CS 577- Intro to Algorithms

Network Flow

Dieter van Melkebeek

October 27, 2020

Outline

Outline

Changing gears

Specific problem vs paradigm

Outline

Changing gears

Specific problem vs paradigm

Motivation

- Networks
- Linear programming duality
- Reductions

Outline

Changing gears

Specific problem vs paradigm

Motivation

- Networks
- Linear programming duality
- Reductions

Plan

- Max flow and min cut
- Applications

Network

Network

Definition

A network N consists of:

- a digraph (V, E)
- edge capacities $c : E \to [0, \infty)$
- the source $s \in V$, which has indegree 0, and
- the sink $t \in V$, which has outdegree 0.

Flow

Flow

Definition

A flow is a mapping $f : E \to [0, \infty)$ satisfying:

Flow

Definition

A flow is a mapping $f : E \to [0, \infty)$ satisfying:

- [capacity constraints] $(\forall e \in E)\, f(e) \leq c(e)$

Flow

Definition

A flow is a mapping $f : E \to [0, \infty)$ satisfying:

- [capacity constraints] $(\forall e \in E)\, f(e) \leq c(e)$
- [conservation constraints] $(\forall v \in V \setminus \{s, t\})\, f_{\text{in}}(v) = f_{\text{out}}(v)$

Flow

Definition

A flow is a mapping $f : E \to [0, \infty)$ satisfying:

- [capacity constraints] $(\forall e \in E)\, f(e) \leq c(e)$
- [conservation constraints] $(\forall v \in V \setminus \{s, t\})\, f_{\text{in}}(v) = f_{\text{out}}(v)$
  where $f_{\text{in}}(v) \doteq \sum_{u \in V : e \doteq (u,v) \in E} f(e)$ and
  $f_{\text{out}}(v) \doteq \sum_{u \in V : e \doteq (v,u) \in E} f(e)$.

Flow

Definition

A flow is a mapping $f : E \to [0, \infty)$ satisfying:

- [capacity constraints] $(\forall e \in E)\, f(e) \leq c(e)$
- [conservation constraints] $(\forall v \in V \setminus \{s, t\})\, f_{\text{in}}(v) = f_{\text{out}}(v)$
  where $f_{\text{in}}(v) \doteq \sum_{u \in V : e \doteq (u,v) \in E} f(e)$ and
  $f_{\text{out}}(v) \doteq \sum_{u \in V : e \doteq (v,u) \in E} f(e)$.

Value of a flow

$\nu(f) \doteq f_{\text{out}}(s)$

Cut

Definition

An *st*-cut is a partition (S, T) of V such that $s \in S$ and $t \in T$.

pause

Note

- (S, T) is a partition of V if $S \cup T = V$ and $S \cap T = \emptyset$.

Cut

Definition

An *st*-cut is a partition (S, T) of V such that $s \in S$ and $t \in T$.

pause

Note

- (S, T) is a partition of V if $S \cup T = V$ and $S \cap T = \emptyset$.
- (S, T) can alternately be written as $(S, \overline{S})$ where $\overline{S} \doteq V \setminus S$.

Cut

Definition

An *st*-cut is a partition (S, T) of V such that $s \in S$ and $t \in T$.

pause

Note

- (S, T) is a partition of V if $S \cup T = V$ and $S \cap T = \emptyset$.
- (S, T) can alternately be written as $(S, \overline{S})$ where $\overline{S} \doteq V \setminus S$.
- Term "cut" is sometimes also used to denote the set of edges that cross the cut, i.e., $(E \cap S \times T) \cup (E \cap T \times S)$.

Invariance Property

Statement

For every flow f and st-cut (S, T) in a network $N = (V, E, c, s, t)$

$$f_{\text{out}}(S) - f_{\text{in}}(S) = \nu(f),$$

where $f_{\text{out}}(S) \doteq \sum_{e \in E \cap S \times T} f(e)$ and $f_{\text{in}}(S) \doteq \sum_{e \in E \cap T \times S} f(e)$.

Proof of Invariance Property

$$
\begin{array}{cccccc}
\nu(f) & \doteq & f_{\text{out}}(s) & - & f_{\text{in}}(s) & \\
0 & = & f_{\text{out}}(v) & - & f_{\text{in}}(v) & v \in S \setminus \{s\} \\
\hline
\end{array}
$$

$$
\begin{array}{ccccc}
\nu(f) & = & \sum\limits_{v \in S} f_{\text{out}}(v) & - & \sum\limits_{v \in S} f_{\text{in}}(v) \\
& = & \sum\limits_{v \in S} \sum\limits_{e \doteq (v,u) \in E} f(e) & - & \sum\limits_{v \in S} \sum\limits_{e \doteq (u,v) \in E} f(e) \\
& \overset{(*)}{=} & \sum\limits_{e \in E \cap S \times T} f(e) & - & \sum\limits_{e \in E \cap T \times SS} f(e) \\
& = & f_{\text{out}}(S) & - & f_{\text{in}}(S)
\end{array}
$$

Proof of Invariance Property

$$
\begin{array}{rclcll}
\nu(f) & \doteq & f_{\text{out}}(s) & - & f_{\text{in}}(s) & \\
0 & = & f_{\text{out}}(v) & - & f_{\text{in}}(v) & v \in S \setminus \{s\} \\
\hline
\end{array}
$$

$$
\begin{array}{rclcl}
\nu(f) & = & \sum_{v \in S} f_{\text{out}}(v) & - & \sum_{v \in S} f_{\text{in}}(v) \\
& = & \sum_{v \in S} \sum_{e \doteq (v,u) \in E} f(e) & - & \sum_{v \in S} \sum_{e \doteq (u,v) \in E} f(e) \\
& \overset{(*)}{=} & \sum_{e \in E \cap S \times T} f(e) & - & \sum_{e \in E \cap T \times SS} f(e) \\
& = & f_{\text{out}}(S) & - & f_{\text{in}}(S)
\end{array}
$$

(*)

type of $e \in E$	coefficient of $f(e)$
$S \times T$	1 - 0 $=$ 1
$S \times S$	1 - 1 $=$ 0
$T \times S$	0 - 1 $=$ -1
$T \times T$	0 - 0 $=$ 0

Weak duality

Weak duality

Capacity of an *st*-cut

$c(S, T) \doteq \sum_{e \in E \cap S \times T} c(e)$

Weak duality

Capacity of an *st*-cut

$c(S, T) \doteq \sum_{e \in E \cap S \times T} c(e)$

Statement

For every flow f and *st*-cut (S, T) in a network $N = (V, E, c, s, t)$

$$\nu(f) \leq c(S, T).$$

Proof of Weak duality

Proof of Weak duality

$$
\begin{aligned}
\nu(f) &= \sum_{e \in E \cap S \times T} f(e) - \sum_{e \in E \cap T \times S} f(e) \\
&\leq \sum_{e \in E \cap S \times T} c(e) - \sum_{e \in E \cap T \times S} 0 \\
&\doteq c(S, T)
\end{aligned}
$$

Proof of Weak duality

$$
\begin{aligned}
\nu(f) &= \sum_{e \in E \cap S \times T} f(e) - \sum_{e \in E \cap T \times S} f(e) \\
&\leq \sum_{e \in E \cap S \times T} c(e) - \sum_{e \in E \cap T \times S} 0 \\
&\doteq c(S, T)
\end{aligned}
$$

Note

Equality $\nu(f) = c(S, T)$ holds iff

Proof of Weak duality

$$\begin{aligned}
\nu(f) &= \sum_{e \in E \cap S \times T} f(e) - \sum_{e \in E \cap T \times S} f(e) \\
&\leq \sum_{e \in E \cap S \times T} c(e) - \sum_{e \in E \cap T \times S} 0 \\
&\doteq c(S, T)
\end{aligned}$$

Note

Equality $\nu(f) = c(S, T)$ holds iff

- $(\forall e \in E \cap S \times T)\, f(e) = c(e)$

Proof of Weak duality

$$
\begin{aligned}
\nu(f) &= \sum_{e \in E \cap S \times T} f(e) - \sum_{e \in E \cap T \times S} f(e) \\
&\leq \sum_{e \in E \cap S \times T} c(e) - \sum_{e \in E \cap T \times S} 0 \\
&\doteq c(S, T)
\end{aligned}
$$

Note

Equality $\nu(f) = c(S, T)$ holds iff

- $(\forall e \in E \cap S \times T)\, f(e) = c(e)$
- $(\forall e \in E \cap T \times S)\, f(e) = 0$.

Max Flow and Min Cut

Max Flow and Min Cut

Max flow

Input: network $N = (V, E, c, s, t)$

Output: flow f such that $\nu(f)$ is maximized

Max Flow and Min Cut

Max flow

Input: network $N = (V, E, c, s, t)$

Output: flow f such that $\nu(f)$ is maximized

Min cut

Input: network $N = (V, E, c, s, t)$

Output: st-cut (S, T) such that $c(S, T)$ is minimized

Max Flow and Min Cut

Max flow

Input: network $N = (V, E, c, s, t)$

Output: flow f such that $\nu(f)$ is maximized

Min cut

Input: network $N = (V, E, c, s, t)$

Output: st-cut (S, T) such that $c(S, T)$ is minimized

Weak duality

$$\max_{\text{flow } f} \nu(f) \leq \min_{st\text{-cut } (S,T)} c(S, T)$$

Max Flow and Min Cut

Max flow

Input: network $N = (V, E, c, s, t)$

Output: flow f such that $\nu(f)$ is maximized

Min cut

Input: network $N = (V, E, c, s, t)$

Output: st-cut (S, T) such that $c(S, T)$ is minimized

Weak duality

$$\max_{\text{flow } f} \nu(f) \leq \min_{st\text{-cut } (S,T)} c(S, T)$$

We'll show next lecture that equality always holds (strong duality).

Path Augmentation - first attempt

Path Augmentation - first attempt

Algorithm

1. Start with $f \equiv 0$.
2. While there is an *st*-path P along which more flow can be pushed, additionally push as much flow along P as possible.
3. Return f.

Residual Network

Consider a flow f in $N = (V, E, c, s, t)$.

Definition

The residual network $N_f = (V, E_f, c_f, s, t)$ has:

Residual Network

Consider a flow f in $N = (V, E, c, s, t)$.

Definition

The residual network $N_f = (V, E_f, c_f, s, t)$ has:

- For each $e \in E$ with $f(e) < c(e)$, an edge e in E_f with $c_f(e) \doteq c(e) - f(e)$.

Residual Network

Consider a flow f in $N = (V, E, c, s, t)$.

Definition

The residual network $N_f = (V, E_f, c_f, s, t)$ has:

- For each $e \in E$ with $f(e) < c(e)$, an edge e in E_f with $c_f(e) \doteq c(e) - f(e)$.
- For each $e = (u, v) \in E$ with $f(e) > 0$, an edge $e' \doteq (v, u)$ in E_f with $c_f(e') \doteq f(e)$.

Path Augmentation

Algorithm

1. Start with $f \equiv 0$.
2. While there is an *st*-path P in N_f, additionally push as much flow along P as possible.
3. Return f.