# C&O 355
# Mathematical Programming
# Fall 2010
# Lecture 22

N. Harvey

# Topics

- Kruskal's Algorithm for the
  Max Weight Spanning Tree Problem
- Vertices of the Spanning Tree Polytope

# Review of Lecture 21

- Defined **spanning tree polytope**

$$P_{ST} = \left\{ \begin{array}{rcl} x(E) & = & n-1 \\ x(C) & \leq & n-\kappa(C) \;\; \forall C \subsetneq E \\ x & \geq & 0 \end{array} \right\}$$

where $\kappa$(C) = # connected components in (V,C).

- We showed, for any spanning tree T, its **characteristic vector** is in $P_{ST}$.
- We showed how to optimize over $P_{ST}$ in polynomial time by the ellipsoid method, even though there are **exponentially many constraints**
  - This is a bit complicated: it uses the Min s-t Cut problem as a separation oracle.

# How to solve combinatorial IPs?

(From Lecture 17)

- Two common approaches
  1. Design combinatorial algorithm that directly solves IP
     - Often such algorithms have a nice LP interpretation

     TODAY
  2. Relax IP to an LP; prove that they give same solution; solve LP by the ellipsoid method
     - Need to show special structure of the LP's extreme points
     - Sometimes we can analyze the extreme points **combinatorially**
     - Sometimes we can use **algebraic** structure of the constraints. For example, if constraint matrix is **Totally Unimodular** then IP and LP are equivalent

Perfect Matching

Max Flow, Max Matching

# Kruskal's Algorithm

- Let G = (V,E) be a connected graph, n=|V|, m=|E|
- Edges are weighted: $w_e \in \mathbb{R}$ for every $e \in E$

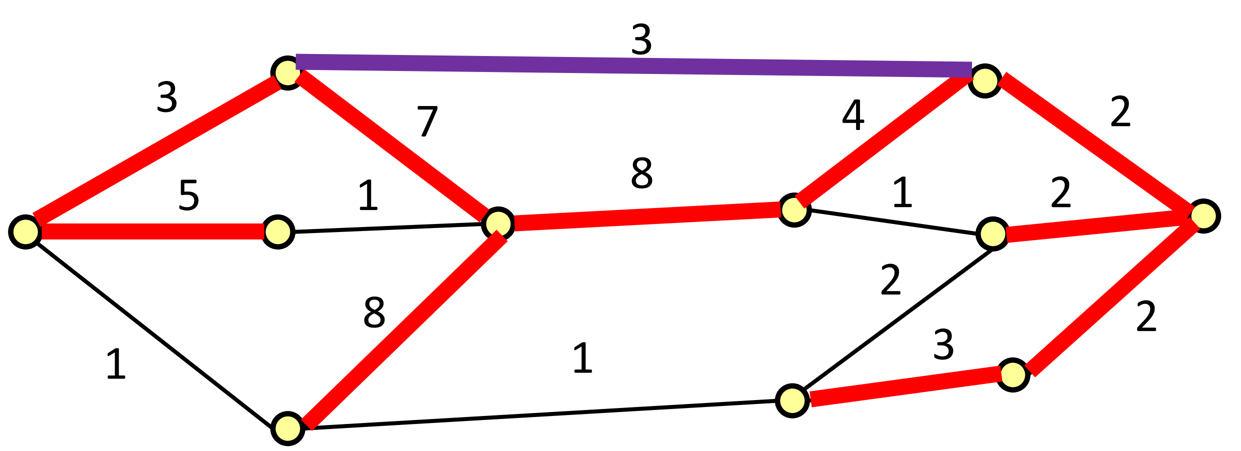
> Order E as $(e_1, \ldots, e_m)$, where $w_{e1} \geq w_{e2} \geq \ldots \geq w_{em}$
> Initially $T = \emptyset$
> For i=1,...,m
>     If the ends of $e_i$ are in different components of (V,T)
>         Add $e_i$ to T

- We will show:
- **Claim:** When this algorithm adds an edge to T, no cycle is created.
- **Claim:** At the end of the algorithm, T is connected.
- **Theorem:** This algorithm outputs a **maximum-cost** spanning tree.

# Kruskal's Algorithm

- Let G = (V,E) be a connected graph, n=|V|, m=|E|
- Edges are weighted: $w_e \in \mathbb{R}$ for every $e \in E$

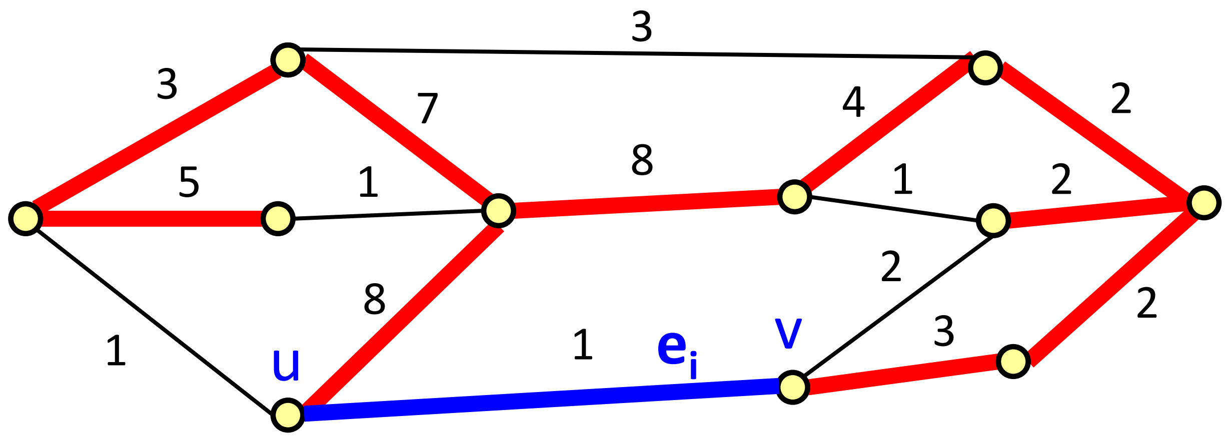
Order E as ($e_1$, …, $e_m$), where $w_{e1} \geq w_{e2} \geq \ldots \geq w_{em}$
Initially T = $\emptyset$
For i=1,…,m
    If the ends of $e_i$ are in different components of (V,T)
        Add $e_i$ to T



Order E as $(e_1, \ldots, e_m)$, where $w_{e1} \geq w_{e2} \geq \ldots \geq w_{em}$
Initially $T = \emptyset$
For i=1,...,m
    If the ends of $e_i$ are in different components of (V,T)
        Add $e_i$ to T

- **Claim:** When this algorithm adds an edge to T, no cycle is created.
- **Proof:** Let $\mathbf{e_i} = \{u,v\}$.

$T \cup \{\mathbf{e_i}\}$ contains a cycle iff there is a path from u to v in (V,T).



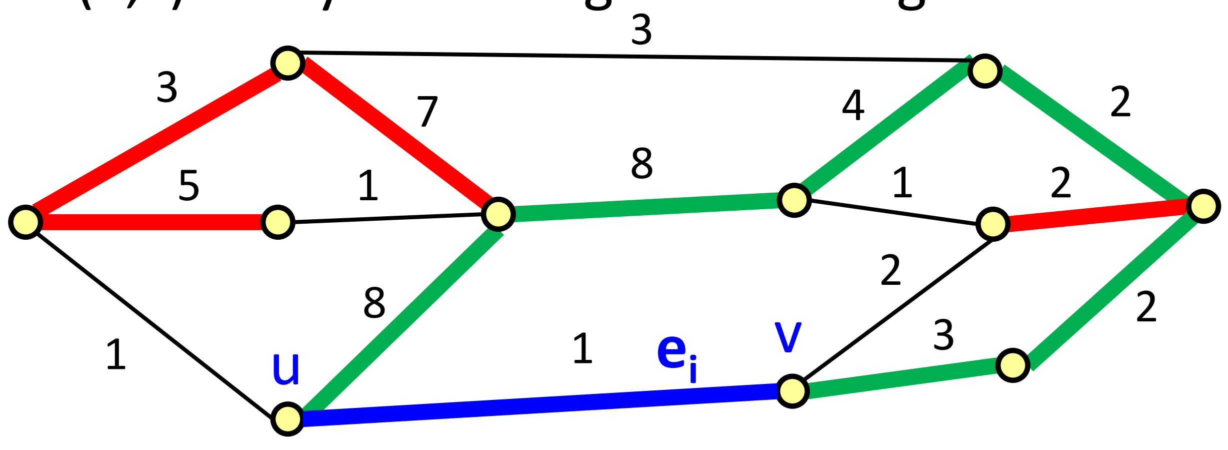
Order E as $(e_1, \ldots, e_m)$, where $w_{e1} \geq w_{e2} \geq \ldots \geq w_{em}$
Initially $T = \emptyset$
For i=1,...,m
    If the ends of $e_i$ are in different components of (V,T)
        Add $e_i$ to T

- **Claim:** When this algorithm adds an edge to T, no cycle is created.
- **Proof:** Let $\mathbf{e_i}$ = {u,v}.

  $T \cup \{\mathbf{e_i}\}$ contains a cycle iff there is a **path** from u to v in (V,T).

  Since $e_i$ only added when u and v are in different components, no such path exists.

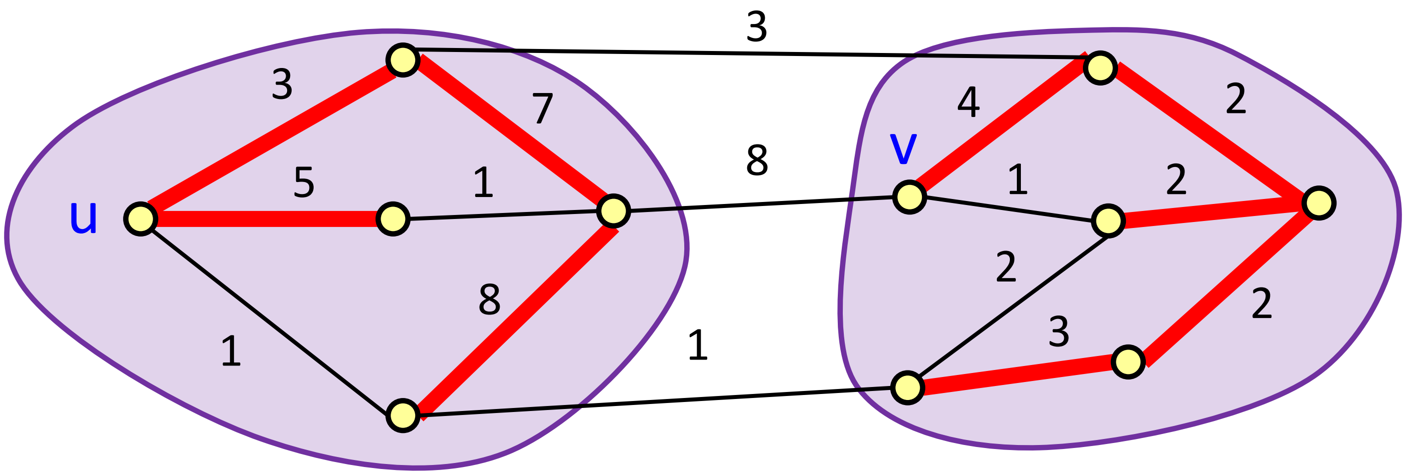
  Therefore (V,T) is acyclic throughout the algorithm. ■



Order E as $(e_1, \ldots, e_m)$, where $w_{e1} \geq w_{e2} \geq \ldots \geq w_{em}$
Initially $T = \emptyset$
For i=1,...,m
    If the ends of $e_i$ are in different components of (V,T)
        Add $e_i$ to T

- **Claim:** At the end of the algorithm, T is connected.
- **Proof:** Suppose not.

Then there are vertices u and v in different components of (V,T).



Order E as ($e_1$, ..., $e_m$), where $w_{e1} \geq w_{e2} \geq \ldots \geq w_{em}$

Initially T = $\emptyset$

For i=1,...,m

    If the ends of $e_i$ are in different components of (V,T)

        Add $e_i$ to T

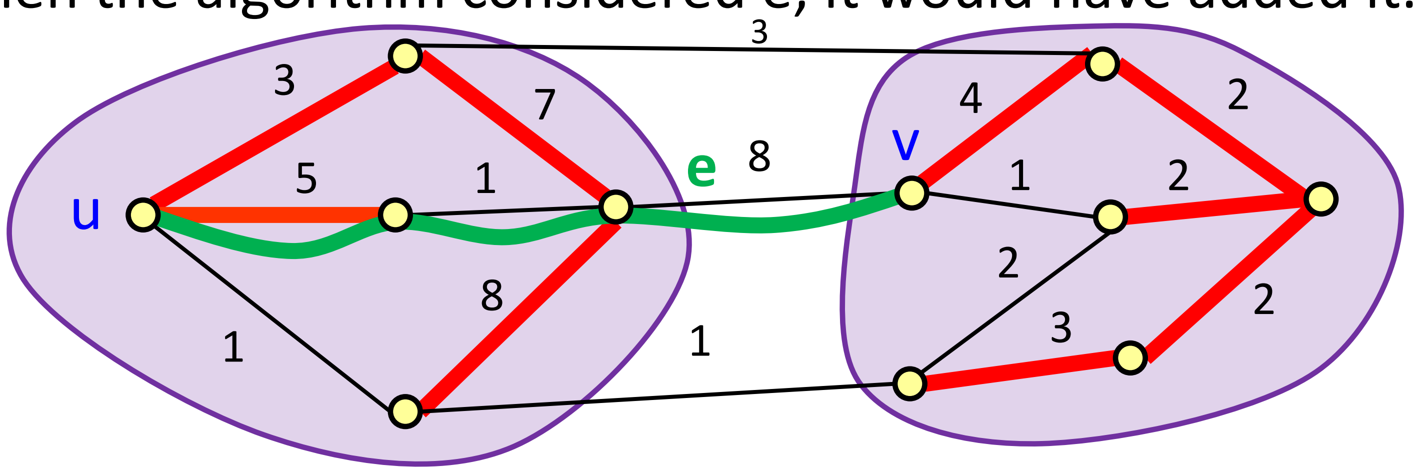
- **Claim:** At the end of the algorithm, T is connected.
- **Proof:** Suppose not.

  Then there are vertices u and v in different components of (V,T).

  Since G is connected, there is a u-v path **P** in G.

  Some edge **e**$\in$**P** must connect different components of (V,T).

  When the algorithm considered e, it would have added it. ■



# Our Analysis So Far

Order E as ($e_1$, ..., $e_m$), where $w_{e1} \geq w_{e2} \geq \ldots \geq w_{em}$
Initially T = $\emptyset$
For i=1,...,m
    If the ends of $e_i$ are in different components of (V,T)
        Add $e_i$ to T

- We have shown:
- **Claim:** When this algorithm adds an edge to T, no cycle is created.
- **Claim:** At the end of the algorithm, T is connected.
- So T is an acyclic, connected subgraph, i.e., a spanning tree.

- We will show:
- **Theorem:** This algorithm outputs a **maximum-cost** spanning tree.

# Main Theorem

Order E as ($e_1$, ..., $e_m$), where $w_{e1} \geq w_{e2} \geq ... \geq w_{em}$
Initially T = $\emptyset$
For i=1,...,m
    If the ends of $e_i$ are in different components of (V,T)
        Add $e_i$ to T

- In fact, we will show a stronger fact:
- **Theorem:** Let x be the characteristic vector of T at end of algorithm. Then x is an optimal solution of max { $w^T x$ : $x \in P_{ST}$ }, where $P_{ST}$ is the spanning tree polytope:

$$P_{ST} = \left\{ \begin{array}{rcll} x(E) & = & n-1 & \\ x(C) & \leq & n-\kappa(C) & \forall C \subsetneq E \\ x & \geq & 0 & \end{array} \right\}$$

# Optimal LP Solutions

- We saw last time that the characteristic vector of any spanning tree is feasible for $P_{ST}$.
- We will modify Kruskal's Algorithm to output a feasible **dual** solution as well.
- These primal & dual solutions will satisfy the complementary slackness conditions, and hence both are optimal.
- The dual of max { $w^T x : x \in P_{ST}$ } is

$$\begin{array}{llll} \min & \sum_{C \subseteq E} (n - \kappa(C)) y_C & & \\ \text{s.t.} & \sum_{C \ni e} y_C & \geq w_e & \forall e \in E \\ & y_C & \geq 0 & \forall C \subsetneq E \end{array}$$

# Complementary Slackness Conditions

(From Lecture 5)

Let x be feasible for primal and y be feasible for dual.

| | Primal | Dual |
|---|---|---|
| Objective | max $c^T x$ | min $b^T y$ |
| Variables | $x_1, \ldots, x_n$ | $y_1, \ldots, y_m$ |
| Constraint matrix | A | $A^T$ |
| Right-hand vector | b | c |
| Constraints versus Variables | $i^{th}$ constraint: $\leq$<br>$i^{th}$ constraint: $\geq$<br>$i^{th}$ constraint: $=$<br><br>$x_j \geq 0$<br>$x_j \leq 0$<br>$x_j$ unrestricted | $y_i \geq 0$<br>$y_i \leq 0$<br>$y_i$ unrestricted<br><br>$j^{th}$ constraint: $\geq$<br>$j^{th}$ constraint: $\leq$<br>$j^{th}$ constraint: $=$ |

for all i, equality holds either for primal or dual

**and**

for all j, equality holds either for primal or dual

$\Leftrightarrow$

x and y are both optimal

# Complementary Slackness

- **Primal:**

$$
\begin{array}{lll}
\max & w^{\mathsf{T}} x & \\
\text{s.t.} & x(E) & = n - 1 \\
 & x(C) & \leq n - \kappa(C) \quad \forall C \subsetneq E \\
 & x & \geq 0
\end{array}
$$

- **Dual:**

$$
\begin{array}{llll}
\min & \sum_{C \subseteq E} (n - \kappa(C)) y_C & & \\
\text{s.t.} & \sum_{C \ni e} y_C & \geq w_e & \forall e \in E \\
 & y_C & \geq 0 & \forall C \subsetneq E
\end{array}
$$

- **Complementary Slackness Conditions:**

$$
\text{(CS1)} \quad \text{For all } e \in E, \quad x_e > 0 \implies \sum_{C \ni e} y_C = w_e
$$

$$
\text{(CS2)} \quad \text{For all } C \subsetneq E, \quad y_C > 0 \implies x(C) = n - \kappa(C)
$$

If x and y satisfy these conditions, both are optimal.

# “Primal-Dual” Kruskal Algorithm

- **Notation:** Let $R_i = \{e_1,\ldots,e_i\}$ and $w_{e_{m+1}} = 0$

> Order E as $(e_1, \ldots, e_m)$, where $w_{e1} \geq w_{e2} \geq \ldots \geq w_{em}$
> Initially $T = \emptyset$ and $y = 0$
> For $i=1,\ldots,m$
>     Set $y_{R_i} = w_{e_i} - w_{e_{i+1}}$
>     If the ends of $e_i$ are in different components of (V,T)
>         Add $e_i$ to T

- **Claim:** $y$ is feasible for dual LP.
- **Proof:** $y_C \geq 0$ for all $C \subsetneq E$, since $w_{e_i} \geq w_{e_{i+1}}$. (Except when i=m)

Consider any edge $e_i$. The only non-zero $y_C$ with $e_i \in C$ are $y_{R_k}$ for $k \geq i$.

So $\sum_{C \ni e_i} y_C = \sum_{k=i}^{m} y_{R_i} = \sum_{k=i}^{m} (w_{e_i} - w_{e_{i+1}}) = w_{e_i}$. ■

- **Lemma:** Suppose $B \subseteq E$ and $C \subseteq E$ satisfy $|B \cap C| < n-\kappa(C)$. Let $\kappa=\kappa(C)$. Let the components of $(V,C)$ be $(V_1,C_1), \ldots, (V_\kappa,C_\kappa)$. Then for some j, $(V_j, B \cap C_j)$ is not connected.
- **Proof:**

  We showed last time that $n - \kappa = \sum_{j=1}^{\kappa}(|V_j| - 1)$

  So $\sum_{j=1}^{\kappa}|B \cap C_j| = |B \cap C| < n - \kappa = \sum_{j=1}^{\kappa}(|V_j| - 1)$

  So, for some j, $|B \cap C_j| < |V_j|-1$.

  So $B \cap C_j$ doesn't have enough edges to form a tree spanning $V_j$.

  So $(V_j, B \cap C_j)$ is not connected. ■

- Let x be the characteristic vector of T at end of algorithm.
- **Claim:** x and y satisfy the complementary slackness conditions.
- **Proof:** We showed $\sum_{C \ni e} y_C = w_e$ for **every** edge e, so CS1 holds.

  Let's check CS2. We only have $y_C > 0$ if $C = R_i$ for some i.

  So suppose $x(R_i) < n - \kappa(R_i)$ for some i.

  Recall that $x(R_i) = |T \cap R_i|$. (Since x is characteristic vector of T.)

  Let the components of $(V, R_i)$ be $(V_1, C_1)$, ..., $(V_\kappa, C_\kappa)$.

  By previous lemma, for some a, $(V_a, T \cap C_a)$ is not connected.

  There are vertices $u, v \in V_a$ such that

  - u and v are not connected in $(V_a, T \cap C_a)$
  - there is a path $P \subseteq C_a$ connecting u and v in $(V_a, C_a)$

  So, some edge $e_b \in P$ connects two components of $(V_a, T \cap C_a)$, which are also two components of $(V, T \cap R_i)$.

  Note that $T \cap R_i$ is the partial tree at step i of the algorithm.

  So when the algorithm considered $e_b$, it would have added it. ■

# Vertices of the Spanning Tree Polytope

- **Corollary:** Every vertex of $P_{ST}$ is the characteristic vector of a spanning tree.
- **Proof:**
  Consider any vertex **x** of spanning tree polytope.
  By definition, there is a weight vector **w** such that **x** is the **unique optimal solution** of max{ $\mathbf{w}^T x : x \in P_{ST}$ }.
  If we ran Kruskal's algorithm with the weights **w**, it would output an **optimal solution** to max{ $\mathbf{w}^T x : x \in P_{ST}$ } that is the **characteristic vector** of a spanning tree **T**.
  Thus **x** is the characteristic vector of **T**. ■
- **Corollary:** The World's Worst Spanning Tree Algorithm (in Lecture 21) outputs a max weight spanning tree.

# What’s Next?

- Future C&O classes you could take

| If you liked… | You might like… |
|---|---|
| Max Flows, Min Cuts, Spanning Trees | C&O 351 “Network Flows”<br>C&O 450 “Combinatorial Optimization”<br>C&O 453 “Network Design” |
| Integer Programs, Polyhedra | C&O 452 “Integer Programming” |
| Konig’s Theorem | C&O 342 “Intro to Graph Theory”<br>C&O 442 “Graph Theory”<br>C&O 444 “Algebraic Graph Theory” |
| Convex Functions,<br>Subgradient Inequality,<br>KKT Theorem | C&O 367 “Nonlinear Optimization”<br>C&O 463 “Convex Optimization”<br>C&O 466 “Continuous Optimization” |
| Semidefinite Programs | C&O 471 “Semidefinite Optimization” |

- If you’re unhappy that the ellipsoid method is too slow, you can learn about practical methods in:
  - C&O 466: Continuous Optimization