# COT 6936: Topics in Algorithms


## Giri Narasimhan

ECS 254A / EC 2443; Phone: x3748

giri@cs.fiu.edu


http://www.cs.fiu.edu/~giri/teach/COT6936_S12.html

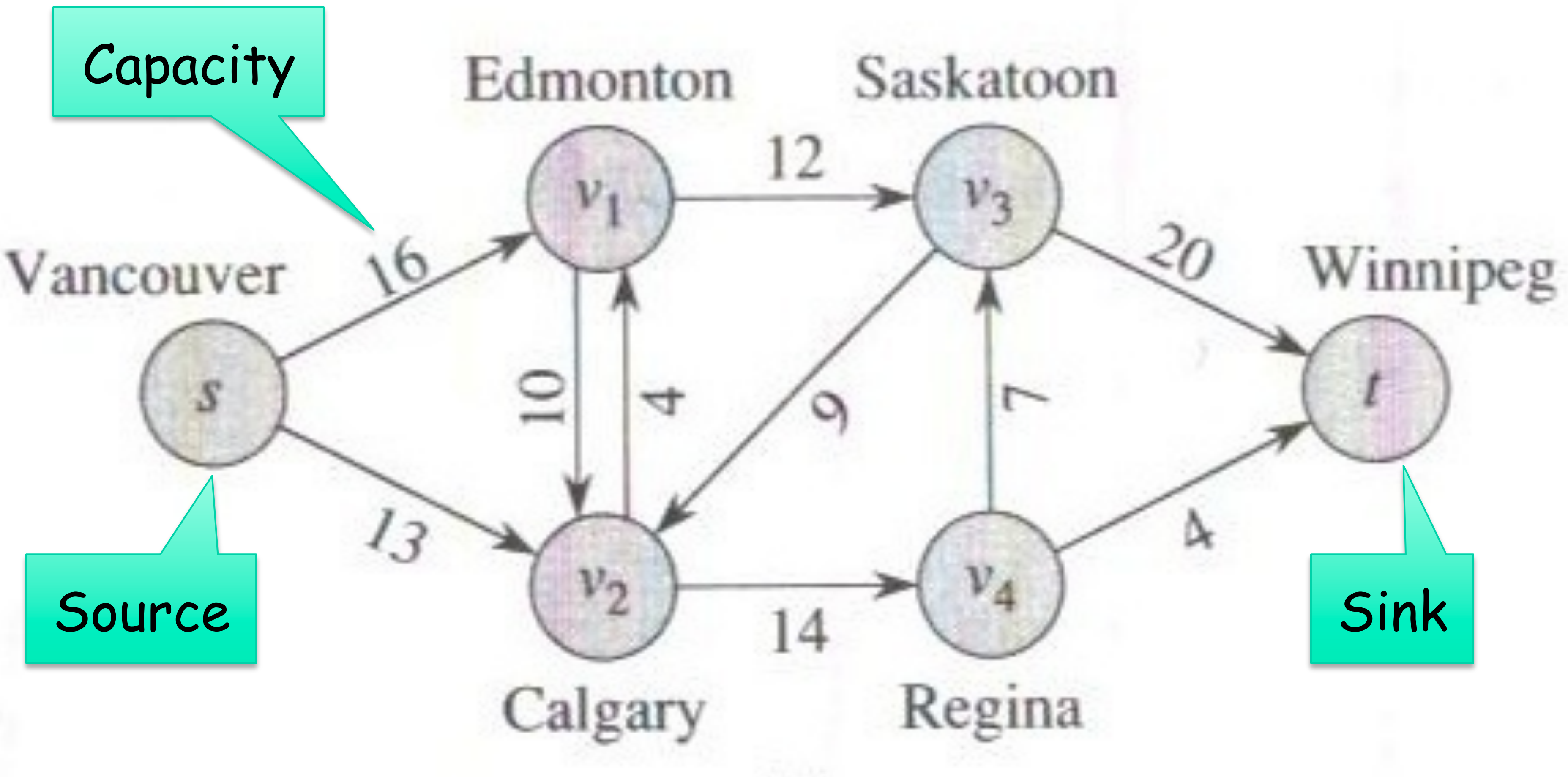
https://moodle.cis.fiu.edu/v2.1/course/view.php?id=**174**

# Types of networks & Types of queries

- Road, highway, rail
- Electrical power, water, oil, gas, sewer
- Internet, phone, wireless, sensor
- …

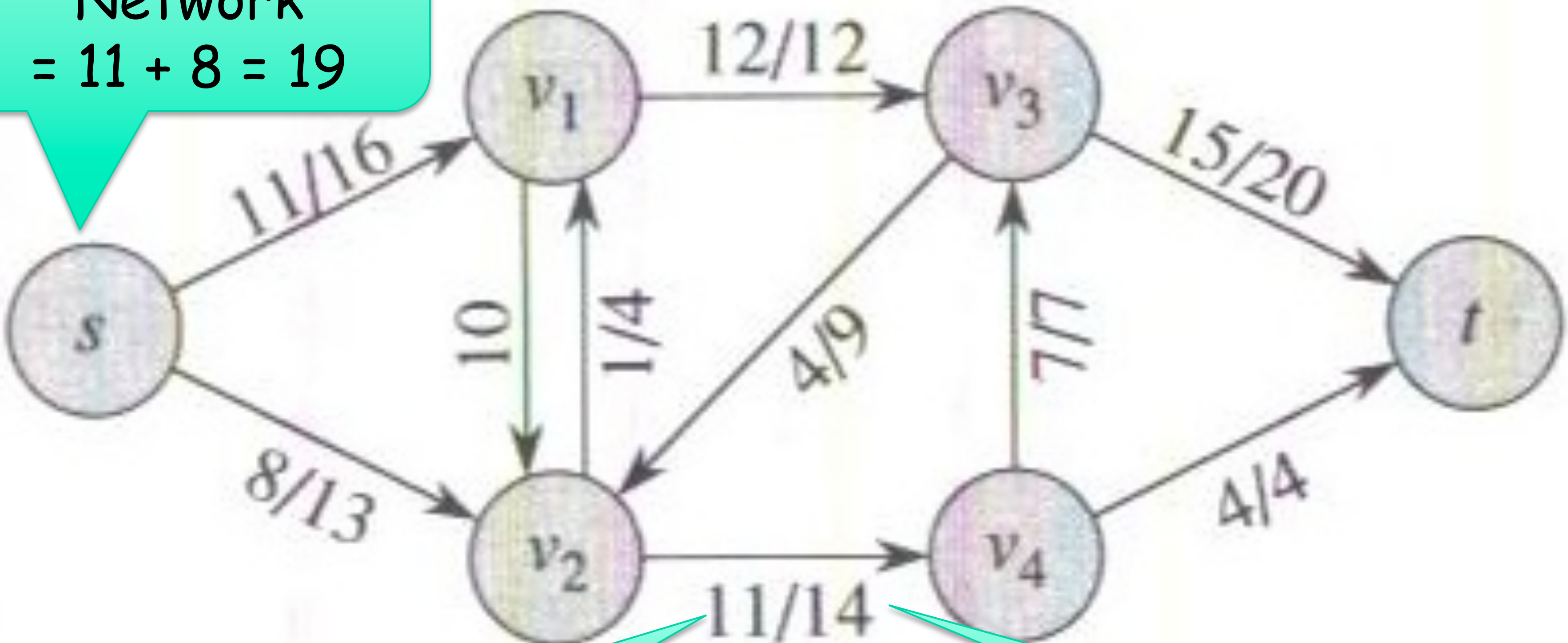
- (1950s) How quickly can Soviet Union get supplies through its rail network to Europe?
- Which links to destroy to reduce flow to under a threshold?

# Network Flow: Example



# Network Flow: Example of a flow

Total Flow in Network
= 11 + 8 = 19




Flow value along edge

Capacity of edge

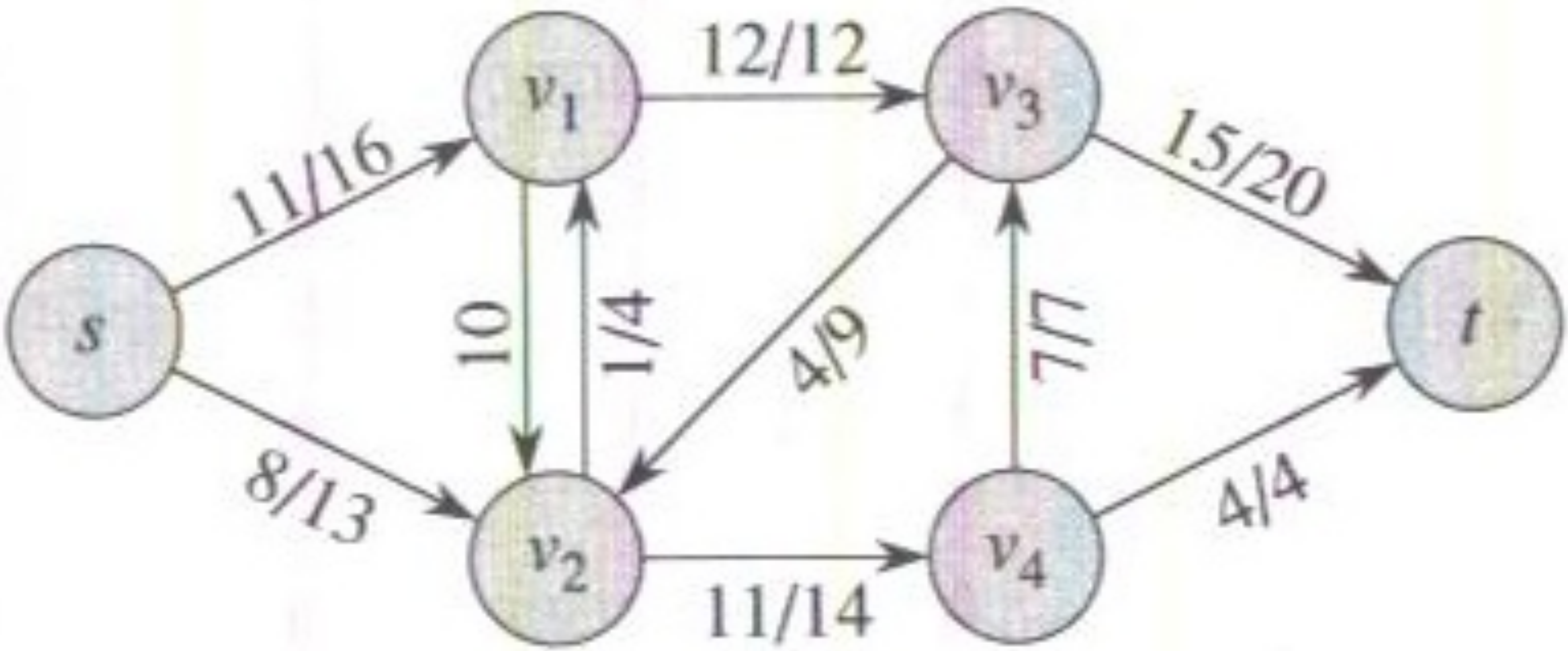
# Network Flow

- Directed graph G(V,E) with capacity function on edges given by non-negative function c: E(G) ➔ $\mathcal{R}^+$.
  - Capacity of each edge, e, is given by c(e)
  - Source vertex s
  - Sink vertex t
- Flow function f is a non-negative function of the edges
  - f: E(G) ➔ $\mathcal{R}^+$
  - Capacity constraints: $f(e) \leq c(e)$
  - Flow conservation constraints: For all vertices except source and sink, sum of flow values along edges entering a vertex equals sum of flow values along edges leaving that vertex
- **Flow value**: sum of flow values from source vertex (or sum of flow into sink vertex)

# Flow Conservation

- For any legal flow function:
  - Flow out of source = Flow into sink (Why?)

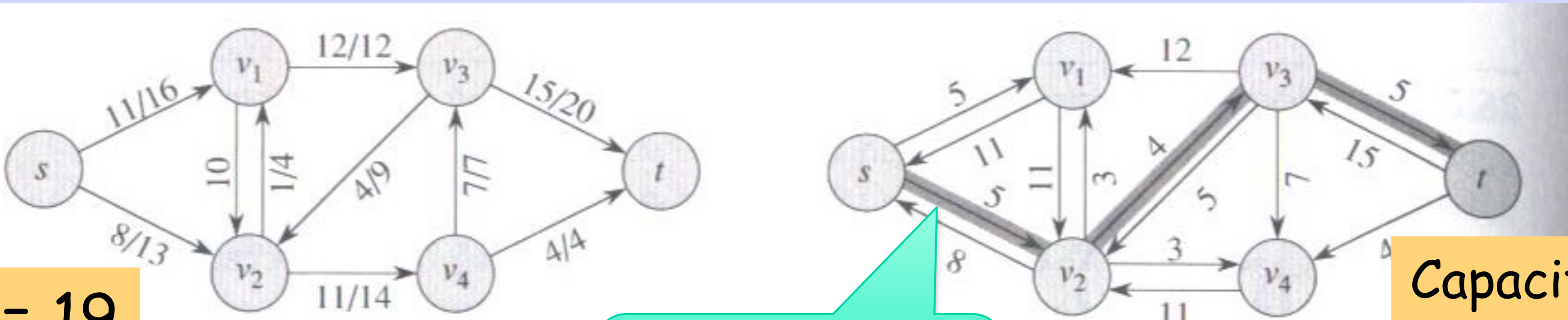
# Network Flow: How to increase flow




Find path with residual capacity and increase flow along path.

- Path s to $v_1$ to $v_3$ to t has no residual capacity
  - edge $v_1$ to $v_3$ is saturated
- Path s to $v_2$ to $v_3$ to t has residual capacity

# Residual Flows and Augmenting Paths




(a) (b)

Flow = 19

Augmenting Path

Capacity of augmenting path = 4

Flow = 23

# Residual Flow Network: Definition

- Directed Graph $G(V,E)$ with capacity function $c$ and flow function $f$
- Residual flow network $G_f(V,E')$
  - For every edge $e = (u,v)$ in $E$ with $f(e) < c(e)$, there are two edges in $E'$: $(u,v)$ and $(v,u)$ with capacities $c(e) - f(e)$ and $f(e)$, respectively
  - For every edge $e = (u,v)$ in $E$ with $f(e) = c(e)$, there is one edge in $E'$: $(v,u)$ with capacity $f(e)$
  - For every edge $e = (u,v)$ in $E$ with $f(e) = 0$, there is one edge in $E'$: $(u,v)$ with capacity $f(e)$

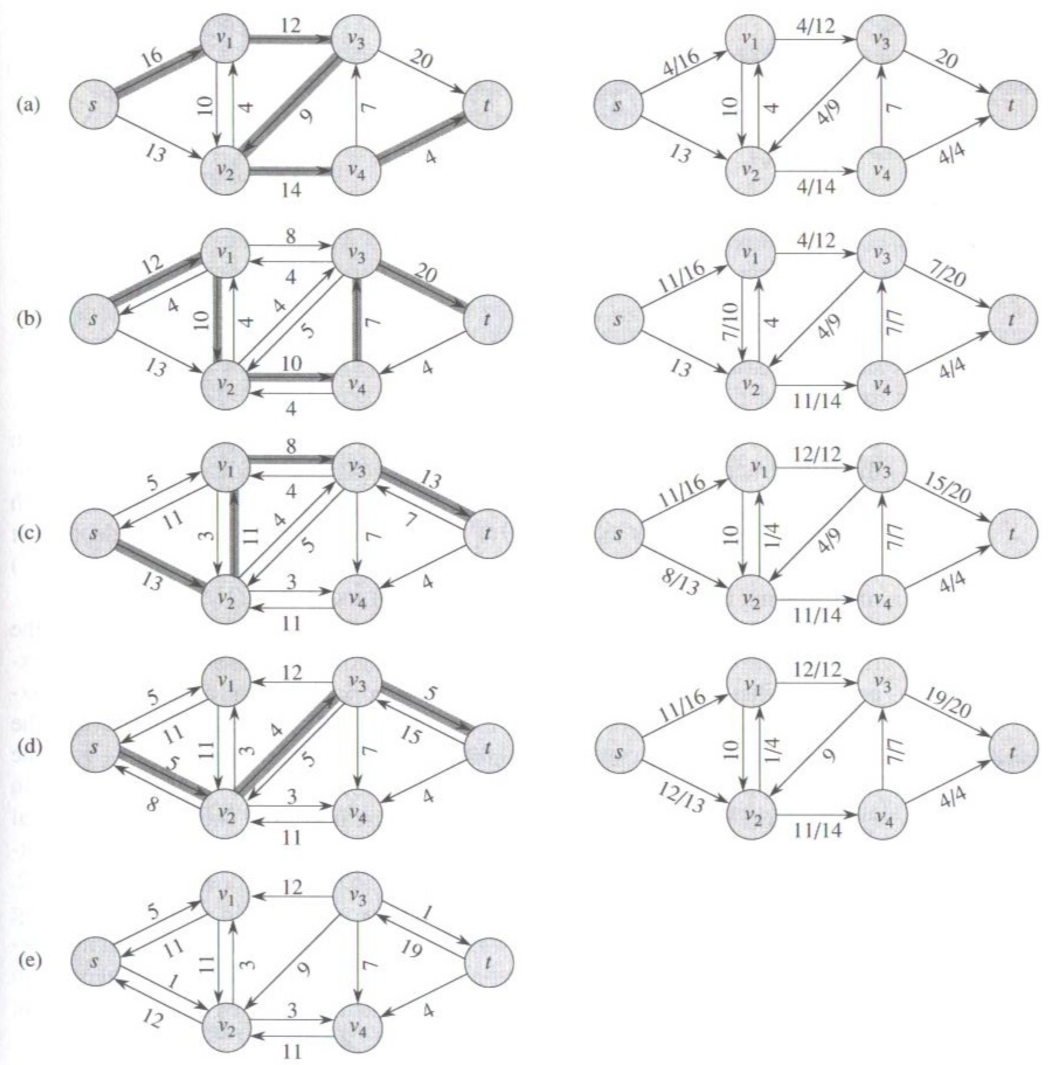
# Ford Fulkerson Algorithm

- Initialize flow f to 0.
- While (there exists augmenting path p from s to t) do
  - Augment flow along augmenting path p
- Return flow f as maximum flow from s to t

# Ford Fulkerson Algorithm

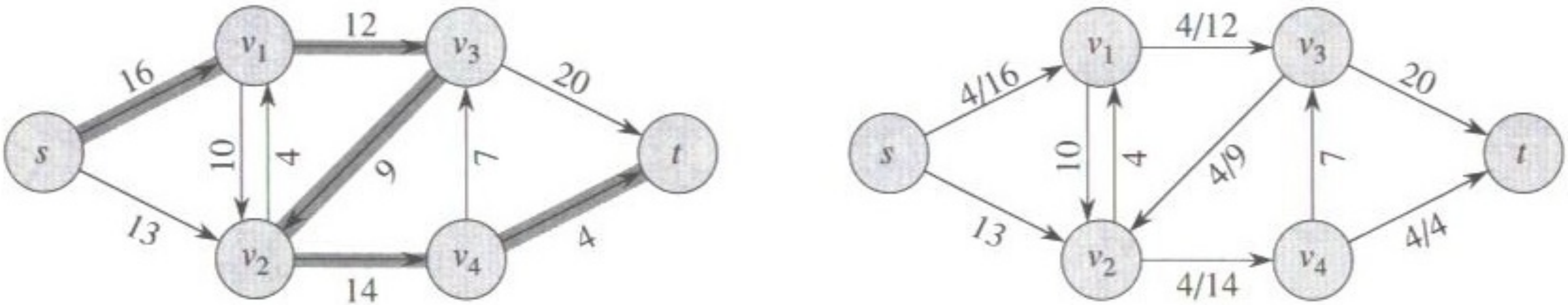
- Initialize flow f to 0.
- While (there exists directed path p from s to t in residual flow network $G_f$) do
  - Augment flow along augmenting path p
- Return flow f as maximum flow from s to t

# Ford-Fulkerson Method: Example

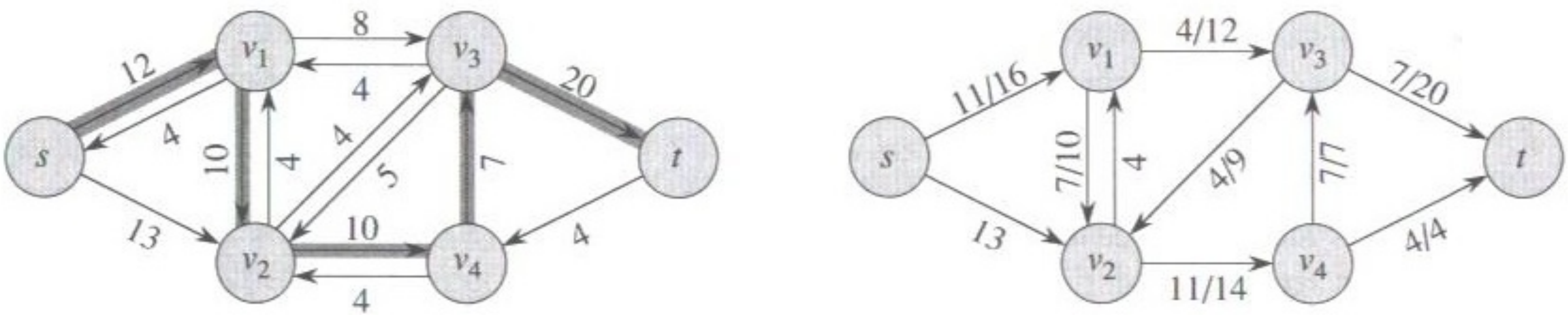


# Ford-Fulkerson Method: Example

A

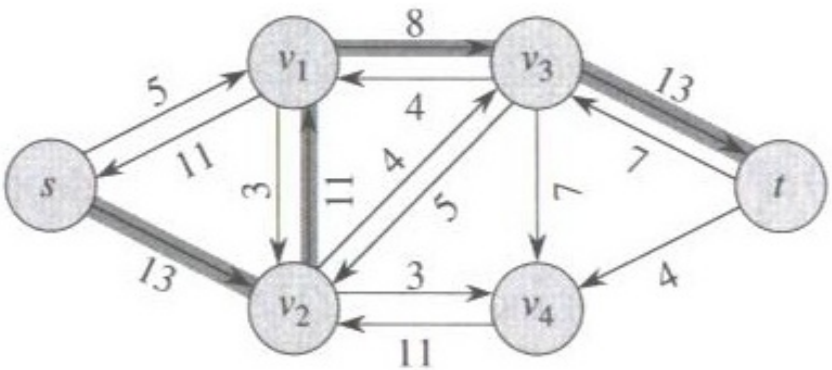


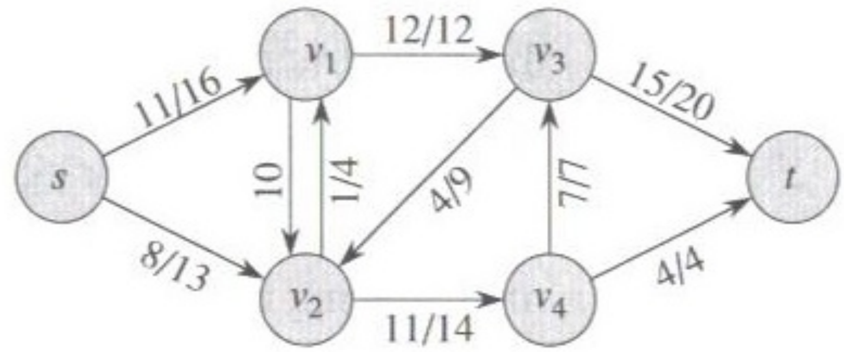

B



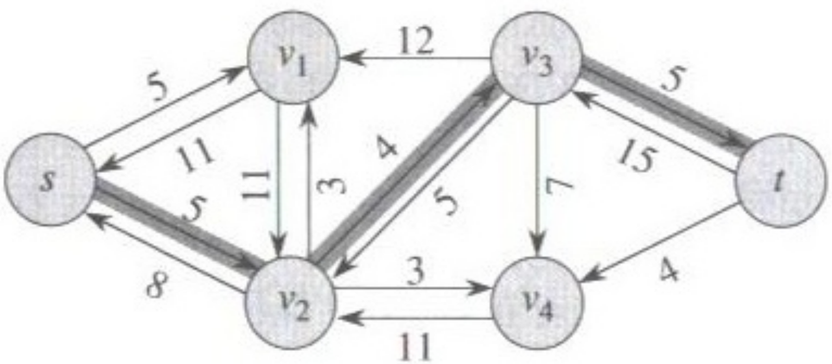
# Ford-Fulkerson Method: Example

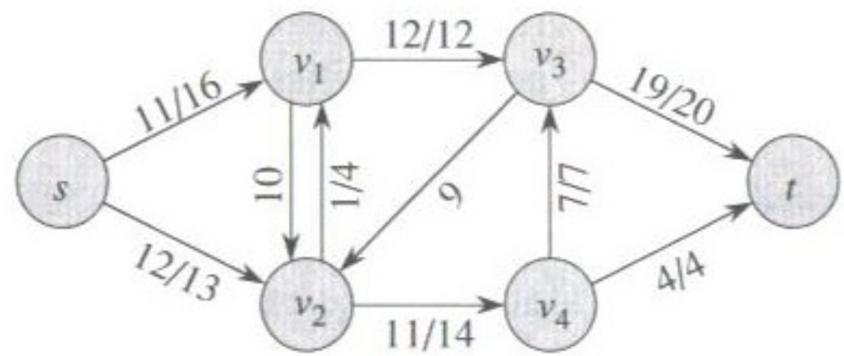
C



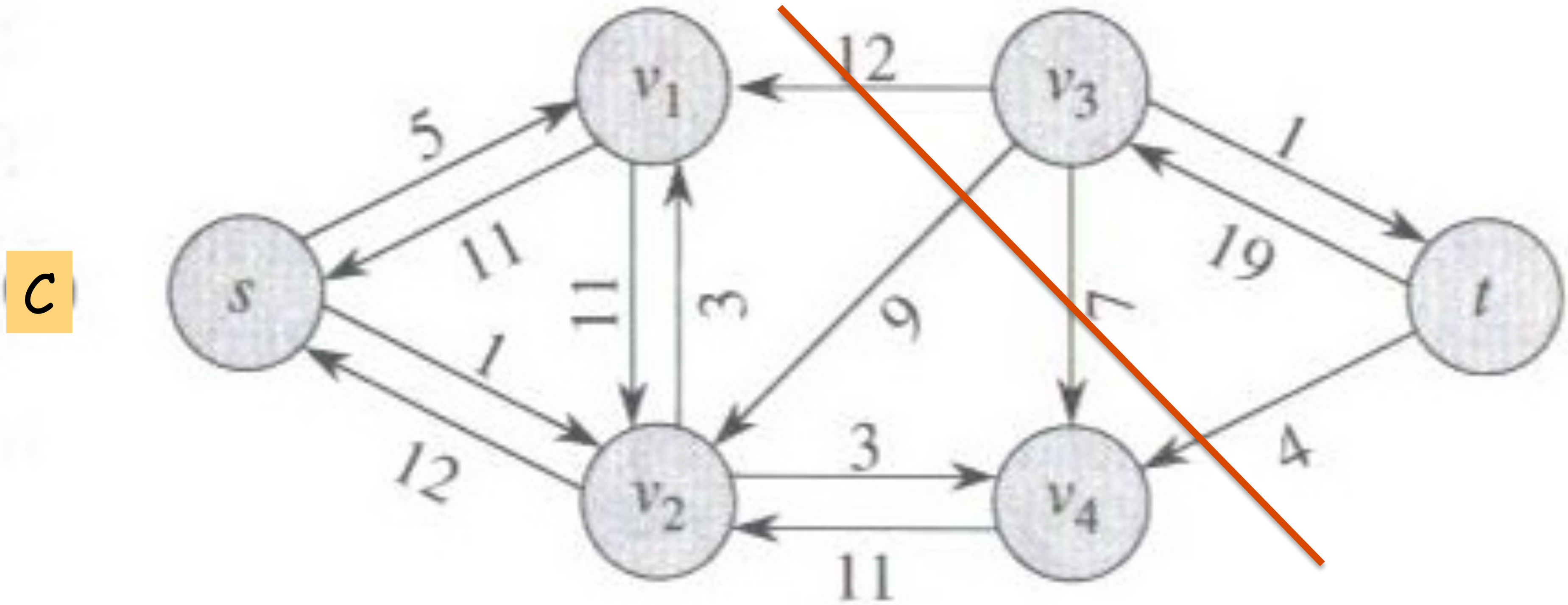




D



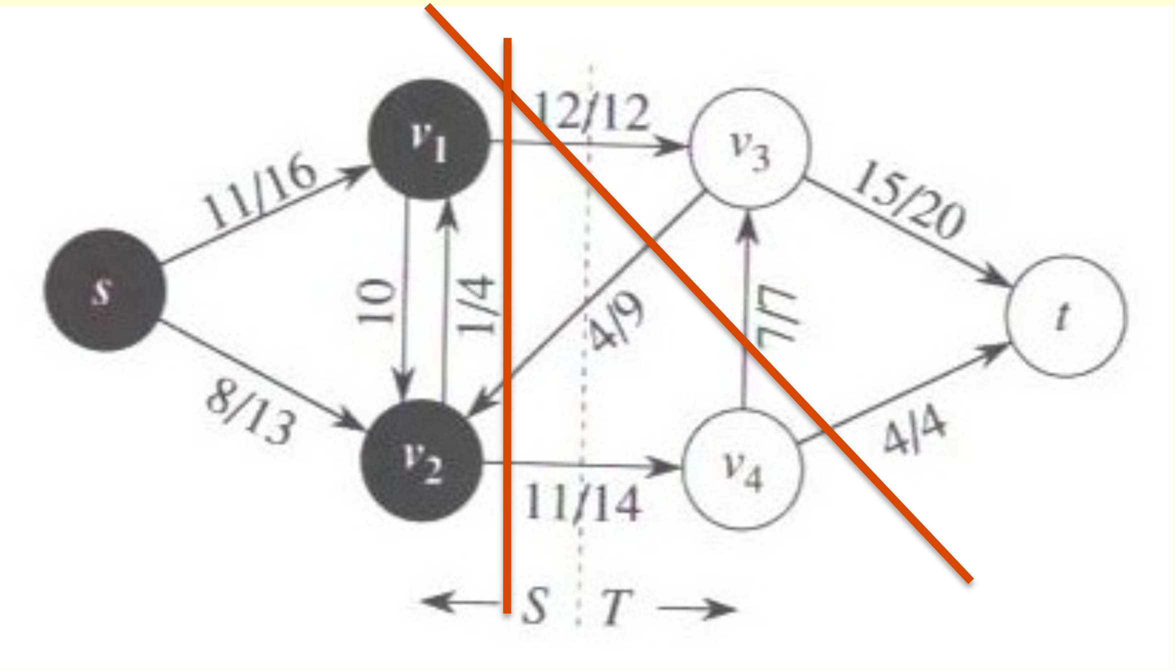
# Ford-Fulkerson Method: Example




- Max-Flow has been reached. Why?
- Cut with zero capacity has been found. Which Cut?
  - $(\{s,v_1,v_2,v_4\},\{v_3,t\})$

# Correctness of Ford-Fulkerson Method

- Augmentation is possible if
  - Every cut-set is NOT saturated



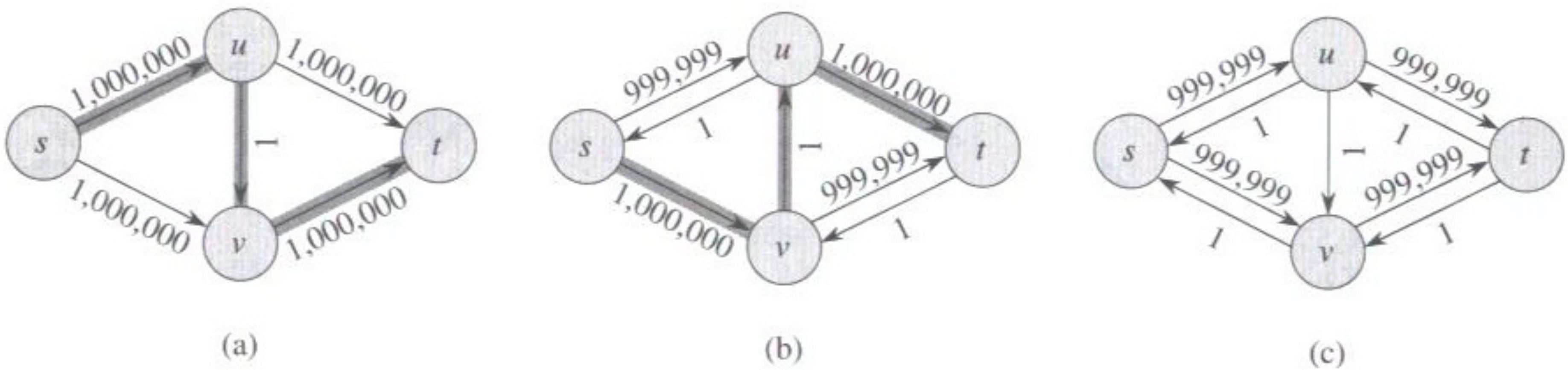

Cut (S,T):
- Capacity = 26
- Flow across cut = 19

Cut (S',T'):
- Capacity = 23
- Flow across cut = 19

- Theorem: Min-Cut = Max-Flow

# Time Complexity

- It can be arbitrarily large.




- Solution: When finding augmenting path, find the shortest path
- In that case, # of augmentations = O(mn)

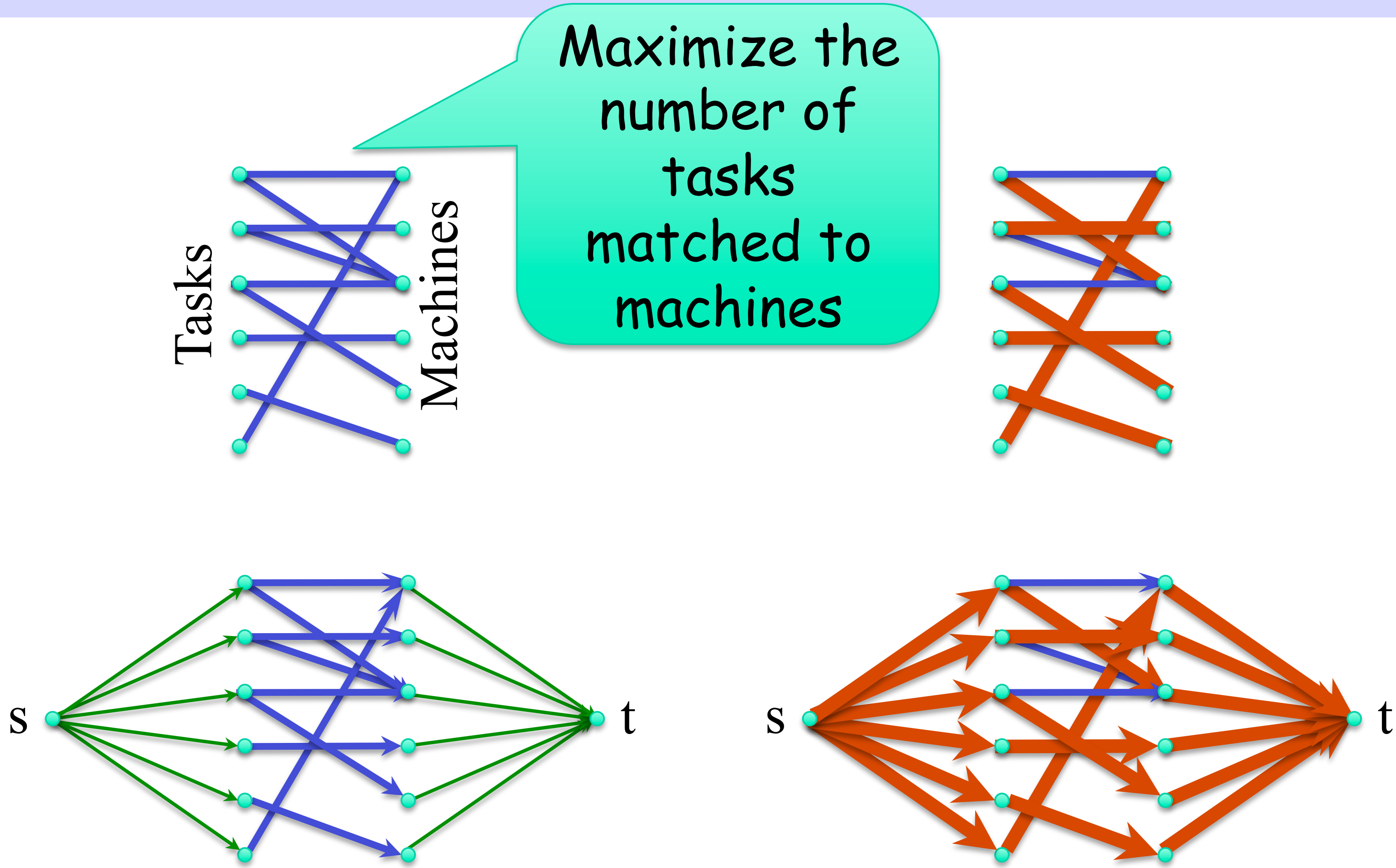
# More efficient Network Flow algorithms

- Push-relabel algorithms [Goldberg, '87]
  - Local algorithm, works on one vertex at a time
  - Avoids maintaining flow conservation rule
    - Excess flow in each node
    - Height function
  - $O(mn^2)$ time complexity
  - Can be improved to $O(n^3)$

# Generalizations

- Multiple sources and sinks.
  - Can be reduced to single source and sink

# Bipartite Matching



# Network Flow

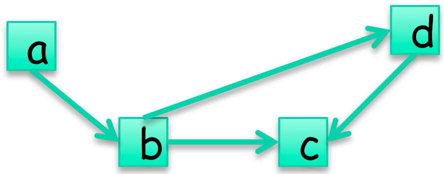
- Input: Directed graph G(V,E) with capacity function on edges given by non-negative function c: E(G) → $\mathcal{R}^+$.
  - Capacity of each edge, e, is given by c(e)
  - Source vertex s
  - Sink vertex t
- Question: Find a flow function f with the maximum flow value

# Min-Cost Network Flow

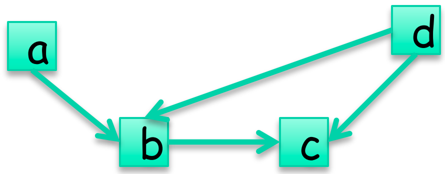
- Input: Directed graph G(V,E) with capacity function on edges given by non-negative function c: E(G) → $\mathcal{R}^+$.
  - Capacity of each edge, e, is given by c(e)
  - Flow cost of each edge, e, is given by a(e)
    - Implies that cost of flow in e is a(e)•f(e)
    - Total cost of flow = Σ a(e)•f(e)
  - Source vertex s
  - Sink vertex t
  - Flow required = F
- Question: Find **min-cost** flow function f with flow value = F

# Minimum Path Cover in DAGs

- **Path Cover**: set of vertex disjoint paths that cover all vertices
- **Minimum Path Cover** in directed acyclic graphs can be reduced to **network flow** (?)
- Examples:




Can be covered with one path: a ➔b ➔d ➔c




Cannot be covered with one path; needs at least two paths to cover all vertices

# COT 6936: Topics in Algorithms

## Linear Programming

# Gaussian Elimination

- Solving a system of simultaneous equations

$$
\begin{aligned}
x_1 \quad\quad -2x_3 \quad\quad &= 2 \\
x_2 + x_3 \quad\quad &= 3 \\
x_1 + x_2 \quad\quad - x_4 &= 4 \\
x_2 + 3x_3 + x_4 &= 5
\end{aligned}
$$

O(n[3]) algorithm

---

$$
\begin{aligned}
x_1 \quad\quad -2x_3 \quad\quad &= 2 \\
x_2 + x_3 \quad\quad &= 3 \\
x_2 + 2x_3 - x_4 &= 2 \\
x_2 + 3x_3 + x_4 &= 5
\end{aligned}
$$

# Linear Programming

- Want more than solving simultaneous equations
- We have an objective function to optimize

# Chocolate Shop [DPV book]

- 2 kinds of chocolate
  - milk [Profit: $1 per box] [Demand: 200]
  - Deluxe [Profit: $6 per box] [Demand: 300]
- Production capacity: 400 boxes
- Goal: maximize profit
  - Maximize $x_1 + 6x_2$ subject to constraints:
    - $x_1 \le 200$
    - $x_2 \le 300$
    - $x_1 + x_2 \le 400$
    - $x_1, x_2 \ge 0$

# Diet Problem

- Food type: $F_1,\ldots,F_m$
- Nutrients: $N_1,\ldots,N_n$
- Min daily requirement of nutrients: $c_1,\ldots,c_n$
- Price per unit of food: $b_1,\ldots,b_m$
- Nutrient $N_j$ in food $F_i$: $a_{ij}$
- Problem: Supply daily nutrients at minimum cost
    - Min $\Sigma_i\, b_i x_i$
    - $\Sigma_i\, a_{ij} x_i \geq c_j$ for $1 \leq j \leq n$
    - $x_i \geq 0$

# Transportation Problem

- Ports or Production Units: $P_1, \ldots, P_m$
- Markets to be shipped to: $M_1, \ldots, M_n$
- Min daily market need: $r_1, \ldots, r_n$
- Port/production capacity: $s_1, \ldots, s_m$
- Cost of transporting to $M_j$ from port $P_i$: $a_{ij}$
- Problem: Meet market need at minimum transportation cost

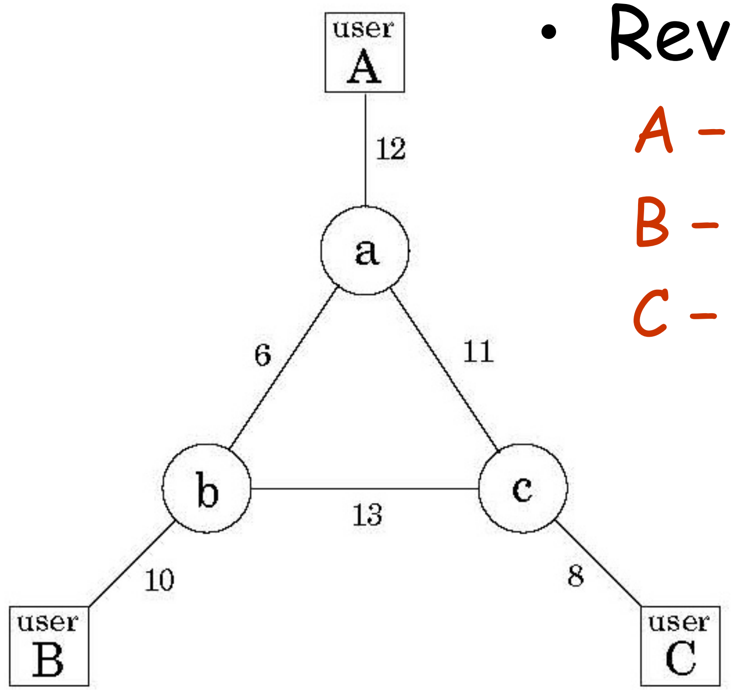
# Assignment Problem

- Workers: $b_1, \ldots, b_n$
- Jobs: $g_1, \ldots, g_m$
- Value of assigning person $b_i$ to job $g_j$: $a_{ij}$
- Problem: Choose job assignment to maximize value

# Bandwidth Allocation Problem

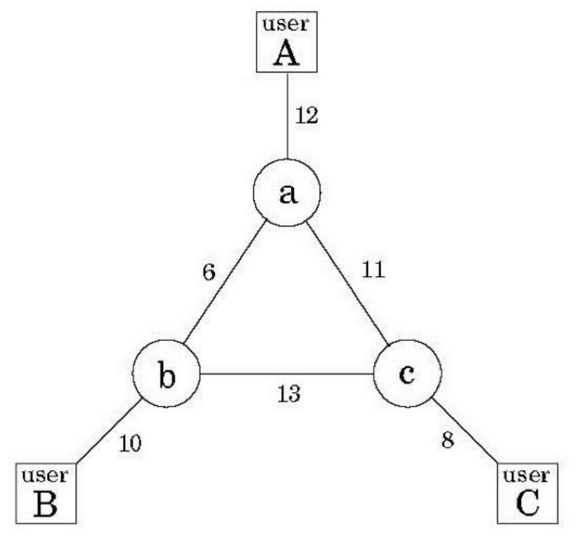
**Figure 7.3** A communications network between three users $A, B$, and $C$. Bandwidths are shown.

- Need:
  - A – B ≥ 2 units
  - B – C ≥ 2 units
  - C – A ≥ 2 units
- Connections:
  - Short route
  - Long route




- Revenue:
  - A – B pays \$3 per unit
  - B – C pays \$2 per unit
  - C – A pays \$4 per unit

# Bandwidth Allocation Pr




- Maximize revenue by allocating connections along two routes wi exceeding bandwidth capacities
- Max $3(x_{AB}+x_{AB}') + 2(x_{BC}+x_{BC}') + 4(x_{AC}+x_{AC}')$ s.t.

$x_{AB} + x_{AB}' + x_{BC} + x_{BC}' \le 10$

$x_{AB} + x_{AB}' + x_{AC} + x_{AC}' \le 12$

$x_{BC} + x_{BC}' + x_{AC} + x_{AC}' \le 8$

$x_{AB} + x_{BC}' + x_{AC}' \le 6$; $x_{AB} + x_{AB}' \ge 2$; $x_{BC} + x_{BC}' \ge 2$

$x_{AB}' + x_{BC} + x_{AC}' \le 13$; $x_{AC} + x_{AC}' \ge 2$

$x_{AB}' + x_{BC}' + x_{AC} \le 11$; & all nonneg constraints

# Standard LP

- Maximize $\Sigma c_j x_j$ [Objective Function]

  Subject to $\Sigma a_{ij} x_j \leq b_j$ [Constraints]

  and $x_j \geq 0$ [Nonnegativity Constraints]

- Matrix formulation of LP

  Maximize $c^T x$

  Subject to $Ax \leq b$

  and $x \geq 0$

# Converting to standard form

- Min $-2x_1 + 3x_2$ Subject to

  $x_1 + x_2 = 7$

  $x_1 - 2x_2 \leq 4$

  $x_1 \geq 0$

- Max $2x_1 - 3x_2$ Subject to

  $x_1 + x_2 \leq 7$

  $-x_1 - x_2 \leq -7$

  $x_1 - 2x_2 \leq 4$

  $x_1 \geq 0$

# Converting to standard form

- Max $2x_1 - 3x_2$ Subject to

  $x_1 + x_2 \leq 7$

  $-x_1 - x_2 \leq -7$

  $x_1 - 2x_2 \leq 4$

  $x_1 \geq 0$

> $x_2$ is not constrained to be non-negative

- Max $2x_1 - 3(x_3 - x_4)$ Subject to

  $x_1 + x_3 - x_4 \leq 7$

  $-x_1 - (x_3 - x_4) \leq -7$

  $x_1 - 2(x_3 - x_4) \leq 4$

  $x_1, x_3, x_4 \geq 0$

# Converting to Standard form

- Max $2x_1 - 3x_2 + 3x_3$ Subject to

  $x_1 + x_2 - x_3 \leq 7$

  $-x_1 - x_2 + x_3 \leq -7$

  $x_1 - 2x_2 - 2x_3 \leq 4$

  $x_1, x_2, x_3 \geq 0$

# Slack Form

- Max $2x_1 - 3x_2 + 3x_3$ Subject to

  $x_1 + x_2 - x_3 \leq 7$

  $-x_1 - x_2 + x_3 \leq -7$

  $x_1 - 2x_2 - 2x_3 \leq 4$

  $x_1, x_2, x_3 \geq 0$
- Max $2x_1 - 3x_2 + 3x_3$ Subject to

  $x_1 + x_2 - x_3 + x_4 = 7$

  $-x_1 - x_2 + x_3 + x_5 = -7$

  $x_1 - 2x_2 - 2x_3 + x_6 = 4$

  $x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$

# Duality

- Max $c^T x$ [Primal]
  Subject to $Ax \leq b$
  and $x \geq 0$

- Min $y^T b$ [Dual]
  Subject to $y^T A \geq c^T$
  and $y \geq 0$

# Understanding Duality

- Maximize $x_1 + 6x_2$ subject to constraints:
  - $x_1 \leq 200$ (1)
  - $x_2 \leq 300$ (2)
  - $x_1 + x_2 \leq 400$ (3)
  - $x_1, x_2 \geq 0$

How were mutipliers determined?

- (100,300) is feasible; value = 1900. Optimum?
- Adding 1 times (1) + 6 times (2) gives us
  - $x_1 + 6x_2 \leq 2000$
- Adding 1 times (3) + 5 times (2) gives us
  - $x_1 + 6x_2 \leq 1900$
  - "Certificate of Optimality" for solution (100,300)

# Understanding Duality

- Maximize $x_1 + 6x_2$ subject to:
  - $x_1 \leq 200$ $(y_1)$
  - $x_2 \leq 300$ $(y_2)$ [(100,300)]
  - $x_1 + x_2 \leq 400$ $(y_3)$
  - $x_1, x_2 \geq 0$
- Different choice of multipliers gives us different bounds. We want smallest bound.
- Minimize $200y_1 + 300y_2 + 400y_3$ subject to:
  - $y_1 + y_3 \geq 1$ $(x_1)$
  - $y_2 + y_3 \geq 6$ $(x_2)$ [(0,5,1)]
  - $y_1, y_2 \geq 0$

# Duality Principle

- Primal feasible values $\leq$ dual feasible values
- Max primal value = min dual value
- Duality Theorem: If a linear program has a bounded optimal value then so does its dual and the two optimal values are equal.

# Visualizing Duality

- Shortest Path Problem
  - Build a physical model and between each pair of vertices attach a string of appropriate length
  - To find shortest path from s to t, hold the two vertices and pull them apart as much as possible without breaking the strings
  - This is exactly what a dual LP solves!
    - Max $x_s - x_t$
    - subject to $|x_u - x_v| \leq w_{uv}$ for every edge (u.v)
  - The taut strings correspond to the shortest path, i.e., they have no slack

# Simplex Algorithm

- Start at v, any vertex of feasible region
- while (there is neighbor v' of v with better objective value) do

  set v = v'
- Report v as optimal point and its value as optimal value

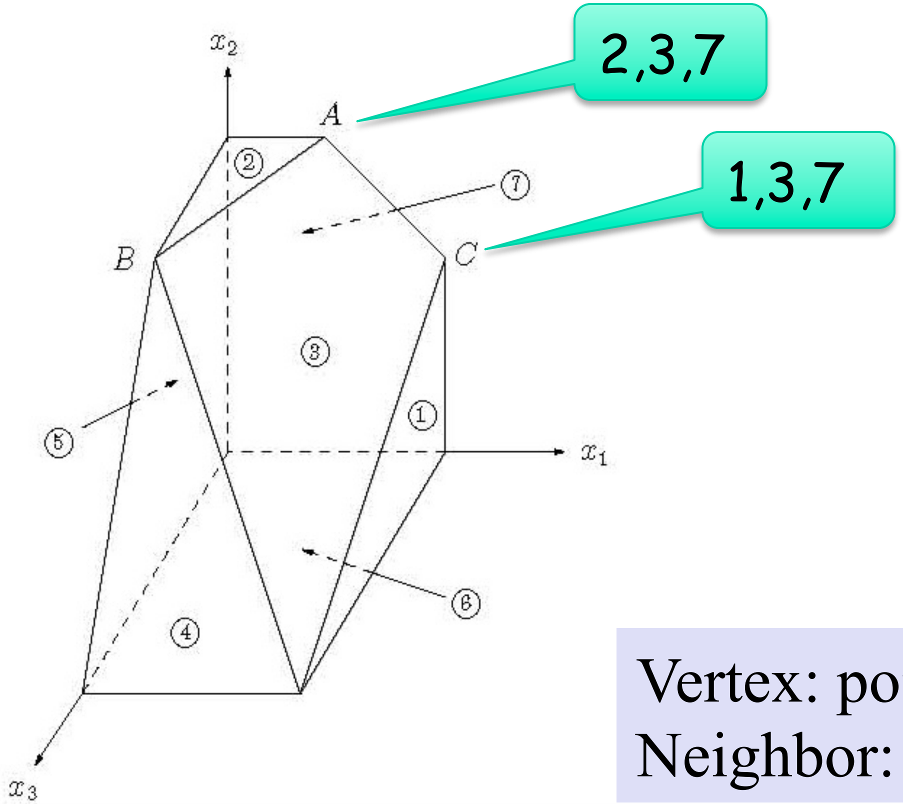
---

- What is a
  - Vertex?, neighbor?
- Start vertex? How to pick next neighbor?

# Simplex Algorithm: Example

i.e., some inequalities satisfied as equalities

**Figure 7.12** A polyhedron defined by seven inequalities.




$$\max \quad x_1 + 6x_2 + 13x_3$$

$$x_1 \le 200 \quad (1)$$

$$x_2 \le 300 \quad (2)$$

$$x_1 + x_2 + x_3 \le 400 \quad (3)$$

$$x_2 + 3x_3 \le 600 \quad (4)$$

$$x_1 \ge 0 \quad (5)$$

$$x_2 \ge 0 \quad (6)$$

$$x_3 \ge 0 \quad (7)$$

Vertex: point where n hyperplanes meet;
Neighbor: vertices sharing n-1 hyperplanes

# Steps of Simplex Algorithm

- In order to find next neighbor from arbitrary vertex, we do a change of origin (pivot)

Initial LP:

$$\max\ 2x_1 + 5x_2$$

$$\begin{aligned} 2x_1 - x_2 &\leq 4 & \text{①} \\ x_1 + 2x_2 &\leq 9 & \text{②} \\ -x_1 + x_2 &\leq 3 & \text{③} \\ x_1 &\geq 0 & \text{④} \\ x_2 &\geq 0 & \text{⑤} \end{aligned}$$

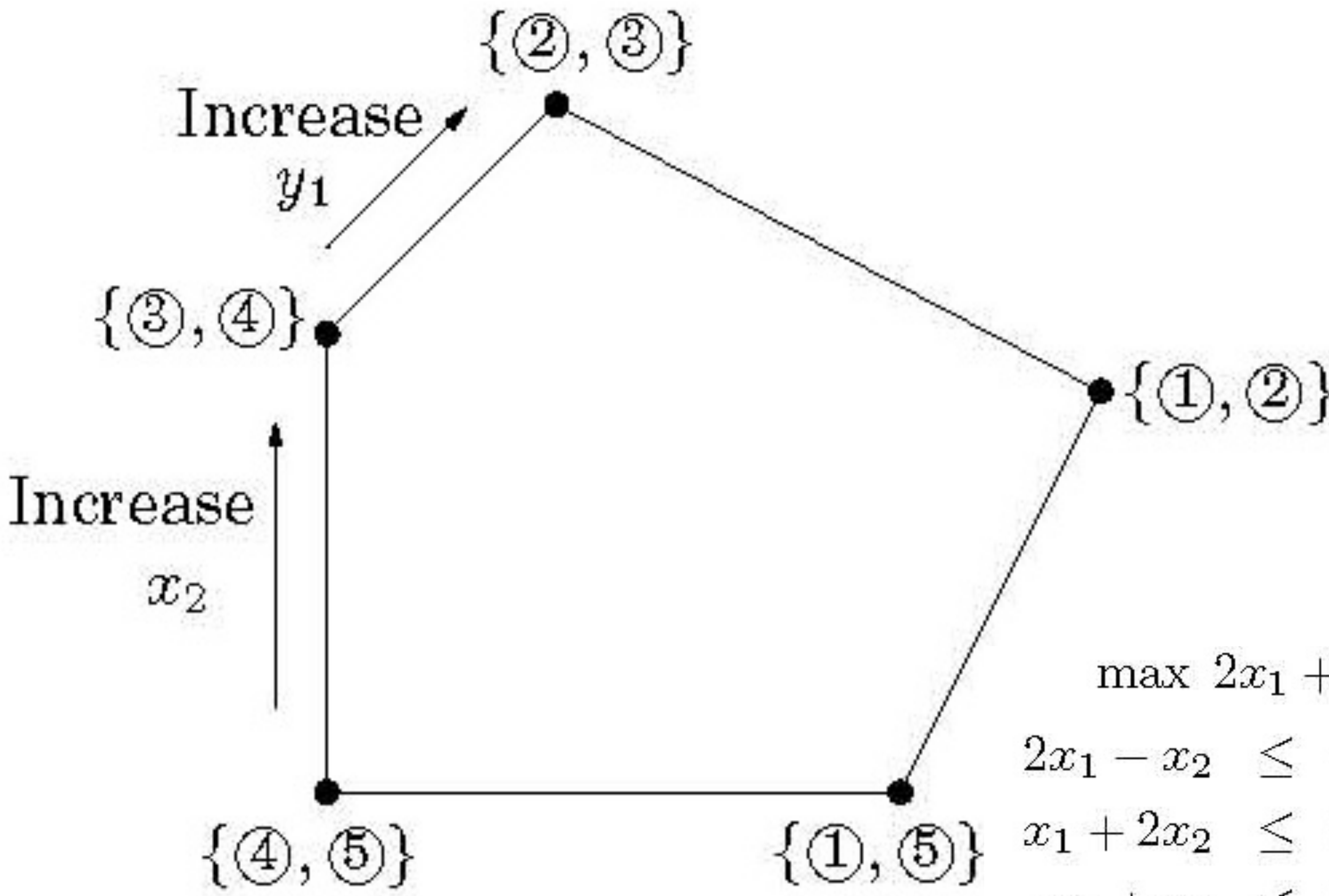
*Current vertex:* {④, ⑤} (origin).
*Objective value:* 0.

*Move:* increase $x_2$.
⑤ is released, ③ becomes tight. Stop at $x_2 = 3$.

New vertex {④, ③} has local coordinates $(y_1, y_2)$:

$$y_1 = x_1, \quad y_2 = 3 + x_1 - x_2$$

# Simplex Algorithm Example




$$\max \; 2x_1 + 5x_2$$

$$\begin{array}{rcll}
2x_1 - x_2 & \leq & 4 & ① \\
x_1 + 2x_2 & \leq & 9 & ② \\
-x_1 + x_2 & \leq & 3 & ③ \\
x_1 & \geq & 0 & ④ \\
x_2 & \geq & 0 & ⑤
\end{array}$$

# Simplex Algorithm Example

Initial LP:

$$\max\ 2x_1 + 5x_2$$

$$\begin{aligned} 2x_1 - x_2 &\leq 4 && ① \\ x_1 + 2x_2 &\leq 9 && ② \\ -x_1 + x_2 &\leq 3 && ③ \\ x_1 &\geq 0 && ④ \\ x_2 &\geq 0 && ⑤ \end{aligned}$$

*Current vertex:* {④, ⑤} (origin).
*Objective value:* 0.

*Move:* increase $x_2$.
⑤ is released, ③ becomes tight. Stop at $x_2 = 3$.

New vertex {④, ③} has local coordinates $(y_1, y_2)$:

$$y_1 = x_1, \quad y_2 = 3 + x_1 - x_2$$

Rewritten LP:

$$\max\ 15 + 7y_1 - 5y_2$$

$$\begin{aligned} y_1 + y_2 &\leq 7 && ① \\ 3y_1 - 2y_2 &\leq 3 && ② \\ y_2 &\geq 0 && ③ \\ y_1 &\geq 0 && ④ \\ -y_1 + y_2 &\leq 3 && ⑤ \end{aligned}$$

*Current vertex:* {④, ③}.
*Objective value:* 15.

*Move:* increase $y_1$.
④ is released, ② becomes tight. Stop at $y_1 = 1$.

New vertex {②, ③} has local coordinates $(z_1, z_2)$:

$$z_1 = 3 - 3y_1 + 2y_2, \quad z_2 = y_2$$

# Simplex Algorithm Example

Rewritten LP:

$$\max\ 15 + 7y_1 - 5y_2$$

$$\begin{aligned} y_1 + y_2 &\leq 7 & \text{①} \\ 3y_1 - 2y_2 &\leq 3 & \text{②} \\ y_2 &\geq 0 & \text{③} \\ y_1 &\geq 0 & \text{④} \\ -y_1 + y_2 &\leq 3 & \text{⑤} \end{aligned}$$

*Current vertex:* {④, ③}.
*Objective value:* 15.

*Move:* increase $y_1$.
④ is released, ② becomes tight. Stop at $y_1 = 1$.

New vertex {②, ③} has local coordinates $(z_1, z_2)$:

$$z_1 = 3 - 3y_1 + 2y_2, \quad z_2 = y_2$$

Rewritten LP:

$$\max\ 22 - \tfrac{7}{3}z_1 - \tfrac{1}{3}z_2$$

$$\begin{aligned} -\tfrac{1}{3}z_1 + \tfrac{5}{3}z_2 &\leq 6 & \text{①} \\ z_1 &\geq 0 & \text{②} \\ z_2 &\geq 0 & \text{③} \\ \tfrac{1}{3}z_1 - \tfrac{2}{3}z_2 &\leq 1 & \text{④} \\ \tfrac{1}{3}z_1 + \tfrac{1}{3}z_2 &\leq 4 & \text{⑤} \end{aligned}$$

*Current vertex:* {②, ③}.
*Objective value:* 22.

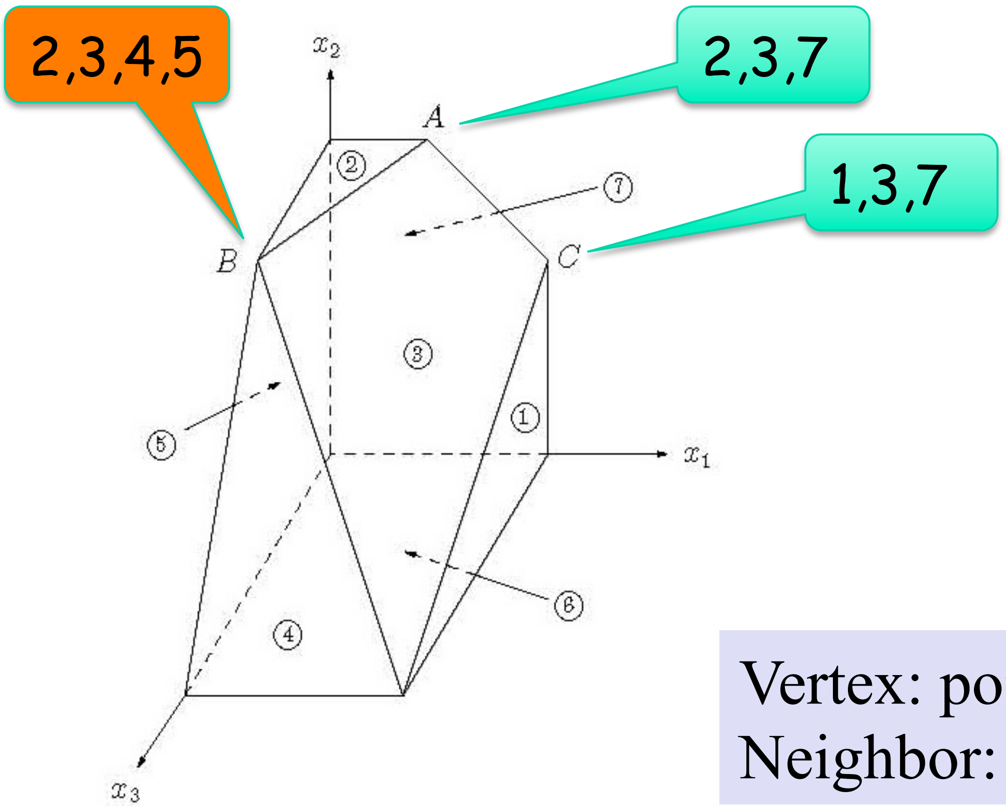
*Optimal:* all $c_i < 0$.

Solve ②, ③ (in original LP) to get optimal solution $(x_1, x_2) = (1, 4)$.

# Simplex Algorithm: Degenerate vertices

i.e., some inequalities satisfied as equalities

**Figure 7.12** A polyhedron defined by seven inequalities.




$$
\begin{array}{lrr}
\max & x_1 + 6x_2 + 13x_3 & \\
 & x_1 \le 200 & ① \\
 & x_2 \le 300 & ② \\
 & x_1 + x_2 + x_3 \le 400 & ③ \\
 & x_2 + 3x_3 \le 600 & ④ \\
 & x_1 \ge 0 & ⑤ \\
 & x_2 \ge 0 & ⑥ \\
 & x_3 \ge 0 & ⑦
\end{array}
$$

Vertex: point where n hyperplanes meet;
Neighbor: vertices sharing n-1 hyperplanes

# Polynomial-time algorithms for LP

- Simplex is not poly-time in the worst-case
- Khachiyan's ellipsoid algorithm: LP is in $\mathcal{P}$
- Karmarkar's interior-point algorithm
- Good implementations for LP exist
  - Works very well in practice
  - More competitive than the poly-time methods for LP

# Integer Linear Programming

- LP with integral solutions
- NP-hard
- If *A* is a totally unimodular matrix, then the LP solution is always integral.
  - *A* TUM is a matrix for which every nonsingular submatrix has determinant 0, +1 or -1.
  - *A* TUM is a matrix for which every nonsingular submatrix has integral inverse.

# Vertex Cover as an LP?

- For vertex v, create variable $x_v$
- For edge (u,v), create constraint $x_u + x_v \geq 1$
- Objective function: $\Sigma x_v$
- Additional constraints: $x_v \leq 1$

- Doesn't work because $x_v$ needs to be from {0,1}