sigma_e + \sum_{S \ni e} \pi_S = 1$ would force that $\sigma = \chi(E - S^*)$, and so the dual objective value would be $u(E - S^*) + f(S^*)$. Let's prove this.
  - (Weak duality:) $\mathbb{1}^T x = x(E) = x(S) + x(E - S) \leq f(S) + u(E - S)$. Thus we only have to show that this is satisfied with equality.
  - Suppose that x^* is primal optimal, and S and T are both x^*-tight, i.e., $x^*(S) = f(S)$ and $x^*(T) = f(T)$. Then (homework) both $S \cap T$ and $S \cup T$ are also x^*-tight.

LP optimal structure

- We believe that there exists an optimal solution to the dual where only one π_S is positive, say $\pi_{S^*} = 1$.
- Then the constraints $\sigma_e + \sum_{S \ni e} \pi_S = 1$ would force that $\sigma = \chi(E - S^*)$, and so the dual objective value would be $u(E - S^*) + f(S^*)$. Let's prove this.
  - (Weak duality:) $\mathbb{1}^T x = x(E) = x(S) + x(E - S) \leq f(S) + u(E - S)$. Thus we only have to show that this is satisfied with equality.
  - Suppose that x^* is primal optimal, and S and T are both x^*-tight, i.e., $x^*(S) = f(S)$ and $x^*(T) = f(T)$. Then (homework) both $S \cap T$ and $S \cup T$ are also x^*-tight.
  - Thus we can take the union of all x^*-tight sets to get S^*, which is also x^*-tight.

LP optimal structure

- We believe that there exists an optimal solution to the dual where only one π_S is positive, say $\pi_{S^*} = 1$.
- Then the constraints $\sigma_e + \sum_{S \ni e} \pi_S = 1$ would force that $\sigma = \chi(E - S^*)$, and so the dual objective value would be $u(E - S^*) + f(S^*)$. Let's prove this.
  - (Weak duality:) $\mathbb{1}^T x = x(E) = x(S) + x(E - S) \leq f(S) + u(E - S)$. Thus we only have to show that this is satisfied with equality.
  - Suppose that x^* is primal optimal, and S and T are both x^*-tight, i.e., $x^*(S) = f(S)$ and $x^*(T) = f(T)$. Then (homework) both $S \cap T$ and $S \cup T$ are also x^*-tight.
  - Thus we can take the union of all x^*-tight sets to get S^*, which is also x^*-tight.
  - If $x_e^* < u_e$ then we must have that $e \in S^*$; if not, then we could feasibly increase x_e^*, contradicting optimality. Thus $x_e = u_e$ for all $e \notin S^*$.

LP optimal structure

- We believe that there exists an optimal solution to the dual where only one π_S is positive, say $\pi_{S^*} = 1$.
- Then the constraints $\sigma_e + \sum_{S \ni e} \pi_S = 1$ would force that $\sigma = \chi(E - S^*)$, and so the dual objective value would be $u(E - S^*) + f(S^*)$. Let's prove this.
  - (Weak duality:) $\mathbb{1}^T x = x(E) = x(S) + x(E - S) \leq f(S) + u(E - S)$. Thus we only have to show that this is satisfied with equality.
  - Suppose that x^* is primal optimal, and S and T are both x^*-tight, i.e., $x^*(S) = f(S)$ and $x^*(T) = f(T)$. Then (homework) both $S \cap T$ and $S \cup T$ are also x^*-tight.
  - Thus we can take the union of all x^*-tight sets to get S^*, which is also x^*-tight.
  - If $x_e^* < u_e$ then we must have that $e \in S^*$; if not, then we could feasibly increase x_e^*, contradicting optimality. Thus $x_e = u_e$ for all $e \notin S^*$.
  - Thus $x(E) = x(S^*) + x(E - S^*) = f(S^*) + u(E - S^*)$, proving that S^* induces a dual optimal solution.

Specialize the LP to get SFMin

- Our LP strong duality says that $\max_{x \in P(f): x \leq u} x(E) = \min_{S \subseteq E}(f(S) + u(E - S))$.

Specialize the LP to get SFMin

- Our LP strong duality says that $\max_{x \in P(f): x \le u} x(E) = \min_{S \subseteq E}(f(S) + u(E - S))$.
- If we choose $u = 0$ then we get $\max_{x \in P(f): x \le 0} x(E) = \min_{S \subseteq E} f(S) + 0$. This dual LP is just SFMin!

Specialize the LP to get SFMin

- Our LP strong duality says that $\max_{x \in P(f): x \leq u} x(E) = \min_{S \subseteq E}(f(S) + u(E - S))$.
- If we choose $u = 0$ then we get $\max_{x \in P(f): x \leq 0} x(E) = \min_{S \subseteq E} f(S) + 0$. This dual LP is just SFMin!
- For $y \in \mathbb{R}^E$ define $y^- \in \mathbb{R}^E$ via $y_e^- = \min(y_e, 0) \leq 0$.

Specialize the LP to get SFMin

- Our LP strong duality says that $\max_{x \in P(f): x \le u} x(E) = \min_{S \subseteq E}(f(S) + u(E - S))$.
- If we choose $u = 0$ then we get $\max_{x \in P(f): x \le 0} x(E) = \min_{S \subseteq E} f(S) + 0$. This dual LP is just SFMin!
- For $y \in \mathbb{R}^E$ define $y^- \in \mathbb{R}^E$ via $y_e^- = \min(y_e, 0) \le 0$.
- If $y \in B(f)$ then $y^- \le 0$ and $y^- \in P(F)$, so it is primal feasible.

Specialize the LP to get SFMin

- Our LP strong duality says that $\max_{x \in P(f): x \le u} x(E) = \min_{S \subseteq E}(f(S) + u(E - S))$.
- If we choose $u = 0$ then we get $\max_{x \in P(f): x \le 0} x(E) = \min_{S \subseteq E} f(S) + 0$. This dual LP is just SFMin!
- For $y \in \mathbb{R}^E$ define $y^- \in \mathbb{R}^E$ via $y_e^- = \min(y_e, 0) \le 0$.
- If $y \in B(f)$ then $y^- \le 0$ and $y^- \in P(F)$, so it is primal feasible.
- We now want to show the converse, that if $x \in P(f)$ and $x \le 0$, then there is some $y \in B(f)$ with $y \ge x$ and $y^- = x$.

Specialize the LP to get SFMin

- Our LP strong duality says that $\max_{x \in P(f): x \leq u} x(E) = \min_{S \subseteq E}(f(S) + u(E - S))$.
- If we choose $u = 0$ then we get $\max_{x \in P(f): x \leq 0} x(E) = \min_{S \subseteq E} f(S) + 0$. This dual LP is just SFMin!
- For $y \in \mathbb{R}^E$ define $y^- \in \mathbb{R}^E$ via $y_e^- = \min(y_e, 0) \leq 0$.
- If $y \in B(f)$ then $y^- \leq 0$ and $y^- \in P(F)$, so it is primal feasible.
- We now want to show the converse, that if $x \in P(f)$ and $x \leq 0$, then there is some $y \in B(f)$ with $y \geq x$ and $y^- = x$.
- We know that an optimal $x^* \in P(f)$ with $x^* \leq 0$ looks like:

$$(\underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{not in any } x^*\text{-tight set}}\ \underbrace{0\ 0\ 0\ 0\ -\ -\ -\ -\ -\ -\ -\ -}_{S^* = \text{ biggest } x^*\text{-tight set}})$$

Moving from $P(f)$ to $B(f)$

- We know that an optimal $x^* \in P(f)$ with $x^* \leq 0$ looks like:

$$(\underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{not in any } x^*\text{-tight set}}\ \underbrace{0\ 0\ 0\ 0\ -\ -\ -\ -\ -\ -\ -\ -}_{S^* = \text{ biggest } x^*\text{-tight set}})$$

Moving from $P(f)$ to $B(f)$

- We know that an optimal $x^* \in P(f)$ with $x^* \leq 0$ looks like:

$$(\underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{not in any } x^*\text{-tight set}}\ \underbrace{0\ 0\ 0\ 0\ -\ -\ -\ -\ -\ -\ -\ -}_{S^* = \text{ biggest } x^*\text{-tight set}})$$

- Set $y = x^*$ and pick some $e \notin S^*$ and increase y_e (making it positive) until it becomes tight (there is an exponential but finite number of constraints to check).

Moving from $P(f)$ to $B(f)$

- We know that an optimal $x^* \in P(f)$ with $x^* \leq 0$ looks like:

$$(\underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{not in any } x^*\text{-tight set}}\ \underbrace{0\ 0\ 0\ 0\ -\ -\ -\ -\ -\ -\ -\ -}_{S^* = \text{ biggest } x^*\text{-tight set}})$$

- Set $y = x^*$ and pick some $e \notin S^*$ and increase y_e (making it positive) until it becomes tight (there is an exponential but finite number of constraints to check).
- Continue until every e is contained in an y-tight set.

Moving from $P(f)$ to $B(f)$

- We know that an optimal $x^* \in P(f)$ with $x^* \leq 0$ looks like:

$$(\underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{not in any } x^*\text{-tight set}}\ \underbrace{0\ 0\ 0\ 0\ -\ -\ -\ -\ -\ -\ -\ -}_{S^* = \text{ biggest } x^*\text{-tight set}})$$

- Set $y = x^*$ and pick some $e \notin S^*$ and increase y_e (making it positive) until it becomes tight (there is an exponential but finite number of constraints to check).
- Continue until every e is contained in an y-tight set.
- Now every e is in an y-tight set, and so E is tight, so the new y is in $B(f)$. It looks like:

$$(\underbrace{+\ +\ +\ +\ +\ +\ +\ +}_{\text{increased elements}}\ 0\ 0\ 0\ 0\ \underbrace{0\ 0\ 0\ 0\ -\ -\ -\ -\ -\ -\ -\ -}_{S^* \text{ is still } y\text{-tight}})$$

Moving from $P(f)$ to $B(f)$

- We know that an optimal $x^* \in P(f)$ with $x^* \leq 0$ looks like:

$$(\underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{not in any } x^*\text{-tight set}}\ \underbrace{0\ 0\ 0\ 0\ -\ -\ -\ -\ -\ -\ -\ -}_{S^* = \text{ biggest } x^*\text{-tight set}})$$

- Set $y = x^*$ and pick some $e \notin S^*$ and increase y_e (making it positive) until it becomes tight (there is an exponential but finite number of constraints to check).
- Continue until every e is contained in an y-tight set.
- Now every e is in an y-tight set, and so E is tight, so the new y is in $B(f)$. It looks like:

$$(\underbrace{+\ +\ +\ +\ +\ +\ +\ +}_{\text{increased elements}}\ 0\ 0\ 0\ 0\ \underbrace{0\ 0\ 0\ 0\ -\ -\ -\ -\ -\ -\ -\ -}_{S^* \text{ is still } y\text{-tight}})$$

- Thus we can use the modified primal LP $\max_{y \in B(f)} y^-(E)$.

Moving from $P(f)$ to $B(f)$

- We know that an optimal $x^* \in P(f)$ with $x^* \leq 0$ looks like:

$$(\underbrace{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0}_{\text{not in any } x^*\text{-tight set}}\;\underbrace{0\,0\,0\,0\;-\;-\;-\;-\;-\;-\;-\;-}_{S^* = \text{ biggest } x^*\text{-tight set}})$$

- Set $y = x^*$ and pick some $e \notin S^*$ and increase y_e (making it positive) until it becomes tight (there is an exponential but finite number of constraints to check).
- Continue until every e is contained in an y-tight set.
- Now every e is in an y-tight set, and so E is tight, so the new y is in $B(f)$. It looks like:

$$(\underbrace{+\;+\;+\;+\;+\;+\;+\;+}_{\text{increased elements}}\,0\,0\,0\,0\,\underbrace{0\,0\,0\,0\;-\;-\;-\;-\;-\;-\;-\;-}_{S^* \text{ is still } y\text{-tight}})$$

- Thus we can use the modified primal LP $\max_{y \in B(f)} y^-(E)$.
  - This is the form of the LP that we will use.

Moving from $P(f)$ to $B(f)$

- We know that an optimal $x^* \in P(f)$ with $x^* \leq 0$ looks like:

$$(\underbrace{0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0}_{\text{not in any } x^*\text{-tight set}}\;\underbrace{0\;0\;0\;0\;-\;-\;-\;-\;-\;-\;-\;-}_{S^* = \text{ biggest } x^*\text{-tight set}})$$

- Set $y = x^*$ and pick some $e \notin S^*$ and increase y_e (making it positive) until it becomes tight (there is an exponential but finite number of constraints to check).
- Continue until every e is contained in an y-tight set.
- Now every e is in an y-tight set, and so E is tight, so the new y is in $B(f)$. It looks like:

$$(\underbrace{+\;+\;+\;+\;+\;+\;+\;+}_{\text{increased elements}}\;0\;0\;0\;0\;\underbrace{0\;0\;0\;0\;-\;-\;-\;-\;-\;-\;-\;-}_{S^* \text{ is still } y\text{-tight}})$$

- Thus we can use the modified primal LP $\max_{y \in B(f)} y^-(E)$.
  - This is the form of the LP that we will use.
  - This LP is quite close to the Greedy LP, except that the objective is the piecewise linear $y^-(E)$ instead of $x(E)$, and this makes solving the problem *much* harder.

SFMin weak duality, complementary slackness

- Here is weak duality for these LPs:

$$\begin{array}{rcll} y^-(E) & \leq & y^-(S) & \text{tight if } y_e < 0 \implies e \in S \\ & \leq & y(S) & \text{tight if } e \in S \implies y_e \leq 0 \\ & \leq & f(S) & \text{tight if } S \text{ is } y\text{-tight.} \end{array}$$

Complementary slackness is equivalent to the tightness conditions that ensure that each inequality is an equality.

SFMin weak duality, complementary slackness

- Here is weak duality for these LPs:

$$\begin{array}{rcll} y^-(E) & \leq & y^-(S) & \text{tight if } y_e < 0 \implies e \in S \\ & \leq & y(S) & \text{tight if } e \in S \implies y_e \leq 0 \\ & \leq & f(S) & \text{tight if } S \text{ is } y\text{-tight.} \end{array}$$

  Complementary slackness is equivalent to the tightness conditions that ensure that each inequality is an equality.

- Therefore an optimal y and S look like:

$$y = (\underbrace{- \ - \ - \ - \ 0\,0\,0\,0\,0\,0}_{S \text{ includes all } -, \text{ no } +}\ 0\,0\,0\,0\ + \ + \ + \ + \ + \ + +)$$

SFMin weak duality, complementary slackness

- Here is weak duality for these LPs:

$$\begin{array}{rcll} y^-(E) & \leq & y^-(S) & \text{tight if } y_e < 0 \implies e \in S \\ & \leq & y(S) & \text{tight if } e \in S \implies y_e \leq 0 \\ & \leq & f(S) & \text{tight if } S \text{ is } y\text{-tight.} \end{array}$$

  Complementary slackness is equivalent to the tightness conditions that ensure that each inequality is an equality.

- Therefore an optimal y and S look like:

$$y = (\underbrace{- \; - \; - \; - \; 0\,0\,0\,0\,0\,0}_{S \text{ includes all } -, \text{ no } +}\; 0\,0\,0\,0 \; + \; + \; + \; + \; + \; + +)$$

- If we can achieve this picture along with $y(S) = f(S)$, it *proves* that y and S jointly solve SFMin.

SFMin weak duality, complementary slackness

- Here is weak duality for these LPs:

$$\begin{aligned} y^-(E) &\le y^-(S) && \text{tight if } y_e < 0 \implies e \in S \\ &\le y(S) && \text{tight if } e \in S \implies y_e \le 0 \\ &\le f(S) && \text{tight if } S \text{ is } y\text{-tight.} \end{aligned}$$

  Complementary slackness is equivalent to the tightness conditions that ensure that each inequality is an equality.

- Therefore an optimal y and S look like:

$$y = (\underbrace{- \ - \ - \ - \ 0\,0\,0\,0\,0\,0}_{S \text{ includes all } -, \text{ no } +}\ 0\,0\,0\,0 \ + \ + \ + \ + \ + \ + +)$$

- If we can achieve this picture along with $y(S) = f(S)$, it *proves* that y and S jointly solve SFMin.
- Or does it? What is missing? . . . .

How do we know that $y \in B(f)$?

- How can we verify that $y \in B(f)$? There are 2^n inequalities to check.

How do we know that $y \in B(f)$?

- How can we verify that $y \in B(f)$? There are 2^n inequalities to check.
- Here is a clever way to do it (Cunningham):

How do we know that $y \in B(f)$?

- How can we verify that $y \in B(f)$? There are 2^n inequalities to check.
- Here is a clever way to do it (Cunningham):
  1. $B(f)$ is bounded, and so it is the convex hull of its vertices, i.e., $y \in B(f)$ iff y is a convex combination of vertices of $B(f)$.

How do we know that $y \in B(f)$?

- How can we verify that $y \in B(f)$? There are 2^n inequalities to check.
- Here is a clever way to do it (Cunningham):
  1. $B(f)$ is bounded, and so it is the convex hull of its vertices, i.e., $y \in B(f)$ iff y is a convex combination of vertices of $B(f)$.
  2. We know that all vertices of $B(f)$ come from Greedy applied to linear orders, which have succinct certificates.

How do we know that $y \in B(f)$?

- How can we verify that $y \in B(f)$? There are 2^n inequalities to check.
- Here is a clever way to do it (Cunningham):
  1. $B(f)$ is bounded, and so it is the convex hull of its vertices, i.e., $y \in B(f)$ iff y is a convex combination of vertices of $B(f)$.
  2. We know that all vertices of $B(f)$ come from Greedy applied to linear orders, which have succinct certificates.
  3. Carathéodory's Theorem says that in fact there is always a convex hull representation of y using at most n vertices.

How do we know that $y \in B(f)$?

- How can we verify that $y \in B(f)$? There are 2^n inequalities to check.
- Here is a clever way to do it (Cunningham):
  1. $B(f)$ is bounded, and so it is the convex hull of its vertices, i.e., $y \in B(f)$ iff y is a convex combination of vertices of $B(f)$.
  2. We know that all vertices of $B(f)$ come from Greedy applied to linear orders, which have succinct certificates.
  3. Carathéodory's Theorem says that in fact there is always a convex hull representation of y using at most n vertices.
- Therefore the algorithms will keep a representation of y like this:

How do we know that $y \in B(f)$?

- How can we verify that $y \in B(f)$? There are 2^n inequalities to check.
- Here is a clever way to do it (Cunningham):
  1. $B(f)$ is bounded, and so it is the convex hull of its vertices, i.e., $y \in B(f)$ iff y is a convex combination of vertices of $B(f)$.
  2. We know that all vertices of $B(f)$ come from Greedy applied to linear orders, which have succinct certificates.
  3. Carathéodory's Theorem says that in fact there is always a convex hull representation of y using at most n vertices.
- Therefore the algorithms will keep a representation of y like this:
  - We have an index set $\mathcal{I}$ of size $O(n)$.

How do we know that $y \in B(f)$?

- How can we verify that $y \in B(f)$? There are 2^n inequalities to check.
- Here is a clever way to do it (Cunningham):
  1. $B(f)$ is bounded, and so it is the convex hull of its vertices, i.e., $y \in B(f)$ iff y is a convex combination of vertices of $B(f)$.
  2. We know that all vertices of $B(f)$ come from Greedy applied to linear orders, which have succinct certificates.
  3. Carathéodory's Theorem says that in fact there is always a convex hull representation of y using at most n vertices.
- Therefore the algorithms will keep a representation of y like this:
  - We have an index set $\mathcal{I}$ of size $O(n)$.
  - For each $i \in \mathcal{I}$ we have a linear order $\prec_i$ with associated Greedy vertex v^i.

How do we know that $y \in B(f)$?

- How can we verify that $y \in B(f)$? There are 2^n inequalities to check.
- Here is a clever way to do it (Cunningham):
  1. $B(f)$ is bounded, and so it is the convex hull of its vertices, i.e., $y \in B(f)$ iff y is a convex combination of vertices of $B(f)$.
  2. We know that all vertices of $B(f)$ come from Greedy applied to linear orders, which have succinct certificates.
  3. Carathéodory's Theorem says that in fact there is always a convex hull representation of y using at most n vertices.
- Therefore the algorithms will keep a representation of y like this:
  - We have an index set $\mathcal{I}$ of size $O(n)$.
  - For each $i \in \mathcal{I}$ we have a linear order $\prec_i$ with associated Greedy vertex v^i.
  - We keep multipliers $\lambda_i \geq 0$ for $i \in \mathcal{I}$ satisfying $\sum_{i \in \mathcal{I}} \lambda_i = 1$.

How do we know that $y \in B(f)$?

- How can we verify that $y \in B(f)$? There are 2^n inequalities to check.
- Here is a clever way to do it (Cunningham):
  1. $B(f)$ is bounded, and so it is the convex hull of its vertices, i.e., $y \in B(f)$ iff y is a convex combination of vertices of $B(f)$.
  2. We know that all vertices of $B(f)$ come from Greedy applied to linear orders, which have succinct certificates.
  3. Carathéodory's Theorem says that in fact there is always a convex hull representation of y using at most n vertices.
- Therefore the algorithms will keep a representation of y like this:
  - We have an index set $\mathcal{I}$ of size $O(n)$.
  - For each $i \in \mathcal{I}$ we have a linear order $\prec_i$ with associated Greedy vertex v^i.
  - We keep multipliers $\lambda_i \geq 0$ for $i \in \mathcal{I}$ satisfying $\sum_{i \in \mathcal{I}} \lambda_i = 1$.
  - Then $y = \sum_{i \in \mathcal{I}} \lambda_i v^i$ is a succinct certificate proving that $y \in B(f)$.

Keeping $|\mathcal{I}| = O(n)$

- As the algorithms proceed, they will add new indices to $\mathcal{I}$ (and possibly delete some old indices), and so $|\mathcal{I}|$ grows over time.

Keeping $|\mathcal{I}| = O(n)$

- As the algorithms proceed, they will add new indices to $\mathcal{I}$ (and possibly delete some old indices), and so $|\mathcal{I}|$ grows over time.
- When $\mathcal{I}$ becomes too large, from time to time we need to "Carathéodory-ize" it and bring its size back down to n.

Keeping $|\mathcal{I}| = O(n)$

- As the algorithms proceed, they will add new indices to $\mathcal{I}$ (and possibly delete some old indices), and so $|\mathcal{I}|$ grows over time.
- When $\mathcal{I}$ becomes too large, from time to time we need to "Carathéodory-ize" it and bring its size back down to n.
- Let V be the matrix with $|E| + 1$ rows and $|\mathcal{I}|$ columns which has a row of all ones at the top, and whose column i otherwise is v^i.

Keeping $|\mathcal{I}| = O(n)$

- As the algorithms proceed, they will add new indices to $\mathcal{I}$ (and possibly delete some old indices), and so $|\mathcal{I}|$ grows over time.
- When $\mathcal{I}$ becomes too large, from time to time we need to "Carathéodory-ize" it and bring its size back down to n.
- Let V be the matrix with $|E| + 1$ rows and $|\mathcal{I}|$ columns which has a row of all ones at the top, and whose column i otherwise is v^i.
- Therefore we keep the equation $V\lambda = (1 \quad y)$.

Keeping $|\mathcal{I}| = O(n)$

- As the algorithms proceed, they will add new indices to $\mathcal{I}$ (and possibly delete some old indices), and so $|\mathcal{I}|$ grows over time.
- When $\mathcal{I}$ becomes too large, from time to time we need to "Carathéodory-ize" it and bring its size back down to n.
- Let V be the matrix with $|E| + 1$ rows and $|\mathcal{I}|$ columns which has a row of all ones at the top, and whose column i otherwise is v^i.
- Therefore we keep the equation $V\lambda = (1 \quad y)$.
- The task of subroutine $\textsc{ReduceV}$ is to eliminate redundant columns of V while maintaining $V\lambda = (1 \quad y)$ and $\lambda \geq 0$.

Keeping $|\mathcal{I}| = O(n)$

- As the algorithms proceed, they will add new indices to $\mathcal{I}$ (and possibly delete some old indices), and so $|\mathcal{I}|$ grows over time.
- When $\mathcal{I}$ becomes too large, from time to time we need to "Carathéodory-ize" it and bring its size back down to n.
- Let V be the matrix with $|E| + 1$ rows and $|\mathcal{I}|$ columns which has a row of all ones at the top, and whose column i otherwise is v^i.
- Therefore we keep the equation $V\lambda = (1 \quad y)$.
- The task of subroutine $\textsc{ReduceV}$ is to eliminate redundant columns of V while maintaining $V\lambda = (1 \quad y)$ and $\lambda \geq 0$.
- This can be done with standard linear algebra techniques in $O(n^3)$ time.

Outline of a generic SFMin algorithm

- We keep linear orders $\prec_i$ with associated v^i, and $y \in B(f)$ as $y = \sum_{i \in \mathcal{I}} \lambda_i v^i$.

Outline of a generic SFMin algorithm

- We keep linear orders $\prec_i$ with associated v^i, and $y \in B(f)$ as $y = \sum_{i \in \mathcal{I}} \lambda_i v^i$.
- Suppose that y looks like:

$$y = (\underbrace{- \; - \; - \; - \; - \, -}_{S^-(y)} \; \underbrace{0 \; 0 \; 0 \; 0 \; 0}_{S^0(y)} \; \underbrace{+ \; + \; + \, +}_{S^+(y)})$$

Outline of a generic SFMin algorithm

- We keep linear orders $\prec_i$ with associated v^i, and $y \in B(f)$ as $y = \sum_{i \in \mathcal{I}} \lambda_i v^i$.
- Suppose that y looks like:

$$y = (\underbrace{- \ - \ - \ - \ - \ -}_{S^-(y)} \ \underbrace{0\ 0\ 0\ 0\ 0}_{S^0(y)} \ \underbrace{+ \ + \ + \ +}_{S^+(y)})$$

- To maximize $y^-(E)$ ($\iff \min_S y^+(E)$), we want to increase y_e for some $e \in S^-(y)$ (or decrease y_e for some $e \in S^+(y)$).

Outline of a generic SFMin algorithm

- We keep linear orders $\prec_i$ with associated v^i, and $y \in B(f)$ as $y = \sum_{i \in \mathcal{I}} \lambda_i v^i$.
- Suppose that y looks like:

$$y = (\underbrace{- \;\; - \;\; - \;\; - \;\; - \; -}_{S^-(y)} \; \underbrace{0\;0\;0\;0\;0}_{S^0(y)} \; \underbrace{+ \;\; + \;\; + \; +}_{S^+(y)})$$

- To maximize $y^-(E)$ ($\iff \min_S y^+(E)$), we want to increase y_e for some $e \in S^-(y)$ (or decrease y_e for some $e \in S^+(y)$).
  - We know that y_e increases if we move e to the left in some $\prec_i$, and y_e decreases if we move e to the right in some $\prec_i$.

Outline of a generic SFMin algorithm

- We keep linear orders $\prec_i$ with associated v^i, and $y \in B(f)$ as $y = \sum_{i \in \mathcal{I}} \lambda_i v^i$.
- Suppose that y looks like:

$$y = (\underbrace{- \; - \; - \; - \; - \; -}_{S^-(y)} \; \underbrace{0 \; 0 \; 0 \; 0 \; 0}_{S^0(y)} \; \underbrace{+ \; + \; + \; +}_{S^+(y)})$$

- To maximize $y^-(E)$ ($\iff \min_S y^+(E)$), we want to increase y_e for some $e \in S^-(y)$ (or decrease y_e for some $e \in S^+(y)$).
  - We know that y_e increases if we move e to the left in some $\prec_i$, and y_e decreases if we move e to the right in some $\prec_i$.
  - This suggests we find some $k \in S^-(y)$ and $l \in S^+(y)$ and compute $c(k, l; y)$, then set $y' \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \leq c(k, l; y)$.

Outline of a generic SFMin algorithm

- We keep linear orders $\prec_i$ with associated v^i, and $y \in B(f)$ as $y = \sum_{i \in \mathcal{I}} \lambda_i v^i$.
- Suppose that y looks like:

$$y = (\underbrace{-\ -\ -\ -\ -\ -}_{S^-(y)}\ \underbrace{0\ 0\ 0\ 0\ 0}_{S^0(y)}\ \underbrace{+\ +\ +\ +}_{S^+(y)})$$

- To maximize $y^-(E)$ ($\iff \min_S y^+(E)$), we want to increase y_e for some $e \in S^-(y)$ (or decrease y_e for some $e \in S^+(y)$).
  - We know that y_e increases if we move e to the left in some $\prec_i$, and y_e decreases if we move e to the right in some $\prec_i$.
  - This suggests we find some $k \in S^-(y)$ and $l \in S^+(y)$ and compute $c(k,l;y)$, then set $y' \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \leq c(k,l;y)$.
  - But unfortunately computing $c(k,l;y)$ is as hard as SFMin.

Outline of a generic SFMin algorithm

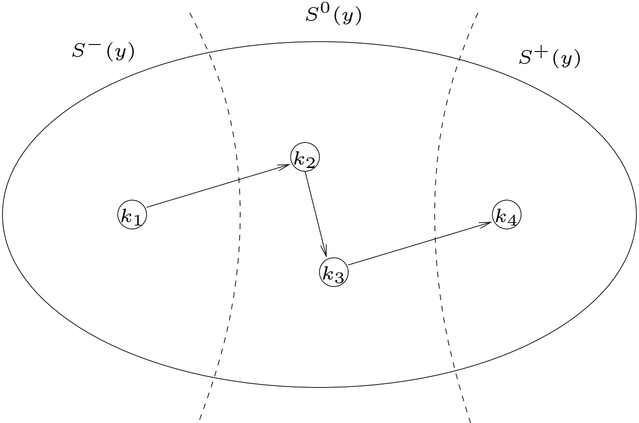
- We keep linear orders $\prec_i$ with associated v^i, and $y \in B(f)$ as $y = \sum_{i \in \mathcal{I}} \lambda_i v^i$.
- Suppose that y looks like:

$$y = (\underbrace{- \; - \; - \; - \; - \; -}_{S^-(y)} \; \underbrace{0\;0\;0\;0\;0}_{S^0(y)} \; \underbrace{+ \; + \; + \; +}_{S^+(y)})$$

- To maximize $y^-(E)$ ($\iff \min_S y^+(E)$), we want to increase y_e for some $e \in S^-(y)$ (or decrease y_e for some $e \in S^+(y)$).
  - We know that y_e increases if we move e to the left in some $\prec_i$, and y_e decreases if we move e to the right in some $\prec_i$.
  - This suggests we find some $k \in S^-(y)$ and $l \in S^+(y)$ and compute $c(k, l; y)$, then set $y' \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \le c(k, l; y)$.
  - But unfortunately computing $c(k, l; y)$ is as hard as SFMin.
  - And if we don't have any $\prec_i$ with (l, k) consecutive in $\prec_i$, then how can we change the representation $y = \sum_{i \in \mathcal{I}} \lambda_i v^i$ to track this $\chi_k - \chi_l$ direction?

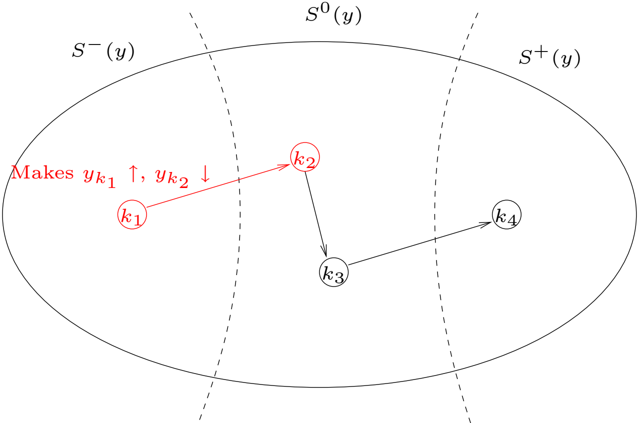
SFMin augmenting paths

Assume that we have the situation as in the picture below, where (k_2, k_1) is consecutive in $\prec_1$, (k_3, k_2) is consecutive in $\prec_2$, and (k_4, k_3) is consecutive in $\prec_3$.



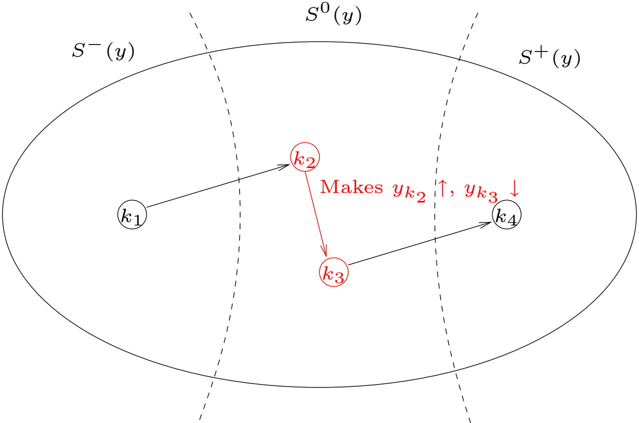
SFMin augmenting paths

Assume that we have the situation as in the picture below, where (k_2, k_1) is consecutive in $\prec_1$, (k_3, k_2) is consecutive in $\prec_2$, and (k_4, k_3) is consecutive in $\prec_3$.
We could swap k_2 and k_1 in $\prec_1$ to $\uparrow y_{k_1}$ and $\downarrow y_{k_2}$, but this wouldn't increase $y^-(E)$.



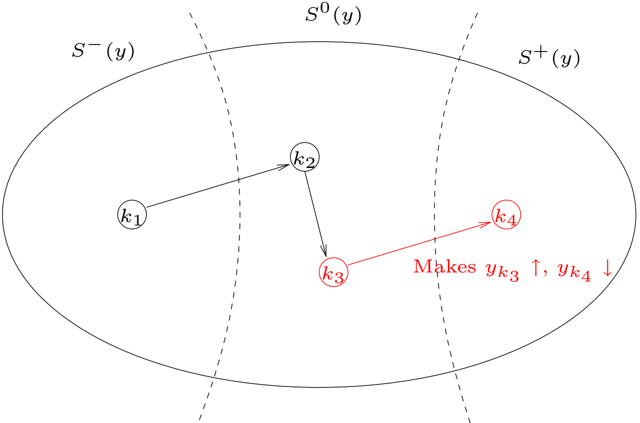
SFMin augmenting paths

Assume that we have the situation as in the picture below, where (k_2, k_1) is consecutive in $\prec_1$, (k_3, k_2) is consecutive in $\prec_2$, and (k_4, k_3) is consecutive in $\prec_3$.
We could swap k_3 and k_2 in $\prec_2$ to $\uparrow y_{k_2}$ and $\downarrow y_{k_3}$, but this wouldn't increase $y^-(E)$.



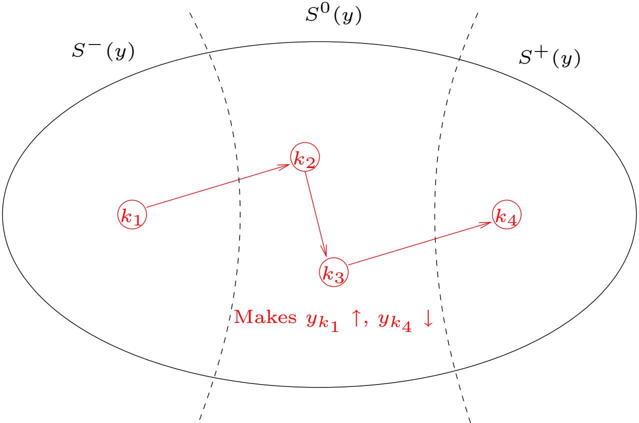
SFMin augmenting paths

Assume that we have the situation as in the picture below, where (k_2, k_1) is consecutive in $\prec_1$, (k_3, k_2) is consecutive in $\prec_2$, and (k_4, k_3) is consecutive in $\prec_3$.
We could swap k_4 and k_3 in $\prec_3$ to $\uparrow y_{k_3}$ and $\downarrow y_{k_4}$, but this wouldn't increase $y^-(E)$.



SFMin augmenting paths

Assume that we have the situation as in the picture below, where (k_2, k_1) is consecutive in $\prec_1$, (k_3, k_2) is consecutive in $\prec_2$, and (k_4, k_3) is consecutive in $\prec_3$.
But if we do all three swaps at the same time this would $\uparrow y_{k_1}$ and $\downarrow y_{k_4}$, and this would increase $y^-(E)$.



SFMin is like Max Flow / Min Cut

- This suggests a rudimentary algorithm:

SFMin is like Max Flow / Min Cut

- This suggests a rudimentary algorithm:
  1. Make a network with nodes E, and arc $e \rightarrow g$ whenever (g, e) is consecutive in some $\prec_i$.

SFMin is like Max Flow / Min Cut

- This suggests a rudimentary algorithm:
  1. Make a network with nodes E, and arc $e \to g$ whenever (g, e) is consecutive in some $\prec_i$.
  2. If there is a directed path from $S^-(y)$ to $S^+(y)$ then augment along it; repeat until no such path remains.

SFMin is like Max Flow / Min Cut

- This suggests a rudimentary algorithm:
  1. Make a network with nodes E, and arc $e \to g$ whenever (g, e) is consecutive in some $\prec_i$.
  2. If there is a directed path from $S^-(y)$ to $S^+(y)$ then augment along it; repeat until no such path remains.
  3. If $\nexists$ a directed path from $S^-(y)$ to $S^+(y)$, define $S^* = \{e \in E \mid \exists$ augmenting path from $S^-(y)$ up to $e\}$. Then we show below that S^* solves SFMin.

SFMin is like Max Flow / Min Cut

- This suggests a rudimentary algorithm:
  1. Make a network with nodes E, and arc $e \to g$ whenever (g, e) is consecutive in some $\prec_i$.
  2. If there is a directed path from $S^-(y)$ to $S^+(y)$ then augment along it; repeat until no such path remains.
  3. If $\nexists$ a directed path from $S^-(y)$ to $S^+(y)$, define $S^* = \{e \in E \mid \exists \text{ augmenting path from } S^-(y) \text{ up to } e\}$. Then we show below that S^* solves SFMin.
- Note that $S^-(y) \subseteq S^* \subseteq E - S^+(y)$, so S^* satisfies two of the three complementary slackness conditions.

SFMin is like Max Flow / Min Cut

- This suggests a rudimentary algorithm:
  1. Make a network with nodes E, and arc $e \to g$ whenever (g, e) is consecutive in some $\prec_i$.
  2. If there is a directed path from $S^-(y)$ to $S^+(y)$ then augment along it; repeat until no such path remains.
  3. If $\nexists$ a directed path from $S^-(y)$ to $S^+(y)$, define $S^* = \{e \in E \mid \exists$ augmenting path from $S^-(y)$ up to $e\}$. Then we show below that S^* solves SFMin.
- Note that $S^-(y) \subseteq S^* \subseteq E - S^+(y)$, so S^* satisfies two of the three complementary slackness conditions.
- I claim that S^* is at the left of every $\prec_i$.

SFMin is like Max Flow / Min Cut

- This suggests a rudimentary algorithm:
  1. Make a network with nodes E, and arc $e \rightarrow g$ whenever (g, e) is consecutive in some $\prec_i$.
  2. If there is a directed path from $S^-(y)$ to $S^+(y)$ then augment along it; repeat until no such path remains.
  3. If $\nexists$ a directed path from $S^-(y)$ to $S^+(y)$, define $S^* = \{e \in E \mid \exists$ augmenting path from $S^-(y)$ up to $e\}$. Then we show below that S^* solves SFMin.
- Note that $S^-(y) \subseteq S^* \subseteq E - S^+(y)$, so S^* satisfies two of the three complementary slackness conditions.
- I claim that S^* is at the left of every $\prec_i$.
  - Suppose that there is some $\prec_i$ with $l \notin S^*$ to the left of some $k \in S^*$.

SFMin is like Max Flow / Min Cut

- This suggests a rudimentary algorithm:
  1. Make a network with nodes E, and arc $e \to g$ whenever (g, e) is consecutive in some $\prec_i$.
  2. If there is a directed path from $S^-(y)$ to $S^+(y)$ then augment along it; repeat until no such path remains.
  3. If $\nexists$ a directed path from $S^-(y)$ to $S^+(y)$, define $S^* = \{e \in E \mid \exists \text{ augmenting path from } S^-(y) \text{ up to } e\}$. Then we show below that S^* solves SFMin.
- Note that $S^-(y) \subseteq S^* \subseteq E - S^+(y)$, so S^* satisfies two of the three complementary slackness conditions.
- I claim that S^* is at the left of every $\prec_i$.
  - Suppose that there is some $\prec_i$ with $l \notin S^*$ to the left of some $k \in S^*$.
  - Then there must be such a pair (l, k) that is consecutive in $\prec_i$.

SFMin is like Max Flow / Min Cut

- This suggests a rudimentary algorithm:
  1. Make a network with nodes E, and arc $e \to g$ whenever (g, e) is consecutive in some $\prec_i$.
  2. If there is a directed path from $S^-(y)$ to $S^+(y)$ then augment along it; repeat until no such path remains.
  3. If $\nexists$ a directed path from $S^-(y)$ to $S^+(y)$, define $S^* = \{e \in E \mid \exists$ augmenting path from $S^-(y)$ up to $e\}$. Then we show below that S^* solves SFMin.
- Note that $S^-(y) \subseteq S^* \subseteq E - S^+(y)$, so S^* satisfies two of the three complementary slackness conditions.
- I claim that S^* is at the left of every $\prec_i$.
  - Suppose that there is some $\prec_i$ with $l \notin S^*$ to the left of some $k \in S^*$.
  - Then there must be such a pair (l, k) that is consecutive in $\prec_i$.
  - But then we could extend the augmenting path to k along arc $k \to l$ coming from consecutive pair (l, k), contradicting that $l \notin S^*$.

SFMin is like Max Flow / Min Cut

- I just showed that S^* is at the left of every $\prec_i$.

SFMin is like Max Flow / Min Cut

- I just showed that S^* is at the left of every $\prec_i$.
- Now S^* at the left of $\prec_i$ implies that $v^i(S^*) = f(S^*)$.

SFMin is like Max Flow / Min Cut

- I just showed that S^* is at the left of every $\prec_i$.
- Now S^* at the left of $\prec_i$ implies that $v^i(S^*) = f(S^*)$.
- Then $y(S^*) = \sum_{i \in \mathcal{I}} \lambda_i v^i(S^*) = \sum_{i \in \mathcal{I}} \lambda_i f(S^*) = f(S^*) \sum_{i \in \mathcal{I}} \lambda_i = f(S^*)$.

SFMin is like Max Flow / Min Cut

- I just showed that S^* is at the left of every $\prec_i$.
- Now S^* at the left of $\prec_i$ implies that $v^i(S^*) = f(S^*)$.
- Then $y(S^*) = \sum_{i \in \mathcal{I}} \lambda_i v^i(S^*) = \sum_{i \in \mathcal{I}} \lambda_i f(S^*) = f(S^*) \sum_{i \in \mathcal{I}} \lambda_i = f(S^*)$.
- Thus S^* is y-tight, the third complementary slackness condition, and so S^* is indeed optimal for SFMin.

SFMin is like Max Flow / Min Cut

- I just showed that S^* is at the left of every $\prec_i$.
- Now S^* at the left of $\prec_i$ implies that $v^i(S^*) = f(S^*)$.
- Then $y(S^*) = \sum_{i \in \mathcal{I}} \lambda_i v^i(S^*) = \sum_{i \in \mathcal{I}} \lambda_i f(S^*) = f(S^*) \sum_{i \in \mathcal{I}} \lambda_i = f(S^*)$.
- Thus S^* is y-tight, the third complementary slackness condition, and so S^* is indeed optimal for SFMin.
- This proof is very much in the same spirit as the Max Flow / Min Cut augmenting path proof.

SFMin is like Max Flow / Min Cut

- I just showed that S^* is at the left of every $\prec_i$.
- Now S^* at the left of $\prec_i$ implies that $v^i(S^*) = f(S^*)$.
- Then $y(S^*) = \sum_{i \in \mathcal{I}} \lambda_i v^i(S^*) = \sum_{i \in \mathcal{I}} \lambda_i f(S^*) = f(S^*) \sum_{i \in \mathcal{I}} \lambda_i = f(S^*)$.
- Thus S^* is y-tight, the third complementary slackness condition, and so S^* is indeed optimal for SFMin.
- This proof is very much in the same spirit as the Max Flow / Min Cut augmenting path proof.
- The same proof works with a more general definition of arcs: Put $e \to g \in A$ whenever $g \prec_i e$ for some $i \in \mathcal{I}$.

SFMin is like Max Flow / Min Cut

- I just showed that S^* is at the left of every $\prec_i$.
- Now S^* at the left of $\prec_i$ implies that $v^i(S^*) = f(S^*)$.
- Then $y(S^*) = \sum_{i \in \mathcal{I}} \lambda_i v^i(S^*) = \sum_{i \in \mathcal{I}} \lambda_i f(S^*) = f(S^*) \sum_{i \in \mathcal{I}} \lambda_i = f(S^*)$.
- Thus S^* is y-tight, the third complementary slackness condition, and so S^* is indeed optimal for SFMin.
- This proof is very much in the same spirit as the Max Flow / Min Cut augmenting path proof.
- The same proof works with a more general definition of arcs: Put $e \to g \in A$ whenever $g \prec_i e$ for some $i \in \mathcal{I}$.
- The "only" remaining thing to do is to find some way to arrange augmentations so there is only a polynomial number of them.

SFMin is *not* like Max Flow / Min Cut

- The set of arcs changes dynamically as $\mathcal{I}$ changes and y changes.

SFMin is *not* like Max Flow / Min Cut

- The set of arcs changes dynamically as $\mathcal{I}$ changes and y changes.
- The "capacity" of arcs changes dynamically.

SFMin is *not* like Max Flow / Min Cut

- The set of arcs changes dynamically as $\mathcal{I}$ changes and y changes.
- The "capacity" of arcs changes dynamically.
- One augmenting path could contain several arcs coming from the same $\prec_i$, implying that computing the augmentation amount is quite complicated.

SFMin is *not* like Max Flow / Min Cut

- The set of arcs changes dynamically as $\mathcal{I}$ changes and y changes.
- The "capacity" of arcs changes dynamically.
- One augmenting path could contain several arcs coming from the same $\prec_i$, implying that computing the augmentation amount is quite complicated.
- Augmentation amounts depend on the λ_i, which can be arbitrarily small.

SFMin is *not* like Max Flow / Min Cut

- The set of arcs changes dynamically as $\mathcal{I}$ changes and y changes.
- The "capacity" of arcs changes dynamically.
- One augmenting path could contain several arcs coming from the same $\prec_i$, implying that computing the augmentation amount is quite complicated.
- Augmentation amounts depend on the λ_i, which can be arbitrarily small.
- These are some of the reasons why it took many, many years to figure out how to get a combinatorial SFMin algorithm, and why Cunningham's SFMin algorithm was only pseudo-polynomial.

Outline

The Fleischer-Iwata Push-Relabel version of Schrijver's Algorithm

- Recall our original naive idea: find some $k \in S^-(y)$ and $l \in S^+(y)$ and update $y \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \leq c(k,l;y)$.

The Fleischer-Iwata Push-Relabel version of Schrijver's Algorithm

- Recall our original naive idea: find some $k \in S^-(y)$ and $l \in S^+(y)$ and update $y \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \leq c(k, l; y)$.
- This seemed like a bad idea for two reasons:

The Fleischer-Iwata Push-Relabel version of Schrijver's Algorithm

- Recall our original naive idea: find some $k \in S^-(y)$ and $l \in S^+(y)$ and update $y \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \leq c(k, l; y)$.
- This seemed like a bad idea for two reasons:
  1. How can we compute a representation of the direction $\chi_k - \chi_l$?

The Fleischer-Iwata Push-Relabel version of Schrijver's Algorithm

- Recall our original naive idea: find some $k \in S^-(y)$ and $l \in S^+(y)$ and update $y \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \leq c(k, l; y)$.
- This seemed like a bad idea for two reasons:
  1. How can we compute a representation of the direction $\chi_k - \chi_l$?
  2. Computing $c(k, l; y)$ is as hard as SFMin.

The Fleischer-Iwata Push-Relabel version of Schrijver's Algorithm

- Recall our original naive idea: find some $k \in S^-(y)$ and $l \in S^+(y)$ and update $y \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \leq c(k, l; y)$.
- This seemed like a bad idea for two reasons:
  1. How can we compute a representation of the direction $\chi_k - \chi_l$?
  2. Computing $c(k, l; y)$ is as hard as SFMin.
- Schrijver figured out a clever way around these difficulties.

The Fleischer-Iwata Push-Relabel version of Schrijver's Algorithm

- Recall our original naive idea: find some $k \in S^-(y)$ and $l \in S^+(y)$ and update $y \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \leq c(k, l; y)$.
- This seemed like a bad idea for two reasons:
  1. How can we compute a representation of the direction $\chi_k - \chi_l$?
  2. Computing $c(k, l; y)$ is as hard as SFMin.
- Schrijver figured out a clever way around these difficulties.
- Recall that $c(k, l; y)$ is easy when (l, k) is consecutive in a linear order defining a vertex.

The Fleischer-Iwata Push-Relabel version of Schrijver's Algorithm

- Recall our original naive idea: find some $k \in S^-(y)$ and $l \in S^+(y)$ and update $y \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \leq c(k, l; y)$.
- This seemed like a bad idea for two reasons:
  1. How can we compute a representation of the direction $\chi_k - \chi_l$?
  2. Computing $c(k, l; y)$ is as hard as SFMin.
- Schrijver figured out a clever way around these difficulties.
- Recall that $c(k, l; y)$ is easy when (l, k) is consecutive in a linear order defining a vertex.
- Define $(l, k]_{\prec} = \{e \in E \mid l \prec e \preceq k\}$.

The Fleischer-Iwata Push-Relabel version of Schrijver's Algorithm

- Recall our original naive idea: find some $k \in S^-(y)$ and $l \in S^+(y)$ and update $y \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \leq c(k, l; y)$.
- This seemed like a bad idea for two reasons:
  1. How can we compute a representation of the direction $\chi_k - \chi_l$?
  2. Computing $c(k, l; y)$ is as hard as SFMin.
- Schrijver figured out a clever way around these difficulties.
- Recall that $c(k, l; y)$ is easy when (l, k) is consecutive in a linear order defining a vertex.
- Define $(l, k]_\prec = \{e \in E \mid l \prec e \preceq k\}$.
- So, intuitively, we can think of $|(l, k]_\prec|$ as being a measure of difficulty of computing $c(k, l; y)$.

The Fleischer-Iwata Push-Relabel version of Schrijver's Algorithm

- Recall our original naive idea: find some $k \in S^-(y)$ and $l \in S^+(y)$ and update $y \leftarrow y + \alpha(\chi_k - \chi_l)$ for some $\alpha \leq c(k, l; y)$.
- This seemed like a bad idea for two reasons:
  1. How can we compute a representation of the direction $\chi_k - \chi_l$?
  2. Computing $c(k, l; y)$ is as hard as SFMin.
- Schrijver figured out a clever way around these difficulties.
- Recall that $c(k, l; y)$ is easy when (l, k) is consecutive in a linear order defining a vertex.
- Define $(l, k]_\prec = \{e \in E \mid l \prec e \preceq k\}$.
- So, intuitively, we can think of $|(l, k]_\prec|$ as being a measure of difficulty of computing $c(k, l; y)$.
  - When (l, k) is consecutive in $\prec$, then $|(l, k]_\prec| = 1$; as $|[l, k]_\prec|$ becomes larger than 1, computing $c(k, l; y)$ becomes more difficult.

Block swaps

- Suppose that we have identified $l \in S^+(y)$ and $k \in S^-(y)$, so we want to $\downarrow y_l$ and $\uparrow y_k$ by moving l to the right and k to the left in some $\prec_h$, call it just $\prec$.

Block swaps

- Suppose that we have identified $l \in S^+(y)$ and $k \in S^-(y)$, so we want to $\downarrow y_l$ and $\uparrow y_k$ by moving l to the right and k to the left in some $\prec_h$, call it just $\prec$.
- Suppose that $\prec$ looks like this for some $j \in (l, k]_\prec$:

$$\cdots s_{a-1} s_a l t_1 t_2 \ldots t_b j u_1 u_2 k w_1 w_2 \cdots ,$$

Block swaps

- Suppose that we have identified $l \in S^+(y)$ and $k \in S^-(y)$, so we want to $\downarrow y_l$ and $\uparrow y_k$ by moving l to the right and k to the left in some $\prec_h$, call it just $\prec$.
- Suppose that $\prec$ looks like this for some $j \in (l, k]_\prec$:

$$\cdots s_{a-1} s_a l t_1 t_2 \ldots t_b j u_1 u_2 k w_1 w_2 \cdots ,$$

- Now define linear order $\prec^{l,j}$ such that j is moved to the left, just ahead of l, so that $\prec^{l,j}$ looks like:

$$\cdots s_{a-1} s_a j l t_1 t_2 \ldots t_b u_1 u_2 k w_1 w_2 \cdots .$$

Block swaps

- Suppose that we have identified $l \in S^+(y)$ and $k \in S^-(y)$, so we want to $\downarrow y_l$ and $\uparrow y_k$ by moving l to the right and k to the left in some $\prec_h$, call it just $\prec$.
- Suppose that $\prec$ looks like this for some $j \in (l, k]_\prec$:

$$\cdots s_{a-1} s_a l t_1 t_2 \ldots t_b j u_1 u_2 k w_1 w_2 \cdots ,$$

- Now define linear order $\prec^{l,j}$ such that j is moved to the left, just ahead of l, so that $\prec^{l,j}$ looks like:

$$\cdots s_{a-1} s_a j l t_1 t_2 \ldots t_b u_1 u_2 k w_1 w_2 \cdots .$$

- Doing this block swap that moves in the direction $v^{l,j} - v^{\prec}$ would indeed $\downarrow y_l$, but it wouldn't affect y_k.

Block swaps

- Suppose that we have identified $l \in S^+(y)$ and $k \in S^-(y)$, so we want to $\downarrow y_l$ and $\uparrow y_k$ by moving l to the right and k to the left in some $\prec_h$, call it just $\prec$.
- Suppose that $\prec$ looks like this for some $j \in (l, k]_\prec$:

$$\cdots s_{a-1} s_a l t_1 t_2 \ldots t_b j u_1 u_2 k w_1 w_2 \cdots ,$$

- Now define linear order $\prec^{l,j}$ such that j is moved to the left, just ahead of l, so that $\prec^{l,j}$ looks like:

$$\cdots s_{a-1} s_a j l t_1 t_2 \ldots t_b u_1 u_2 k w_1 w_2 \cdots .$$

- Doing this block swap that moves in the direction $v^{l,j} - v^\prec$ would indeed $\downarrow y_l$, but it wouldn't affect y_k.
- But $\prec^{l,j}$ does have the nice property that $(l, k]_{\prec^{l,j}} \subset (l, k]_\prec$, so it gets closer to being a $c(k, j; y)$ that we can compute.

The block swap matrix

- Index the elements of $(l,k]_\prec$ as $\color{red}{l} \prec u_1 \prec u_2 \cdots \prec u_q = \color{blue}{k}$ and consider the submatrix of the matrix of columns $v^{l,u_b} - v^\prec$:

$$
\begin{array}{r}
\\
\color{red}{l} \\
u_1 \\
u_2 \\
u_3 \\
\vdots \\
\color{blue}{k} = u_q
\end{array}
\begin{array}{c}
\begin{array}{ccccc} v^{l,u_1} & v^{l,u_2} & v^{l,u_3} & \dots & v^{l,u_q} \end{array} \\
\begin{pmatrix}
\ominus & \ominus & \ominus & \dots & \ominus \\
\oplus & \ominus & \ominus & \dots & \ominus \\
0 & \oplus & \ominus & \dots & \ominus \\
0 & 0 & \oplus & \dots & \ominus \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \dots & \oplus
\end{pmatrix}
\end{array}
$$

The block swap matrix

- Index the elements of $(l, k]_{\prec}$ as $l \prec u_1 \prec u_2 \cdots \prec u_q = k$ and consider the submatrix of the matrix of columns $v^{l,u_b} - v^{\prec}$:

$$
\begin{array}{r}
\\ l \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ k = u_q
\end{array}
\begin{array}{c}
\begin{array}{ccccc} v^{l,u_1} & v^{l,u_2} & v^{l,u_3} & \dots & v^{l,u_q} \end{array} \\
\begin{pmatrix}
\ominus & \ominus & \ominus & \dots & \ominus \\
\oplus & \ominus & \ominus & \dots & \ominus \\
0 & \oplus & \ominus & \dots & \ominus \\
0 & 0 & \oplus & \dots & \ominus \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \dots & \oplus
\end{pmatrix}
\end{array}
$$

- Since v^{l,u_b} differs from $v^{\prec}$ only on elements in $l + (l, k]_{\prec}$, the sum of each column of this submatrix is 0.

The block swap matrix

- Index the elements of $(l,k]_\prec$ as $l \prec u_1 \prec u_2 \cdots \prec u_q = k$ and consider the submatrix of the matrix of columns $v^{l,u_b} - v^\prec$:

$$
\begin{array}{r} \\ l \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ k = u_q \end{array}
\begin{array}{c}
\begin{array}{ccccc} v^{l,u_1} & v^{l,u_2} & v^{l,u_3} & \dots & v^{l,u_q} \end{array} \\
\begin{pmatrix}
\ominus & \ominus & \ominus & \dots & \ominus \\
\oplus & \ominus & \ominus & \dots & \ominus \\
0 & \oplus & \ominus & \dots & \ominus \\
0 & 0 & \oplus & \dots & \ominus \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \dots & \oplus
\end{pmatrix}
\end{array}
$$

- Since v^{l,u_b} differs from $v^\prec$ only on elements in $l + (l,k]_\prec$, the sum of each column of this submatrix is 0.
- Thus the matrix has a redundant row, and we can treat it as a triangular matrix

Setting up the equations to solve

- Our aim is to produce a solution α, μ of the equations

$$v^{\prec} + \alpha(\chi_k - \chi_l) = \sum_{j \in (l,k]_{\prec}} \mu_j v^{l,j} \tag{6}$$

such that $\alpha \geq 0$, $\mu \geq 0$, and $\sum_j \mu_j = 1$.

Setting up the equations to solve

- Our aim is to produce a solution α, μ of the equations

$$v^{\prec} + \alpha(\chi_k - \chi_l) = \sum_{j \in (l,k]_{\prec}} \mu_j v^{l,j} \tag{6}$$

  such that $\alpha \geq 0$, $\mu \geq 0$, and $\sum_j \mu_j = 1$.

- Suppose that for some $j \in (l,k]_{\prec}$ that $v_j^{l,j} = v_j^{\prec}$, i.e., the diagonal entry of the matrix is 0.

Setting up the equations to solve

- Our aim is to produce a solution α, μ of the equations

$$v^{\prec} + \alpha(\chi_k - \chi_l) = \sum_{j \in (l,k]_{\prec}} \mu_j v^{l,j} \tag{6}$$

  such that $\alpha \geq 0$, $\mu \geq 0$, and $\sum_j \mu_j = 1$.
- Suppose that for some $j \in (l,k]_{\prec}$ that $v_j^{l,j} = v_j^{\prec}$, i.e., the diagonal entry of the matrix is 0.
- Then we must have that $v^{l,j} = v^{\prec}$, since if the only possible positive term $v_j^{l,j} - v_j^{\prec} = 0$ and the sum is 0, all possibly negative terms must also be 0.

Setting up the equations to solve

- Our aim is to produce a solution α, μ of the equations

$$v^{\prec} + \alpha(\chi_k - \chi_l) = \sum_{j \in (l,k]_{\prec}} \mu_j v^{l,j} \tag{6}$$

  such that $\alpha \geq 0$, $\mu \geq 0$, and $\sum_j \mu_j = 1$.
- Suppose that for some $j \in (l,k]_{\prec}$ that $v_j^{l,j} = v_j^{\prec}$, i.e., the diagonal entry of the matrix is 0.
- Then we must have that $v^{l,j} = v^{\prec}$, since if the only possible positive term $v_j^{l,j} - v_j^{\prec} = 0$ and the sum is 0, all possibly negative terms must also be 0.
- Thus we can choose $\alpha = 0$ and get this version of (6):

$$v^{\prec} + 0 \cdot (\chi_k - \chi_l) = 1 \cdot v^{l,j}.$$

Setting up the equations to solve

- Our aim is to produce a solution α, μ of the equations

$$v^{\prec} + \alpha(\chi_k - \chi_l) = \sum_{j \in (l,k]_{\prec}} \mu_j v^{l,j} \tag{6}$$

  such that $\alpha \geq 0$, $\mu \geq 0$, and $\sum_j \mu_j = 1$.

- Suppose that for some $j \in (l,k]_{\prec}$ that $v_j^{l,j} = v_j^{\prec}$, i.e., the diagonal entry of the matrix is 0.
- Then we must have that $v^{l,j} = v^{\prec}$, since if the only possible positive term $v_j^{l,j} - v_j^{\prec} = 0$ and the sum is 0, all possibly negative terms must also be 0.
- Thus we can choose $\alpha = 0$ and get this version of (6):

$$v^{\prec} + 0 \cdot (\chi_k - \chi_l) = 1 \cdot v^{l,j}.$$

- Now we are free to assume that all $v_j^{l,j} > v_j^{\prec}$, i.e., the diagonal entries of the matrix are strictly positive.

Synthesizing the direction $\chi_k - \chi_l$

- With all diagonal entries strictly positive, consider the equations in variables η_j

$$
\begin{array}{c}
 \\ l \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ k = u_q
\end{array}
\begin{array}{c}
\begin{array}{ccccc} v^{l,u_1} & v^{l,u_2} & v^{l,u_3} & \dots & v^{l,u_q} \end{array} \\
\begin{pmatrix}
\ominus & \ominus & \ominus & \dots & \ominus \\
+ & \ominus & \ominus & \dots & \ominus \\
0 & + & \ominus & \dots & \ominus \\
0 & 0 & + & \dots & \ominus \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \dots & +
\end{pmatrix}
\end{array}
\begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_q \end{pmatrix}
=
\begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ +1 \end{pmatrix}.
$$

Synthesizing the direction $\chi_k - \chi_l$

- With all diagonal entries strictly positive, consider the equations in variables η_j

$$
\begin{array}{c}
 \\ {\color{red} l} \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ {\color{blue} k} = u_q
\end{array}
\begin{array}{c}
\begin{array}{ccccc} v^{l,u_1} & v^{l,u_2} & v^{l,u_3} & \dots & v^{l,u_q} \end{array} \\
\begin{pmatrix}
\ominus & \ominus & \ominus & \dots & \ominus \\
+ & \ominus & \ominus & \dots & \ominus \\
0 & + & \ominus & \dots & \ominus \\
0 & 0 & + & \dots & \ominus \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \dots & +
\end{pmatrix}
\end{array}
\begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_q \end{pmatrix}
=
\begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ +1 \end{pmatrix}.
$$

- Since this system is triangular with positive diagonal, it has a solution $\eta \geq 0$.

Synthesizing the direction $\chi_k - \chi_l$

- With all diagonal entries strictly positive, consider the equations in variables η_j

$$
\begin{array}{r} \\ \color{red}{l} \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ \color{blue}{k} = u_q \end{array}
\begin{array}{c}
\begin{array}{ccccc} v^{l,u_1} & v^{l,u_2} & v^{l,u_3} & \dots & v^{l,u_q} \end{array} \\
\begin{pmatrix}
\ominus & \ominus & \ominus & \dots & \ominus \\
+ & \ominus & \ominus & \dots & \ominus \\
0 & + & \ominus & \dots & \ominus \\
0 & 0 & + & \dots & \ominus \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \dots & +
\end{pmatrix}
\end{array}
\begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_q \end{pmatrix}
=
\begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ +1 \end{pmatrix}.
$$

- Since this system is triangular with positive diagonal, it has a solution $\eta \geq 0$.
- Put $\alpha = 1/\sum_j \eta_j$ and $\mu = \alpha\eta$. Then $(v^{l,j} - v^{\prec})\eta = \chi_k - \chi_l$ becomes $(v^{l,j} - v^{\prec})\mu = \alpha(\chi_k - \chi_l)$, or $v^{\prec} + \alpha(\chi_k - \chi_l) = \sum_{j \in (l,k]_{\prec}} \mu_j v^{l,j}$ which is (6) as desired.

The ExchBd subroutine

- Computing the matrix take $O(n^2\mathrm{EO})$ time, and solving the triangular system takes $O(n^2)$ time, so computing α and μ is $O(n^2\mathrm{EO})$.

The ExchBd subroutine

- Computing the matrix take $O(n^2\mathrm{EO})$ time, and solving the triangular system takes $O(n^2)$ time, so computing α and μ is $O(n^2\mathrm{EO})$.
- Since $v^{\prec} + \alpha(\chi_k - \chi_l) = \sum_{j \in (l,k]_{\prec}} \mu_j v^{l,j}$ shows that $v^{\prec} + \alpha(\chi_k - \chi_l) \in B(f)$, we must have that $\alpha \leq c(k, l; v^{\prec})$.

The $\textsc{ExchBd}$ subroutine

- Computing the matrix take $O(n^2\text{EO})$ time, and solving the triangular system takes $O(n^2)$ time, so computing α and μ is $O(n^2\text{EO})$.
- Since $v^{\prec} + \alpha(\chi_k - \chi_l) = \sum_{j \in (l,k]_{\prec}} \mu_j v^{l,j}$ shows that $v^{\prec} + \alpha(\chi_k - \chi_l) \in B(f)$, we must have that $\alpha \leq c(k, l; v^{\prec})$.
- Return to using v^h for $v^{\prec}$. If we replace the term $\lambda_h v^h$ in $y = \sum_i \lambda_i v^i$ by $\lambda_h(\sum_{j \in (l,k]_{\prec}} \mu_j v^{l,j})$, then y will change by $\lambda_h \alpha(\chi_k - \chi_l)$, and so we'll move in the direction we want.

The ExchBd subroutine

- Computing the matrix take $O(n^2\mathrm{EO})$ time, and solving the triangular system takes $O(n^2)$ time, so computing α and μ is $O(n^2\mathrm{EO})$.
- Since $v^{\prec} + \alpha(\chi_k - \chi_l) = \sum_{j\in(l,k]_{\prec}} \mu_j v^{l,j}$ shows that $v^{\prec} + \alpha(\chi_k - \chi_l) \in B(f)$, we must have that $\alpha \leq c(k,l;v^{\prec})$.
- Return to using v^h for $v^{\prec}$. If we replace the term $\lambda_h v^h$ in $y = \sum_i \lambda_i v^i$ by $\lambda_h(\sum_{j\in(l,k]_{\prec}} \mu_j v^{l,j})$, then y will change by $\lambda_h \alpha(\chi_k - \chi_l)$, and so we'll move in the direction we want.
- We could also take a partial step where we choose some β with $0 \leq \beta \leq \alpha\lambda_h$ and replace $\lambda_h v^h$ by $(\lambda_h - \beta/\alpha)v^h + \beta\sum_{j\in(l,k]_{\prec}} \mu_j v^{l,j}$, and then y would change by $\beta(\chi_k - \chi_l)$.

The ExchBd subroutine

- Computing the matrix take $O(n^2\mathrm{EO})$ time, and solving the triangular system takes $O(n^2)$ time, so computing α and μ is $O(n^2\mathrm{EO})$.
- Since $v^{\prec} + \alpha(\chi_k - \chi_l) = \sum_{j\in(l,k]_{\prec}} \mu_j v^{l,j}$ shows that $v^{\prec} + \alpha(\chi_k - \chi_l) \in B(f)$, we must have that $\alpha \leq c(k,l;v^{\prec})$.
- Return to using v^h for $v^{\prec}$. If we replace the term $\lambda_h v^h$ in $y = \sum_i \lambda_i v^i$ by $\lambda_h(\sum_{j\in(l,k]_{\prec}} \mu_j v^{l,j})$, then y will change by $\lambda_h\alpha(\chi_k - \chi_l)$, and so we'll move in the direction we want.
- We could also take a partial step where we choose some β with $0 \leq \beta \leq \alpha\lambda_h$ and replace $\lambda_h v^h$ by $(\lambda_h - \beta/\alpha)v^h + \beta\sum_{j\in(l,k]_{\prec}} \mu_j v^{l,j}$, and then y would change by $\beta(\chi_k - \chi_l)$.
- We call this operation ExchBd, since it gives us the bound α on $c(k,l;v^{\prec})$.

Distance labels

- Now we use distance labels as in the Goldberg-Tarjan Push-Relabel Algorithm for Max Flow / Min Cut.

Distance labels

- Now we use distance labels as in the Goldberg-Tarjan Push-Relabel Algorithm for Max Flow / Min Cut.
- Distance labels $d \in \mathbb{Z}^E$ are valid if

Distance labels

- Now we use distance labels as in the Goldberg-Tarjan Push-Relabel Algorithm for Max Flow / Min Cut.
- Distance labels $d \in \mathbb{Z}^E$ are valid if
  1. $d_e = 0$ for all $e \in S^-(y)$.

Distance labels

- Now we use distance labels as in the Goldberg-Tarjan Push-Relabel Algorithm for Max Flow / Min Cut.
- Distance labels $d \in \mathbb{Z}^E$ are valid if
  1. $d_e = 0$ for all $e \in S^-(y)$.
  2. $d_l \leq d_k + 1$ if there is some $i \in \mathcal{I}$ with $l \prec_i k$.

Distance labels

- Now we use distance labels as in the Goldberg-Tarjan Push-Relabel Algorithm for Max Flow / Min Cut.
- Distance labels $d \in \mathbb{Z}^E$ are valid if
  1. $d_e = 0$ for all $e \in S^-(y)$.
  2. $d_l \leq d_k + 1$ if there is some $i \in \mathcal{I}$ with $l \prec_i k$.
- Thus we are effectively looking at a directed graph on nodes E, where arc $k \to l \in A$ exists iff there is some $i \in \mathcal{I}$ with $l \prec_i k$, and d_e is a lower bound on the number of arcs in a path from an element of $S^-(y)$ to e.

Distance labels

- Now we use distance labels as in the Goldberg-Tarjan Push-Relabel Algorithm for Max Flow / Min Cut.
- Distance labels $d \in \mathbb{Z}^E$ are valid if
  1. $d_e = 0$ for all $e \in S^-(y)$.
  2. $d_l \leq d_k + 1$ if there is some $i \in \mathcal{I}$ with $l \prec_i k$.
- Thus we are effectively looking at a directed graph on nodes E, where arc $k \to l \in A$ exists iff there is some $i \in \mathcal{I}$ with $l \prec_i k$, and d_e is a lower bound on the number of arcs in a path from an element of $S^-(y)$ to e.
  - When we call $\textsc{ReduceV}$ it will delete some i from $\mathcal{I}$ and so some arcs from A, but this won't violate validity.

Distance labels

- Now we use distance labels as in the Goldberg-Tarjan Push-Relabel Algorithm for Max Flow / Min Cut.
- Distance labels $d \in \mathbb{Z}^E$ are valid if
  1. $d_e = 0$ for all $e \in S^-(y)$.
  2. $d_l \leq d_k + 1$ if there is some $i \in \mathcal{I}$ with $l \prec_i k$.
- Thus we are effectively looking at a directed graph on nodes E, where arc $k \rightarrow l \in A$ exists iff there is some $i \in \mathcal{I}$ with $l \prec_i k$, and d_e is a lower bound on the number of arcs in a path from an element of $S^-(y)$ to e.
  - When we call ReduceV it will delete some i from $\mathcal{I}$ and so some arcs from A, but this won't violate validity.
- As usual, it's easy to show that if $d_e = n$, then e is never going to be active again in the algorithm, so each $d_e \leq n$.

Distance labels

- Now we use distance labels as in the Goldberg-Tarjan Push-Relabel Algorithm for Max Flow / Min Cut.
- Distance labels $d \in \mathbb{Z}^E$ are valid if
  1. $d_e = 0$ for all $e \in S^-(y)$.
  2. $d_l \leq d_k + 1$ if there is some $i \in \mathcal{I}$ with $l \prec_i k$.
- Thus we are effectively looking at a directed graph on nodes E, where arc $k \to l \in A$ exists iff there is some $i \in \mathcal{I}$ with $l \prec_i k$, and d_e is a lower bound on the number of arcs in a path from an element of $S^-(y)$ to e.
  - When we call ReduceV it will delete some i from $\mathcal{I}$ and so some arcs from A, but this won't violate validity.
- As usual, it's easy to show that if $d_e = n$, then e is never going to be active again in the algorithm, so each $d_e \leq n$.
- The d_e are monotonically non-decreasing during the algorithm, and only subroutine Relabel increases a d_e, so there are $O(n^2)$ Relabels.

Distance labels

- Now we use distance labels as in the Goldberg-Tarjan Push-Relabel Algorithm for Max Flow / Min Cut.
- Distance labels $d \in \mathbb{Z}^E$ are valid if
  1. $d_e = 0$ for all $e \in S^-(y)$.
  2. $d_l \leq d_k + 1$ if there is some $i \in \mathcal{I}$ with $l \prec_i k$.
- Thus we are effectively looking at a directed graph on nodes E, where arc $k \to l \in A$ exists iff there is some $i \in \mathcal{I}$ with $l \prec_i k$, and d_e is a lower bound on the number of arcs in a path from an element of $S^-(y)$ to e.
  - When we call ReduceV it will delete some i from $\mathcal{I}$ and so some arcs from A, but this won't violate validity.
- As usual, it's easy to show that if $d_e = n$, then e is never going to be active again in the algorithm, so each $d_e \leq n$.
- The d_e are monotonically non-decreasing during the algorithm, and only subroutine Relabel increases a d_e, so there are $O(n^2)$ Relabels.
- We can initialize with $d \equiv 0$, which is valid.

PUSHing from active nodes

- Call a node l active if $l \in S^+(y)$ and $d_l < n$.

PUSHing from active nodes

- Call a node l active if $l \in S^+(y)$ and $d_l < n$.
- Suppose that $l \prec_h k$ (and so $k \to l \in A$) and $d_k = d_l - 1$ (we are trying to push the positive part of y_l to a lower distance label).

PUSHing from active nodes

- Call a node l active if $l \in S^+(y)$ and $d_l < n$.
- Suppose that $l \prec_h k$ (and so $k \to l \in A$) and $d_k = d_l - 1$ (we are trying to push the positive part of y_l to a lower distance label).
- If no such k exists (due to all k with $k \to l \in A$ having $d_k \geq d_l$), then we can RELABEL: $d_l \leftarrow d_l + 1$; this does not violate validity.

PUSHing from active nodes

- Call a node l active if $l \in S^+(y)$ and $d_l < n$.
- Suppose that $l \prec_h k$ (and so $k \to l \in A$) and $d_k = d_l - 1$ (we are trying to push the positive part of y_l to a lower distance label).
- If no such k exists (due to all k with $k \to l \in A$ having $d_k \geq d_l$), then we can RELABEL: $d_l \leftarrow d_l + 1$; this does not violate validity.
- Then for $\beta \leq \alpha\lambda_h$ call EXCHBD to do $y \leftarrow y + \beta(\chi_k - \chi_l)$.

PUSHing from active nodes

- Call a node l active if $l \in S^+(y)$ and $d_l < n$.
- Suppose that $l \prec_h k$ (and so $k \to l \in A$) and $d_k = d_l - 1$ (we are trying to push the positive part of y_l to a lower distance label).
- If no such k exists (due to all k with $k \to l \in A$ having $d_k \geq d_l$), then we can RELABEL: $d_l \leftarrow d_l + 1$; this does not violate validity.
- Then for $\beta \leq \alpha\lambda_h$ call EXCHBD to do $y \leftarrow y + \beta(\chi_k - \chi_l)$.
- To keep validity we can't allow y_l to become negative, so we must choose $\beta \leq y_l$; thus we choose $\beta = \min(\alpha\lambda_h, y_l)$.

PUSHing from active nodes

- Call a node l active if $l \in S^+(y)$ and $d_l < n$.
- Suppose that $l \prec_h k$ (and so $k \to l \in A$) and $d_k = d_l - 1$ (we are trying to push the positive part of y_l to a lower distance label).
- If no such k exists (due to all k with $k \to l \in A$ having $d_k \geq d_l$), then we can RELABEL: $d_l \leftarrow d_l + 1$; this does not violate validity.
- Then for $\beta \leq \alpha\lambda_h$ call EXCHBD to do $y \leftarrow y + \beta(\chi_k - \chi_l)$.
- To keep validity we can't allow y_l to become negative, so we must choose $\beta \leq y_l$; thus we choose $\beta = \min(\alpha\lambda_h, y_l)$.
- If $\beta = \alpha\lambda_h$ then we call this a saturating step; if $\beta = y_l$ we call it a non-saturating step.

PUSHing from active nodes

- Call a node l active if $l \in S^+(y)$ and $d_l < n$.
- Suppose that $l \prec_h k$ (and so $k \to l \in A$) and $d_k = d_l - 1$ (we are trying to push the positive part of y_l to a lower distance label).
- If no such k exists (due to all k with $k \to l \in A$ having $d_k \geq d_l$), then we can RELABEL: $d_l \leftarrow d_l + 1$; this does not violate validity.
- Then for $\beta \leq \alpha\lambda_h$ call EXCHBD to do $y \leftarrow y + \beta(\chi_k - \chi_l)$.
- To keep validity we can't allow y_l to become negative, so we must choose $\beta \leq y_l$; thus we choose $\beta = \min(\alpha\lambda_h, y_l)$.
- If $\beta = \alpha\lambda_h$ then we call this a saturating step; if $\beta = y_l$ we call it a non-saturating step.
  - In a saturating step h drops out of $\mathcal{I}$ and so one source of the arc $k \to l$ disappears, which is not so easy to analyze.

PUSHing from active nodes

- Call a node l active if $l \in S^+(y)$ and $d_l < n$.
- Suppose that $l \prec_h k$ (and so $k \to l \in A$) and $d_k = d_l - 1$ (we are trying to push the positive part of y_l to a lower distance label).
- If no such k exists (due to all k with $k \to l \in A$ having $d_k \geq d_l$), then we can RELABEL: $d_l \leftarrow d_l + 1$; this does not violate validity.
- Then for $\beta \leq \alpha\lambda_h$ call EXCHBD to do $y \leftarrow y + \beta(\chi_k - \chi_l)$.
- To keep validity we can't allow y_l to become negative, so we must choose $\beta \leq y_l$; thus we choose $\beta = \min(\alpha\lambda_h, y_l)$.
- If $\beta = \alpha\lambda_h$ then we call this a saturating step; if $\beta = y_l$ we call it a non-saturating step.
  - In a saturating step h drops out of $\mathcal{I}$ and so one source of the arc $k \to l$ disappears, which is not so easy to analyze.
  - In a saturating step y_l drops to zero, which is easier to analyze.

How to choose l, k, and h?

- We choose some active l with maximum d_l.

How to choose l, k, and h?

- We choose some active l with maximum d_l.
- Then we scan through the possible k in a fixed order, and when we find k with $d_k = d_l - 1$ and $k \to l \in A$ we call $\textsc{Push}(l, k)$:

How to choose l, k, and h?

- We choose some active l with maximum d_l.
- Then we scan through the possible k in a fixed order, and when we find k with $d_k = d_l - 1$ and $k \to l \in A$ we call $\textsc{Push}(l, k)$:
  1. Now $k \to l \in A$ because $l \prec_h k$ for some $h \in \mathcal{I}$. Among the choices for h choose one that maximizes $|(l, k]_{\prec_h}|$.

How to choose l, k, and h?

- We choose some active l with maximum d_l.
- Then we scan through the possible k in a fixed order, and when we find k with $d_k = d_l - 1$ and $k \to l \in A$ we call $\textsc{Push}(l, k)$:
  1. Now $k \to l \in A$ because $l \prec_h k$ for some $h \in \mathcal{I}$. Among the choices for h choose one that maximizes $|(l, k]_{\prec_h}|$.
  2. Call $\textsc{ExchBd}$; If the step was non-saturating ($y_l = 0$) exit; else call $\textsc{ReduceV}$, go to 1.

How to choose l, k, and h?

- We choose some active l with maximum d_l.
- Then we scan through the possible k in a fixed order, and when we find k with $d_k = d_l - 1$ and $k \to l \in A$ we call $\textsc{Push}(l, k)$:
  1. Now $k \to l \in A$ because $l \prec_h k$ for some $h \in \mathcal{I}$. Among the choices for h choose one that maximizes $|(l, k]_{\prec_h}|$.
  2. Call $\textsc{ExchBd}$; If the step was non-saturating $(y_l = 0)$ exit; else call $\textsc{ReduceV}$, go to 1.
- I claim that these choices imply that there are $O(n^2)$ calls of $\textsc{ExchBd}$ in $\textsc{Push}(l, k)$.

How to choose l, k, and h?

- We choose some active l with maximum d_l.
- Then we scan through the possible k in a fixed order, and when we find k with $d_k = d_l - 1$ and $k \to l \in A$ we call $\textsc{Push}(l, k)$:
  1. Now $k \to l \in A$ because $l \prec_h k$ for some $h \in \mathcal{I}$. Among the choices for h choose one that maximizes $|(l, k]_{\prec_h}|$.
  2. Call $\textsc{ExchBd}$; If the step was non-saturating $(y_l = 0)$ exit; else call $\textsc{ReduceV}$, go to 1.
- I claim that these choices imply that there are $O(n^2)$ calls of $\textsc{ExchBd}$ in $\textsc{Push}(l, k)$.
  - Non-saturating $\textsc{Push}$es exit, so worry only about saturating $\textsc{Push}$es.

How to choose l, k, and h?

- We choose some active l with maximum d_l.
- Then we scan through the possible k in a fixed order, and when we find k with $d_k = d_l - 1$ and $k \to l \in A$ we call $\textsc{Push}(l, k)$:
  1. Now $k \to l \in A$ because $l \prec_h k$ for some $h \in \mathcal{I}$. Among the choices for h choose one that maximizes $|(l, k]_{\prec_h}|$.
  2. Call $\textsc{ExchBd}$; If the step was non-saturating $(y_l = 0)$ exit; else call $\textsc{ReduceV}$, go to 1.
- I claim that these choices imply that there are $O(n^2)$ calls of $\textsc{ExchBd}$ in $\textsc{Push}(l, k)$.
  - Non-saturating $\textsc{Push}$es exit, so worry only about saturating $\textsc{Push}$es.
  - Each saturating $\textsc{Push}(l, k)$ either reduces $\max_i |(l, k]_{\prec_i}|$ or the number of i achieving this max. Since $|(l, k]_{\prec_i}| \le n$, there are $O(n^2)$ iterations.

The overall algorithm

1. Initialize $\prec_1$ as any linear order, $\mathcal{I} = \{1\}$, $y = v^1$, $d \equiv 0$.

The overall algorithm

1. Initialize $\prec_1$ as any linear order, $\mathcal{I} = \{1\}$, $y = v^1$, $d \equiv 0$.
2. While $S^+(y) \neq \emptyset$ and $S^-(y) \neq \emptyset$ do

The overall algorithm

1. Initialize $\prec_1$ as any linear order, $\mathcal{I} = \{1\}$, $y = v^1$, $d \equiv 0$.
2. While $S^+(y) \neq \emptyset$ and $S^-(y) \neq \emptyset$ do
   2.1 Choose active l that maximizes d_l.

The overall algorithm

1. Initialize $\prec_1$ as any linear order, $\mathcal{I} = \{1\}$, $y = v^1$, $d \equiv 0$.
2. While $S^+(y) \neq \emptyset$ and $S^-(y) \neq \emptyset$ do
   2.1 Choose active l that maximizes d_l.
   2.2 Scan through potential k's with $d_k = d_l - 1$:

The overall algorithm

1. Initialize $\prec_1$ as any linear order, $\mathcal{I} = \{1\}$, $y = v^1$, $d \equiv 0$.
2. While $S^+(y) \neq \emptyset$ and $S^-(y) \neq \emptyset$ do
   2.1 Choose active l that maximizes d_l.
   2.2 Scan through potential k's with $d_k = d_l - 1$:
      2.2.1 If find a k, call $\textsc{Push}(l, k)$.

The overall algorithm

1. Initialize $\prec_1$ as any linear order, $\mathcal{I} = \{1\}$, $y = v^1$, $d \equiv 0$.
2. While $S^+(y) \neq \emptyset$ and $S^-(y) \neq \emptyset$ do
   - 2.1 Choose active l that maximizes d_l.
   - 2.2 Scan through potential k's with $d_k = d_l - 1$:
     - 2.2.1 If find a k, call $\textsc{Push}(l, k)$.
     - 2.2.2 If no such k, $\textsc{Relabel}(l)$.

The overall algorithm

1. Initialize $\prec_1$ as any linear order, $\mathcal{I} = \{1\}$, $y = v^1$, $d \equiv 0$.
2. While $S^+(y) \neq \emptyset$ and $S^-(y) \neq \emptyset$ do
   2.1 Choose active l that maximizes d_l.
   2.2 Scan through potential k's with $d_k = d_l - 1$:
      2.2.1 If find a k, call $\textsc{Push}(l, k)$.
      2.2.2 If no such k, $\textsc{Relabel}(l)$.
   2.3 End scan.

The overall algorithm

1. Initialize $\prec_1$ as any linear order, $\mathcal{I} = \{1\}$, $y = v^1$, $d \equiv 0$.
2. While $S^+(y) \neq \emptyset$ and $S^-(y) \neq \emptyset$ do
   - 2.1 Choose active l that maximizes d_l.
   - 2.2 Scan through potential k's with $d_k = d_l - 1$:
     - 2.2.1 If find a k, call $\textsc{Push}(l, k)$.
     - 2.2.2 If no such k, $\textsc{Relabel}(l)$.
   - 2.3 End scan.
3. End while.

The overall algorithm

1. Initialize $\prec_1$ as any linear order, $\mathcal{I} = \{1\}$, $y = v^1$, $d \equiv 0$.
2. While $S^+(y) \neq \emptyset$ and $S^-(y) \neq \emptyset$ do
   - 2.1 Choose active l that maximizes d_l.
   - 2.2 Scan through potential k's with $d_k = d_l - 1$:
     - 2.2.1 If find a k, call $\textsc{Push}(l, k)$.
     - 2.2.2 If no such k, $\textsc{Relabel}(l)$.
   - 2.3 End scan.
3. End while.
4. Compute $S = \{e \mid e \text{ is reachable from } S_1(y)\}$ and return S as an optimal SFMin solution.

Outline

Validity of distance labels is preserved

- Each call to $\textsc{ExchBd}$ during $\textsc{Push}(l,k)$ adds all the $v^{l,j}$ to $\mathcal{I}$.

Validity of distance labels is preserved

- Each call to $\textsc{ExchBd}$ during $\textsc{Push}(l, k)$ adds all the $v^{l,j}$ to $\mathcal{I}$.
- Suppose that $\prec_h$ had $l \prec_h u \prec_h t \prec_h k$ and we consider $\prec^{l,t}$.

Validity of distance labels is preserved

- Each call to $\textsc{ExchBd}$ during $\textsc{Push}(l, k)$ adds all the $v^{l,j}$ to $\mathcal{I}$.
- Suppose that $\prec_h$ had $l \prec_h u \prec_h t \prec_h k$ and we consider $\prec^{l,t}$.
- Then $\prec' \equiv \prec^{l,t}$ has $t \prec' l \prec' u \prec' k$.

Validity of distance labels is preserved

- Each call to $\textsc{ExchBd}$ during $\textsc{Push}(l, k)$ adds all the $v^{l,j}$ to $\mathcal{I}$.
- Suppose that $\prec_h$ had $l \prec_h u \prec_h t \prec_h k$ and we consider $\prec^{l,t}$.
- Then $\prec' \equiv \prec^{l,t}$ has $t \prec' l \prec' u \prec' k$.
- This creates new arc $u \to t$, and we need to check that $d_t \leq d_u + 1$.

Validity of distance labels is preserved

- Each call to $\textsc{ExchBd}$ during $\textsc{Push}(l, k)$ adds all the $v^{l,j}$ to $\mathcal{I}$.
- Suppose that $\prec_h$ had $l \prec_h u \prec_h t \prec_h k$ and we consider $\prec^{l,t}$.
- Then $\prec' \equiv \prec^{l,t}$ has $t \prec' l \prec' u \prec' k$.
- This creates new arc $u \to t$, and we need to check that $d_t \leq d_u + 1$.
- Validity on $u \to l$ of $\prec_h$ implies that $d_l \leq d_u + 1$.

Validity of distance labels is preserved

- Each call to $\textsc{ExchBd}$ during $\textsc{Push}(l,k)$ adds all the $v^{l,j}$ to $\mathcal{I}$.
- Suppose that $\prec_h$ had $l \prec_h u \prec_h t \prec_h k$ and we consider $\prec^{l,t}$.
- Then $\prec' \equiv \prec^{l,t}$ has $t \prec' l \prec' u \prec' k$.
- This creates new arc $u \to t$, and we need to check that $d_t \leq d_u + 1$.
- Validity on $u \to l$ of $\prec_h$ implies that $d_l \leq d_u + 1$.
- Validity on $k \to t$ of $\prec_h$ implies that $d_t \leq d_k + 1$.

Validity of distance labels is preserved

- Each call to $\textsc{ExchBd}$ during $\textsc{Push}(l,k)$ adds all the $v^{l,j}$ to $\mathcal{I}$.
- Suppose that $\prec_h$ had $l \prec_h u \prec_h t \prec_h k$ and we consider $\prec^{l,t}$.
- Then $\prec' \equiv \prec^{l,t}$ has $t \prec' l \prec' u \prec' k$.
- This creates new arc $u \to t$, and we need to check that $d_t \leq d_u + 1$.
- Validity on $u \to l$ of $\prec_h$ implies that $d_l \leq d_u + 1$.
- Validity on $k \to t$ of $\prec_h$ implies that $d_t \leq d_k + 1$.
- Doing $\textsc{Push}(l,k)$ implies that $d_l = d_k + 1$.

Validity of distance labels is preserved

- Each call to $\textsc{ExchBd}$ during $\textsc{Push}(l,k)$ adds all the $v^{l,j}$ to $\mathcal{I}$.
- Suppose that $\prec_h$ had $l \prec_h u \prec_h t \prec_h k$ and we consider $\prec^{l,t}$.
- Then $\prec' \equiv \prec^{l,t}$ has $t \prec' l \prec' u \prec' k$.
- This creates new arc $u \to t$, and we need to check that $d_t \leq d_u + 1$.
- Validity on $u \to l$ of $\prec_h$ implies that $d_l \leq d_u + 1$.
- Validity on $k \to t$ of $\prec_h$ implies that $d_t \leq d_k + 1$.
- Doing $\textsc{Push}(l,k)$ implies that $d_l = d_k + 1$.
- Thus $d_t \leq d_k + 1 = d_l \leq d_u + 1$, which is valid.

Validity of distance labels is preserved

- Each call to $\textsc{ExchBd}$ during $\textsc{Push}(l, k)$ adds all the $v^{l,j}$ to $\mathcal{I}$.
- Suppose that $\prec_h$ had $l \prec_h u \prec_h t \prec_h k$ and we consider $\prec^{l,t}$.
- Then $\prec' \equiv \prec^{l,t}$ has $t \prec' l \prec' u \prec' k$.
- This creates new arc $u \to t$, and we need to check that $d_t \leq d_u + 1$.
- Validity on $u \to l$ of $\prec_h$ implies that $d_l \leq d_u + 1$.
- Validity on $k \to t$ of $\prec_h$ implies that $d_t \leq d_k + 1$.
- Doing $\textsc{Push}(l, k)$ implies that $d_l = d_k + 1$.
- Thus $d_t \leq d_k + 1 = d_l \leq d_u + 1$, which is valid.
- We already noted that other operations preserve validity.

Number of PUSHes

- I claim that there are $O(n^3)$ non-saturating PUSHes:

Number of PUSHes

- I claim that there are $O(n^3)$ non-saturating PUSHes:
  - A non-saturating $\text{PUSH}(l, k)$ makes y_l drop to 0.

Number of PUSHes

- I claim that there are $O(n^3)$ non-saturating PUSHes:
  - A non-saturating $\text{PUSH}(l, k)$ makes y_l drop to 0.
  - So there needs to be a $\text{PUSH}(u, l)$ to make $y_l > 0$ before the next non-saturating PUSH at l.

Number of $\textsc{Push}$es

- I claim that there are $O(n^3)$ non-saturating $\textsc{Push}$es:
  - A non-saturating $\textsc{Push}(l, k)$ makes y_l drop to 0.
  - So there needs to be a $\textsc{Push}(u, l)$ to make $y_l > 0$ before the next non-saturating $\textsc{Push}$ at l.
  - Since we $\textsc{Push}$ from a max distance node, d_u at the $\textsc{Push}(u, l)$ must be greater than d_l at the $\textsc{Push}(l, k)$, which is at least d_u at the $\textsc{Push}(l, k)$.

Number of PUSHes

- I claim that there are $O(n^3)$ non-saturating PUSHes:
  - A non-saturating $\text{PUSH}(l, k)$ makes y_l drop to 0.
  - So there needs to be a $\text{PUSH}(u, l)$ to make $y_l > 0$ before the next non-saturating PUSH at l.
  - Since we PUSH from a max distance node, d_u at the $\text{PUSH}(u, l)$ must be greater than d_l at the $\text{PUSH}(l, k)$, which is at least d_u at the $\text{PUSH}(l, k)$.
  - Thus there was a $\text{RELABEL}(u)$ between the PUSHes.

Number of PUSHes

- I claim that there are $O(n^3)$ non-saturating PUSHes:
  - A non-saturating $\text{PUSH}(l, k)$ makes y_l drop to 0.
  - So there needs to be a $\text{PUSH}(u, l)$ to make $y_l > 0$ before the next non-saturating PUSH at l.
  - Since we PUSH from a max distance node, d_u at the $\text{PUSH}(u, l)$ must be greater than d_l at the $\text{PUSH}(l, k)$, which is at least d_u at the $\text{PUSH}(l, k)$.
  - Thus there was a $\text{RELABEL}(u)$ between the PUSHes.
  - Each such $\text{RELABEL}(u)$ can re-activate at most n such l's, and there are $O(n^2)$ RELABELs, and so $O(n^3)$ non-saturating PUSHes.

Number of PUSHes

- I claim that there are $O(n^3)$ non-saturating PUSHes:
  - A non-saturating PUSH(l, k) makes y_l drop to 0.
  - So there needs to be a PUSH(u, l) to make $y_l > 0$ before the next non-saturating PUSH at l.
  - Since we PUSH from a max distance node, d_u at the PUSH(u, l) must be greater than d_l at the PUSH(l, k), which is at least d_u at the PUSH(l, k).
  - Thus there was a RELABEL(u) between the PUSHes.
  - Each such RELABEL(u) can re-activate at most n such l's, and there are $O(n^2)$ RELABELs, and so $O(n^3)$ non-saturating PUSHes.
- I claim that there are $O(n^3)$ saturating PUSHes:

Number of PUSHes

- I claim that there are $O(n^3)$ non-saturating PUSHes:
  - A non-saturating PUSH(l, k) makes y_l drop to 0.
  - So there needs to be a PUSH(u, l) to make $y_l > 0$ before the next non-saturating PUSH at l.
  - Since we PUSH from a max distance node, d_u at the PUSH(u, l) must be greater than d_l at the PUSH(l, k), which is at least d_u at the PUSH(l, k).
  - Thus there was a RELABEL(u) between the PUSHes.
  - Each such RELABEL(u) can re-activate at most n such l's, and there are $O(n^2)$ RELABELs, and so $O(n^3)$ non-saturating PUSHes.
- I claim that there are $O(n^3)$ saturating PUSHes:
  - Due to scanning, there are at most n saturating PUSHes before each RELABEL(l).

Number of $\textsc{Push}$es

- I claim that there are $O(n^3)$ non-saturating $\textsc{Push}$es:
  - A non-saturating $\textsc{Push}(l, k)$ makes y_l drop to 0.
  - So there needs to be a $\textsc{Push}(u, l)$ to make $y_l > 0$ before the next non-saturating $\textsc{Push}$ at l.
  - Since we $\textsc{Push}$ from a max distance node, d_u at the $\textsc{Push}(u, l)$ must be greater than d_l at the $\textsc{Push}(l, k)$, which is at least d_u at the $\textsc{Push}(l, k)$.
  - Thus there was a $\textsc{Relabel}(u)$ between the $\textsc{Push}$es.
  - Each such $\textsc{Relabel}(u)$ can re-activate at most n such l's, and there are $O(n^2)$ $\textsc{Relabel}$s, and so $O(n^3)$ non-saturating $\textsc{Push}$es.
- I claim that there are $O(n^3)$ saturating $\textsc{Push}$es:
  - Due to scanning, there are at most n saturating $\textsc{Push}$es before each $\textsc{Relabel}(l)$.
  - There are at most n $\textsc{Relabel}(l)$'s, and so $O(n^2)$ saturating $\textsc{Push}$es from l, and so $O(n^3)$ total saturating $\textsc{Push}$es.

Running time analysis of Push-Relabel version of Schrijver

- The algorithm produces the optimal SFMin solution via the "Max Flow / Min Cut" Lemma

Running time analysis of Push-Relabel version of Schrijver

- The algorithm produces the optimal SFMin solution via the "Max Flow / Min Cut" Lemma
- I claim that the overall running time is $O(n^7\mathrm{EO} + n^8)$:

Running time analysis of Push-Relabel version of Schrijver

- The algorithm produces the optimal SFMin solution via the "Max Flow / Min Cut" Lemma
- I claim that the overall running time is $O(n^7 \mathrm{EO} + n^8)$:
  - There are $O(n^3)$ total PUSHes.

Running time analysis of Push-Relabel version of Schrijver

- The algorithm produces the optimal SFMin solution via the "Max Flow / Min Cut" Lemma
- I claim that the overall running time is $O(n^7 \mathrm{EO} + n^8)$:
  - There are $O(n^3)$ total $\textsc{Push}$es.
  - Each $\textsc{Push}$ calls $\textsc{ExchBd}$ and $\textsc{ReduceV}$ $O(n^2)$ times $\Longrightarrow$ $O(n^5)$ calls to $\textsc{ExchBd}$ and $\textsc{ReduceV}$.

Running time analysis of Push-Relabel version of Schrijver

- The algorithm produces the optimal SFMin solution via the "Max Flow / Min Cut" Lemma
- I claim that the overall running time is $O(n^7 \mathrm{EO} + n^8)$:
  - There are $O(n^3)$ total $\textsc{Push}$es.
  - Each $\textsc{Push}$ calls $\textsc{ExchBd}$ and $\textsc{ReduceV}$ $O(n^2)$ times $\Longrightarrow$ $O(n^5)$ calls to $\textsc{ExchBd}$ and $\textsc{ReduceV}$.
  - Each call to $\textsc{ExchBd}$ costs $O(n^2 \mathrm{EO})$; each call to $\textsc{ReduceV}$ costs $O(n^3)$, for a total of $O(n^2 \mathrm{EO} + n^3)$ per iteration.

Running time analysis of Push-Relabel version of Schrijver

- The algorithm produces the optimal SFMin solution via the "Max Flow / Min Cut" Lemma
- I claim that the overall running time is $O(n^7\mathrm{EO} + n^8)$:
  - There are $O(n^3)$ total $\textsc{Push}$es.
  - Each $\textsc{Push}$ calls $\textsc{ExchBd}$ and $\textsc{ReduceV}$ $O(n^2)$ times $\Longrightarrow$ $O(n^5)$ calls to $\textsc{ExchBd}$ and $\textsc{ReduceV}$.
  - Each call to $\textsc{ExchBd}$ costs $O(n^2\mathrm{EO})$; each call to $\textsc{ReduceV}$ costs $O(n^3)$, for a total of $O(n^2\mathrm{EO} + n^3)$ per iteration.
  - Now $O(n^5)$ iterations times $O(n^2\mathrm{EO} + n^3)$ time per iterations gives $O(n^7\mathrm{EO} + n^8)$ total time.

Running time analysis of Push-Relabel version of Schrijver

- The algorithm produces the optimal SFMin solution via the "Max Flow / Min Cut" Lemma
- I claim that the overall running time is $O(n^7\mathrm{EO} + n^8)$:
  - There are $O(n^3)$ total PUSHes.
  - Each PUSH calls EXCHBD and REDUCEV $O(n^2)$ times $\Longrightarrow$ $O(n^5)$ calls to EXCHBD and REDUCEV.
  - Each call to EXCHBD costs $O(n^2\mathrm{EO})$; each call to REDUCEV costs $O(n^3)$, for a total of $O(n^2\mathrm{EO} + n^3)$ per iteration.
  - Now $O(n^5)$ iterations times $O(n^2\mathrm{EO} + n^3)$ time per iterations gives $O(n^7\mathrm{EO} + n^8)$ total time.
- Thus the Push-Relabel version of Schrijver's Algorithm is a strongly polynomial SFMin algorithm.

Outline

Notes on SFMin algorithms

- Another style of SFMin algorithms started with the Iwata, Fleischer, Fujishige (IFF) paper.

Notes on SFMin algorithms

- Another style of SFMin algorithms started with the Iwata, Fleischer, Fujishige (IFF) paper.
  - IFF relaxes $y \in B(f)$ to $z \in B(f + \rho\kappa)$, where ρ is a relaxation parameter that is scaled towards zero, and $\kappa(S) = \delta(S)$ w.r.t. the complete graph.

Notes on SFMin algorithms

- Another style of SFMin algorithms started with the Iwata, Fleischer, Fujishige (IFF) paper.
  - IFF relaxes $y \in B(f)$ to $z \in B(f + \rho\kappa)$, where ρ is a relaxation parameter that is scaled towards zero, and $\kappa(S) = \delta(S)$ w.r.t. the complete graph.
  - As a scaling algorithm, IFF is naturally weakly polynomial, but there are strongly polynomial and fully combinatorial versions of it.

Notes on SFMin algorithms

- Another style of SFMin algorithms started with the Iwata, Fleischer, Fujishige (IFF) paper.
  - IFF relaxes $y \in B(f)$ to $z \in B(f + \rho\kappa)$, where ρ is a relaxation parameter that is scaled towards zero, and $\kappa(S) = \delta(S)$ w.r.t. the complete graph.
  - As a scaling algorithm, IFF is naturally weakly polynomial, but there are strongly polynomial and fully combinatorial versions of it.
- A third style of SFMin algorithms stems from Orlin, including an Iwata-Orlin (IO) algorithm.

Notes on SFMin algorithms

- Another style of SFMin algorithms started with the Iwata, Fleischer, Fujishige (IFF) paper.
  - IFF relaxes $y \in B(f)$ to $z \in B(f + \rho\kappa)$, where ρ is a relaxation parameter that is scaled towards zero, and $\kappa(S) = \delta(S)$ w.r.t. the complete graph.
  - As a scaling algorithm, IFF is naturally weakly polynomial, but there are strongly polynomial and fully combinatorial versions of it.
- A third style of SFMin algorithms stems from Orlin, including an Iwata-Orlin (IO) algorithm.
  - These algorithms have different distance labels for each $i \in \mathcal{I}$.

Notes on SFMin algorithms

- Another style of SFMin algorithms started with the Iwata, Fleischer, Fujishige (IFF) paper.
  - IFF relaxes $y \in B(f)$ to $z \in B(f + \rho\kappa)$, where ρ is a relaxation parameter that is scaled towards zero, and $\kappa(S) = \delta(S)$ w.r.t. the complete graph.
  - As a scaling algorithm, IFF is naturally weakly polynomial, but there are strongly polynomial and fully combinatorial versions of it.
- A third style of SFMin algorithms stems from Orlin, including an Iwata-Orlin (IO) algorithm.
  - These algorithms have different distance labels for each $i \in \mathcal{I}$.
  - Orlin's Algorithm solves a more complicated system for its improving direction that preserves $S^0(y)$.

Notes on SFMin algorithms

- Another style of SFMin algorithms started with the Iwata, Fleischer, Fujishige (IFF) paper.
  - IFF relaxes $y \in B(f)$ to $z \in B(f + \rho\kappa)$, where ρ is a relaxation parameter that is scaled towards zero, and $\kappa(S) = \delta(S)$ w.r.t. the complete graph.
  - As a scaling algorithm, IFF is naturally weakly polynomial, but there are strongly polynomial and fully combinatorial versions of it.
- A third style of SFMin algorithms stems from Orlin, including an Iwata-Orlin (IO) algorithm.
  - These algorithms have different distance labels for each $i \in \mathcal{I}$.
  - Orlin's Algorithm solves a more complicated system for its improving direction that preserves $S^0(y)$.
  - The IO Algorithm concentrates on an ℓ_2-norm objective that was known to solve SFMin in a strong sense.

Current fastest SFMin algorithms

- The current fastest weakly polynomial SFMin algorithm is Iwata's Hybrid Algorithm, which runs in $O((n^4\mathrm{EO} + n^5)\log M)$ time.

Current fastest SFMin algorithms

- The current fastest weakly polynomial SFMin algorithm is Iwata's Hybrid Algorithm, which runs in $O((n^4\mathrm{EO} + n^5)\log M)$ time.
  - Hybrid uses some IFF ideas, and some Schrijver ideas.

Current fastest SFMin algorithms

- The current fastest weakly polynomial SFMin algorithm is Iwata's Hybrid Algorithm, which runs in $O((n^4\mathrm{EO} + n^5)\log M)$ time.
  - Hybrid uses some IFF ideas, and some Schrijver ideas.
- The current fastest strongly polynomial SFMin algorithm is Orlin's Algorithm, which runs in $O(n^5\mathrm{EO} + n^6)$ time.

Current fastest SFMin algorithms

- The current fastest weakly polynomial SFMin algorithm is Iwata's Hybrid Algorithm, which runs in $O((n^4 \mathrm{EO} + n^5) \log M)$ time.
  - Hybrid uses some IFF ideas, and some Schrijver ideas.
- The current fastest strongly polynomial SFMin algorithm is Orlin's Algorithm, which runs in $O(n^5 \mathrm{EO} + n^6)$ time.
  - This is even faster than the strongly polynomial version of Ellipsoid, which runs in $\tilde{O}(n^5 \mathrm{EO} + n^7)$ time.

Current fastest SFMin algorithms

- The current fastest weakly polynomial SFMin algorithm is Iwata's Hybrid Algorithm, which runs in $O((n^4\mathrm{EO} + n^5)\log M)$ time.
  - Hybrid uses some IFF ideas, and some Schrijver ideas.
- The current fastest strongly polynomial SFMin algorithm is Orlin's Algorithm, which runs in $O(n^5\mathrm{EO} + n^6)$ time.
  - This is even faster than the strongly polynomial version of Ellipsoid, which runs in $\tilde{O}(n^5\mathrm{EO} + n^7)$ time.
- The current fastest fully combinatorial SFMin algorithm is a version of the Iwata-Orlin Algorithm, which runs in $O((n^7 + n^8)\log n)$ time.

Empirical testing of SFMin algorithms

- The fastest running time we've seen is worse than $O(n^5)$, which is quite high.

Empirical testing of SFMin algorithms

- The fastest running time we've seen is worse than $O(n^5)$, which is quite high.
- Iwata did some empirical testing on perturbed Min Cut problems. He found that the best empirical run times were about $O(n^{3.5})$ from Hybrid, using $O(n^{2.5})$ calls to $\mathcal{E}$.

Empirical testing of SFMin algorithms

- The fastest running time we've seen is worse than $O(n^5)$, which is quite high.
- Iwata did some empirical testing on perturbed Min Cut problems. He found that the best empirical run times were about $O(n^{3.5})$ from Hybrid, using $O(n^{2.5})$ calls to $\mathcal{E}$.
- Fujishige et al did further testing on a wider range of instances (though still based on Min Cut). They found empirical performance varied from $O(n^{3.7})$ for Hybrid on Iwata's instances, to $O(n^{4.4})$ for Schrijver-PR on other instances.

Empirical testing of SFMin algorithms

- The fastest running time we've seen is worse than $O(n^5)$, which is quite high.
- Iwata did some empirical testing on perturbed Min Cut problems. He found that the best empirical run times were about $O(n^{3.5})$ from Hybrid, using $O(n^{2.5})$ calls to $\mathcal{E}$.
- Fujishige et al did further testing on a wider range of instances (though still based on Min Cut). They found empirical performance varied from $O(n^{3.7})$ for Hybrid on Iwata's instances, to $O(n^{4.4})$ for Schrijver-PR on other instances.
- There is another SFMin algorithm called the Fujishige-Wolfe (FW) Algorithm.

Empirical testing of SFMin algorithms

- The fastest running time we've seen is worse than $O(n^5)$, which is quite high.
- Iwata did some empirical testing on perturbed Min Cut problems. He found that the best empirical run times were about $O(n^{3.5})$ from Hybrid, using $O(n^{2.5})$ calls to $\mathcal{E}$.
- Fujishige et al did further testing on a wider range of instances (though still based on Min Cut). They found empirical performance varied from $O(n^{3.7})$ for Hybrid on Iwata's instances, to $O(n^{4.4})$ for Schrijver-PR on other instances.
- There is another SFMin algorithm called the Fujishige-Wolfe (FW) Algorithm.
  - It comes from a general algorithm for minimizing ℓ_2 distance to a polytope.

Empirical testing of SFMin algorithms

- The fastest running time we've seen is worse than $O(n^5)$, which is quite high.
- Iwata did some empirical testing on perturbed Min Cut problems. He found that the best empirical run times were about $O(n^{3.5})$ from Hybrid, using $O(n^{2.5})$ calls to $\mathcal{E}$.
- Fujishige et al did further testing on a wider range of instances (though still based on Min Cut). They found empirical performance varied from $O(n^{3.7})$ for Hybrid on Iwata's instances, to $O(n^{4.4})$ for Schrijver-PR on other instances.
- There is another SFMin algorithm called the Fujishige-Wolfe (FW) Algorithm.
  - It comes from a general algorithm for minimizing ℓ_2 distance to a polytope.
  - It has no known polynomial bound, but its empirical performance beat all other algorithms that Fujishige et al tested: $O(n^{3.3})$.

Related SFMin complexity results

- Given $T \subset E$, solving $\min_{S \subseteq T} f(S)$ and $\min_{T \subseteq S \subseteq E} f(S)$ are both also polynomial (homework).

Related SFMin complexity results

- Given $T \subset E$, solving $\min_{S \subseteq T} f(S)$ and $\min_{T \subseteq S \subseteq E} f(S)$ are both also polynomial (homework).
- Instead of considering all of 2^E, what if we have $\mathcal{F} \subset 2^E$? All these versions are polynomial:

Related SFMin complexity results

- Given $T \subset E$, solving $\min_{S \subseteq T} f(S)$ and $\min_{T \subseteq S \subseteq E} f(S)$ are both also polynomial (homework).
- Instead of considering all of 2^E, what if we have $\mathcal{F} \subset 2^E$? All these versions are polynomial:
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ (a ring family).

Related SFMin complexity results

- Given $T \subset E$, solving $\min_{S \subseteq T} f(S)$ and $\min_{T \subseteq S \subseteq E} f(S)$ are both also polynomial (homework).
- Instead of considering all of 2^E, what if we have $\mathcal{F} \subset 2^E$? All these versions are polynomial:
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ (a ring family).
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ only when $S \cap T \neq \emptyset$ (an intersecting family).

Related SFMin complexity results

- Given $T \subset E$, solving $\min_{S \subseteq T} f(S)$ and $\min_{T \subseteq S \subseteq E} f(S)$ are both also polynomial (homework).
- Instead of considering all of 2^E, what if we have $\mathcal{F} \subset 2^E$? All these versions are polynomial:
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ (a ring family).
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ only when $S \cap T \neq \emptyset$ (an intersecting family).
  - $\mathcal{F}$ is $2^E - \{\emptyset, E\}$.

Related SFMin complexity results

- Given $T \subset E$, solving $\min_{S \subseteq T} f(S)$ and $\min_{T \subseteq S \subseteq E} f(S)$ are both also polynomial (homework).
- Instead of considering all of 2^E, what if we have $\mathcal{F} \subset 2^E$? All these versions are polynomial:
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ (a ring family).
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ only when $S \cap T \neq \emptyset$ (an intersecting family).
  - $\mathcal{F}$ is $2^E - \{\emptyset, E\}$.
  - $\mathcal{F}$ is all S with $|S|$ odd/even.

Related SFMin complexity results

- Given $T \subset E$, solving $\min_{S \subseteq T} f(S)$ and $\min_{T \subseteq S \subseteq E} f(S)$ are both also polynomial (homework).
- Instead of considering all of 2^E, what if we have $\mathcal{F} \subset 2^E$? All these versions are polynomial:
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ (a ring family).
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ only when $S \cap T \neq \emptyset$ (an intersecting family).
  - $\mathcal{F}$ is $2^E - \{\emptyset, E\}$.
  - $\mathcal{F}$ is all S with $|S|$ odd/even.
  - $\mathcal{F}$ is all S with $|T \cap S|$ odd/even.

Related SFMin complexity results

- Given $T \subset E$, solving $\min_{S \subseteq T} f(S)$ and $\min_{T \subseteq S \subseteq E} f(S)$ are both also polynomial (homework).
- Instead of considering all of 2^E, what if we have $\mathcal{F} \subset 2^E$? All these versions are polynomial:
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ (a ring family).
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ only when $S \cap T \neq \emptyset$ (an intersecting family).
  - $\mathcal{F}$ is $2^E - \{\emptyset, E\}$.
  - $\mathcal{F}$ is all S with $|S|$ odd/even.
  - $\mathcal{F}$ is all S with $|T \cap S|$ odd/even.
  - …and more, see Goemans and Ramakrishnan.

Related SFMin complexity results

- Given $T \subset E$, solving $\min_{S \subseteq T} f(S)$ and $\min_{T \subseteq S \subseteq E} f(S)$ are both also polynomial (homework).
- Instead of considering all of 2^E, what if we have $\mathcal{F} \subset 2^E$? All these versions are polynomial:
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ (a ring family).
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ only when $S \cap T \neq \emptyset$ (an intersecting family).
  - $\mathcal{F}$ is $2^E - \{\emptyset, E\}$.
  - $\mathcal{F}$ is all S with $|S|$ odd/even.
  - $\mathcal{F}$ is all S with $|T \cap S|$ odd/even.
  - ...and more, see Goemans and Ramakrishnan.
- It is also polynomial to compute a compact representation of *all* SFMin solutions.

Related SFMin complexity results

- Given $T \subset E$, solving $\min_{S \subseteq T} f(S)$ and $\min_{T \subseteq S \subseteq E} f(S)$ are both also polynomial (homework).
- Instead of considering all of 2^E, what if we have $\mathcal{F} \subset 2^E$? All these versions are polynomial:
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ (a ring family).
  - $\mathcal{F}$ is closed under $\cap$ and $\cup$ only when $S \cap T \neq \emptyset$ (an intersecting family).
  - $\mathcal{F}$ is $2^E - \{\emptyset, E\}$.
  - $\mathcal{F}$ is all S with $|S|$ odd/even.
  - $\mathcal{F}$ is all S with $|T \cap S|$ odd/even.
  - ...and more, see Goemans and Ramakrishnan.
- It is also polynomial to compute a compact representation of *all* SFMin solutions.
- But don't get carried away: Solving $\min_{S \subseteq E : |S| = k} f(S)$ is NP Hard.

Future directions for SFMin algorithms

- Computational testing indicates that the linear algebra of ReduceV is a computational bottleneck.

Future directions for SFMin algorithms

- Computational testing indicates that the linear algebra of $\textsc{ReduceV}$ is a computational bottleneck.
  - This implies that trying to find another way to prove that $y \in B(f)$ besides convex hull might be worthwhile.

Future directions for SFMin algorithms

- Computational testing indicates that the linear algebra of REDUCEV is a computational bottleneck.
  - This implies that trying to find another way to prove that $y \in B(f)$ besides convex hull might be worthwhile.
  - Fujishige proposed combinatorial hull (see homework) as a possible replacement. This is an interesting research problem.

Future directions for SFMin algorithms

- Computational testing indicates that the linear algebra of $\textsc{ReduceV}$ is a computational bottleneck.
  - This implies that trying to find another way to prove that $y \in B(f)$ besides convex hull might be worthwhile.
  - Fujishige proposed combinatorial hull (see homework) as a possible replacement. This is an interesting research problem.
- Is there any way to get a non-trivial lower bound on the number of calls to $\mathcal{E}$ necessary to solve SFMin?

Future directions for SFMin algorithms

- Computational testing indicates that the linear algebra of $\textsc{ReduceV}$ is a computational bottleneck.
  - This implies that trying to find another way to prove that $y \in B(f)$ besides convex hull might be worthwhile.
  - Fujishige proposed combinatorial hull (see homework) as a possible replacement. This is an interesting research problem.
- Is there any way to get a non-trivial lower bound on the number of calls to $\mathcal{E}$ necessary to solve SFMin?
- There are other variations:

Future directions for SFMin algorithms

- Computational testing indicates that the linear algebra of $\textsc{ReduceV}$ is a computational bottleneck.
  - This implies that trying to find another way to prove that $y \in B(f)$ besides convex hull might be worthwhile.
  - Fujishige proposed combinatorial hull (see homework) as a possible replacement. This is an interesting research problem.
- Is there any way to get a non-trivial lower bound on the number of calls to $\mathcal{E}$ necessary to solve SFMin?
- There are other variations:
  - Parametric SFMin.

Future directions for SFMin algorithms

- Computational testing indicates that the linear algebra of ReduceV is a computational bottleneck.
  - This implies that trying to find another way to prove that $y \in B(f)$ besides convex hull might be worthwhile.
  - Fujishige proposed combinatorial hull (see homework) as a possible replacement. This is an interesting research problem.
- Is there any way to get a non-trivial lower bound on the number of calls to $\mathcal{E}$ necessary to solve SFMin?
- There are other variations:
  - Parametric SFMin.
  - Constrained SFMin; some versions are NP Hard, some are polynomial.

Future directions for SFMin algorithms

- Computational testing indicates that the linear algebra of $\textsc{ReduceV}$ is a computational bottleneck.
  - This implies that trying to find another way to prove that $y \in B(f)$ besides convex hull might be worthwhile.
  - Fujishige proposed combinatorial hull (see homework) as a possible replacement. This is an interesting research problem.
- Is there any way to get a non-trivial lower bound on the number of calls to $\mathcal{E}$ necessary to solve SFMin?
- There are other variations:
  - Parametric SFMin.
  - Constrained SFMin; some versions are NP Hard, some are polynomial.
  - Minimization of bisubmodular functions, a "signed" analogue of submodular functions.