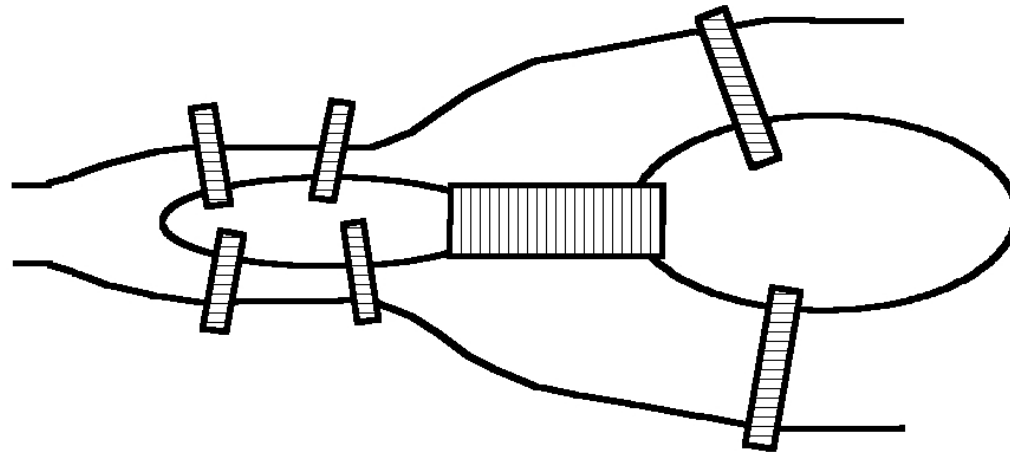


# Networks

---

- People have been thinking about network problems for a long time
- Koenigsberg Bridge problem (Euler, 1736)



The Königsberg Bridge Problem

- Can you cross all 7 bridges exactly once on a walk?

# Types of Network Flow Problems (Winston)

---

- **Transportation problem (Hitchcock, 1941)**
  - Given a set of sources, destinations
  - Have source capacities, destination demands
  - Know shipping cost/unit for each source-destination pair
  - Minimize total cost of meeting demands at destinations
- **Assignment Problem**
  - Restricted transportation problem
  - Each source can supply 1 object
  - Each destination demands 1 object
- **Transshipment Problem**
  - Transportation problem with intermediate (transshipment) points
  - Generalizes both transportation and assignment problem

# Other Network Models in Winston

---

- Shortest path problem (Dijkstra 1959)
  - Find the shortest route between an origin and a destination
  - Algorithms range from simple “greedy” algorithm to very complex
  - Special case of the transshipment problem
- Maximum flow problem
  - Maximize flows from a single source to a single destination
  - Important dual result: finds the minimum “cut” in a network
- Minimum spanning tree
  - Objective is to connect all nodes in a network
  - Want to minimize the total length of the connections
- Project management (PERT-CPM)
  - Find time to complete a linked set of tasks
  - Is actually a “longest path” problem

# Some Network Models *Not* in Winston

---

- Circulation problem
  - Transshipment problem with no source or demand nodes
  - Example: airline routing schedules
- Generalized flow problem
  - Some of the material flowing on the network is lost in transmission
  - Example: electrical power transmission, water distribution
- Multicommodity flow problem
  - More than one type of material is flowing on the network
  - Different materials consume network capacity, but may have different transmission costs
- Network interdiction problem
  - Bad guys are moving on a network; good guys try to stop them
  - Each side has to choose what paths to use or interdict

# Network Models *Not* in Winston (cont'd)

---

- Network reliability problem
  - Find the maximum reliability set of routes in network
  - In certain cases, can be recast as an MCNFP
- Network models with “side constraints”
  - Knapsack problem: optimize the “goodness” of a set of objects that must fit into a finite-sized “knapsack”
  - Traveling Salesman Problem: find the minimum-cost tour among a set of destinations, but only visit each destination once
  - Various vehicle routing problems

- 
- Network optimization literature is gigantic
    - Seminal text is Ahuja, Magnanti, and Orlin (1993); 846 pages!
    - Huge number of applications
    - Inspiration for much of the research into algorithmic efficiency

# Concentration in This Course

---

- I will emphasize the transshipment problem
- Otherwise known as the minimum-cost network flow problem (MCNFP)
- Reasons:
  - Transportation, assignment, max flow, shortest path problems are all cases of MCNFP
  - All commercial software implements a modified simplex method based on MCNFP
  - To exploit this capability, you have to formulate in terms of MCNFP
  - Specialized transportation and assignment problem algorithms are largely unnecessary (e.g., stepping-stone and Hungarian methods)

# Network Jargon

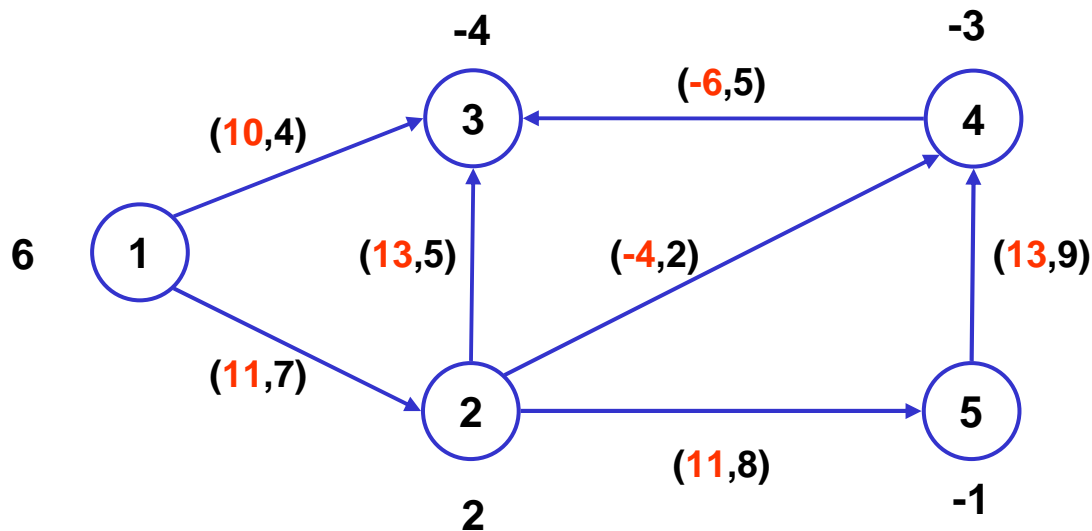
---

- To model a problem as a network, we need some terminology
- A graph  $\mathbf{G}$  consists of:
  - A set of nodes ( $\mathbf{N}$ )
  - A set of arcs ( $\mathbf{A}$ ), which connect the nodes
  - So,  $\mathbf{G} = (\mathbf{N}, \mathbf{A})$  specifies the “topology” of the network
- Graph characteristics
  - Let  $i, j$  be indices for the nodes
  - Then the pair  $(i, j)$  identifies an arc
  - This notation allows us to define things such as:
    - Unit transportation costs on an arc ( $\mathbf{C}_{ij}$ )
    - Supplies at each source node ( $\mathbf{S}_i$ )
    - Demands at each demand node ( $\mathbf{D}_j$ )

# From Network to Optimization Model

---

- Consider the transshipment network below:



- What does all this mean?
  - Numbers in circles are node labels
  - Numbers next to nodes are *supplies* ( $>0$ ) and *demands* ( $<0$ )
  - Numbers on arcs are (**cost/unit to ship**, max flow on arc)
- Assume we want to minimize total cost



# Optimization Model (Winston format)

---

- Let  $x_{ij}$  be the flow on arc  $(i,j)$
- Then, the model is:

$$\min z = 11x_{12} + 10x_{13} + 13x_{23} - 4x_{24} + 11x_{25} - 6x_{43} + 13x_{54}$$

subject to

$$\begin{array}{rcl} x_{12} + x_{13} & & = 6 \text{ (node 1 balance)} \\ -x_{12} + x_{23} + x_{24} + x_{25} & & = 2 \text{ (node 2 balance)} \\ -x_{13} - x_{23} & -x_{43} & = -4 \text{ (node 3 balance)} \\ & -x_{24} + x_{43} - x_{54} & = -3 \text{ (node 4 balance)} \\ & -x_{25} + x_{54} & = -1 \text{ (node 5 balance)} \end{array}$$

$$\text{all } x_{ij} \geq 0$$

$$x_{12} \leq 7, x_{13} \leq 4, x_{23} \leq 5, x_{24} \leq 2, x_{25} \leq 8, x_{43} \leq 5, x_{54} \leq 9$$

# Some Key Points

---

- **FLOW MUST BALANCE!**
  - For pure supply nodes, flow out must equal supply
  - For pure demand nodes, flow in must equal demand
  - For transshipment nodes, flow out must equal flow in
  - Note the example has nodes that demand and transship
- What do we do if supply is unequal to demand?
  - Create dummy node to absorb (ship) excess supply (demand)
  - Create costs that make sense in the model
- Arc costs can be negative
- Can also force flow on arcs by putting in lower bounds
- Sign conventions
  - Flow out is positive
  - Flow in is negative

# Optimization Model (Algebraic format)

---

- Indices
  - $i, j =$  nodes  $\{1,2,3,4,5\}$
- Subsets
  - $ARCS(i,j) =$  connections between nodes
- Data
  - $C_{ij} =$  cost per unit to ship on arc  $i,j$
  - $L_{ij} =$  lower bound on flow on arc  $i,j$
  - $U_{ij} =$  upper bound on flow on arc  $i,j$
  - $SD_i =$  supply or demand at node  $i$
- Variables
  - $x_{ij} =$  flow on arc  $i,j$

# Algebraic Format (cont'd)

---

- So, the general MCNFP is:

$$\min z = \sum_{i,j \in ARCS(i,j)} C_{ij} * x_{ij}$$

subject to

$$\left[ \sum_{j \in ARCS(i,j)} x_{ij} \right] - \left[ \sum_{j \in ARCS(j,i)} x_{ji} \right] = SD_i \text{ for all nodes } i$$

FLOW  
IN

FLOW  
OUT

$$L_{ij} \leq x_{ij} \leq U_{ij} \text{ for all } ARCS(i, j)$$

- One MPL program can represent any MCNFP!
- So, are we done yet?

# Answer is NO

---

- The trick with networks is to translate the problem into a network structure
- Many things that don't appear to be networks are
  - See Winston, pp. 366-368 on the inventory problem
  - Many, many other models can be represented by a network
- So, the challenge is to:
  - Determine if the problem can be represented by a network
  - If so, come up with the nodes, arcs, costs, and capacities
- **Why do we want to do this?**
  - **Speed:** network codes are much faster than normal LP simplex
  - **Integrality:** if a problem can be represented by an MCNFP, it will have integral answers (if the supplies, demands and bounds are integer)

# Common MCNFP Models - Transportation

---

- Transportation problem
  - This is an MCNFP with no transshipment nodes
  - Nodes are divided into two disjoint sets, SOURCES and SINKS
- So, the formulation becomes:

$$\begin{aligned} \min z = & \sum_{i,j \in ARCS(i,j)} C_{ij} * x_{ij} \\ \text{subject to} & \\ & \sum_{j \in ARCS(i,j)} x_{ij} = SUPPLY_i \text{ for all nodes } i \in SOURCES \\ & \sum_{i \in ARCS(i,j)} x_{ij} = DEMAND_j \text{ for all nodes } j \in SINKS \\ & 0 \leq x_{ij} \leq U_{ij} \text{ for all } ARCS(i, j) \end{aligned}$$

- Winston shows a special algorithm for this, but it's unnecessary

# Common MCNFP Models - Assignment

---

- A very simple model (too simple to be of much use)
  - We are matching demands to supplies
  - Each demand (supply) node demands (supplies) 1 unit
  - Number of supply and demand nodes are equal

- So, this model is:

$$\begin{aligned} \min z &= \sum_{i,j \in ARCS(i,j)} C_{ij} * x_{ij} \\ \text{subject to} \\ \sum_{j \in ARCS(i,j)} x_{ij} &= 1 \text{ for all nodes } i \in SOURCES \\ \sum_{i \in ARCS(i,j)} x_{ij} &= 1 \text{ for all nodes } j \in SINKS \\ x_{ij} &\in \{0,1\} \text{ for all } ARCS(i,j) \end{aligned}$$

- Again, Winston shows a special (Hungarian) algorithm for this - it's not needed

# Common MCNFP Problems - Max Flow

---

- This is useful, but is a twist on the MCNFP formulation
  - We are maximizing flow from a single source (s) to a single destination (d)
  - This flow, however, is a variable,  $v$

- This model is:

$$\begin{array}{l} \max z = v \\ \text{subject to} \\ \left[ \sum_{j \in \text{ARCS}(i,j)} x_{ij} \right] - \left[ \sum_{j \in \text{ARCS}(j,i)} x_{ji} \right] = \begin{cases} v & \text{for } i = s \\ 0 & \text{for all nodes } i \neq s, i \neq d \\ -v & \text{for } i = d \end{cases} \\ 0 \leq x_{ij} \leq U_{ij} \text{ for all } \text{ARCS}(i, j) \end{array}$$

- Note here that we have to include a “return arc” from d to s to ensure the flow balances



# Shortest (Longest) Path Problem

---

- This can also be represented by an MCNFP
  - Costs are arc lengths
  - Flow 1 unit from the origin  $s$  to the destination  $d$
  - Minimize (maximize) the sum of the arcs used
- Formulation:

$$\begin{aligned} \min (\text{or max}) \quad z &= \sum_{i,j \in ARCS(i,j)} C_{ij} * x_{ij} \\ \text{subject to} \\ \left[ \sum_{j \in ARCS(i,j)} x_{ij} \right] - \left[ \sum_{j \in ARCS(j,i)} x_{ji} \right] &= \begin{cases} 1, i = s \\ 0, i \neq s \text{ and } i \neq d \\ -1, i = d \end{cases} \\ 0 \leq x_{ij} \leq 1 &\text{ for all } ARCS(i, j) \end{aligned}$$

- NOTE: lots of simple algorithms (e.g., Dijkstra) available for this problem

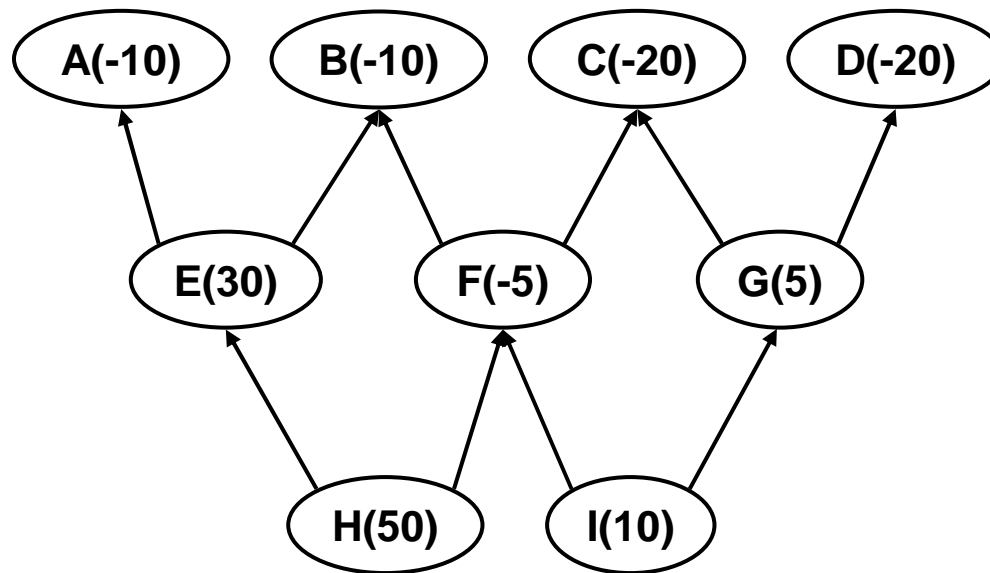
# Converting a Problem to a Network

---

- Many hard problems become easy if you can convert them to a network
- Example: open-pit mining problem
  - An open pit mine can be represented as a set of blocks
  - Each block  $i$  has a net profit  $w_i$  if you choose to extract it
  - But, you have to remove the blocks above it to get to it
- This problem is called a “maximum weight closure”
  - The graph is a set of nodes with weights
  - The arcs show dependencies among nodes
  - A closure is a set of nodes with no outgoing arcs
  - The objective is to find the closure with maximum weight

# Example Closure Problem

---



Number in parentheses  
is payoff for choosing  
that node

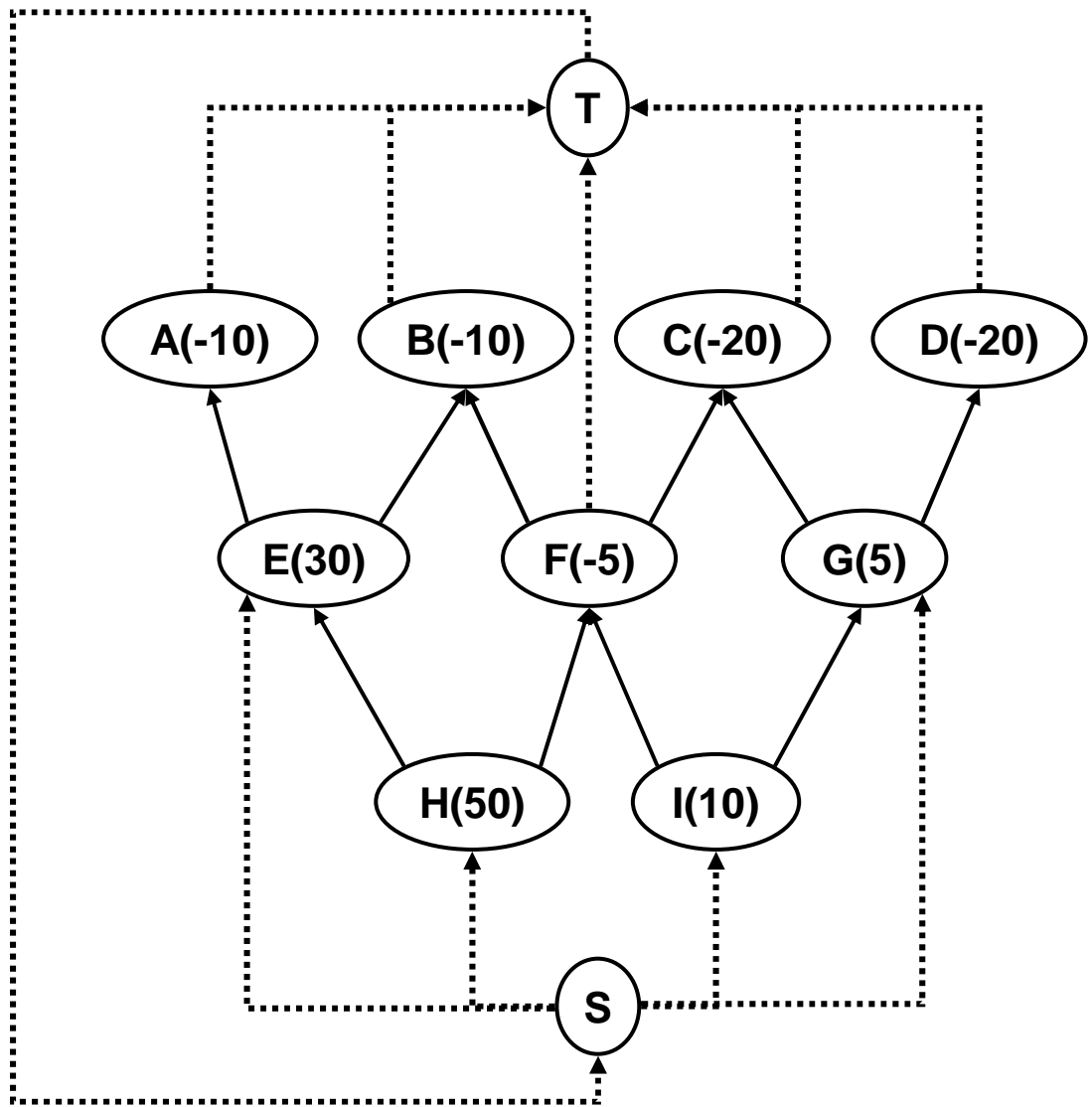
Example: A, B, and  
E is a closure, with  
total weight = 10

# Converting a Closure to a Max Flow Problem

---

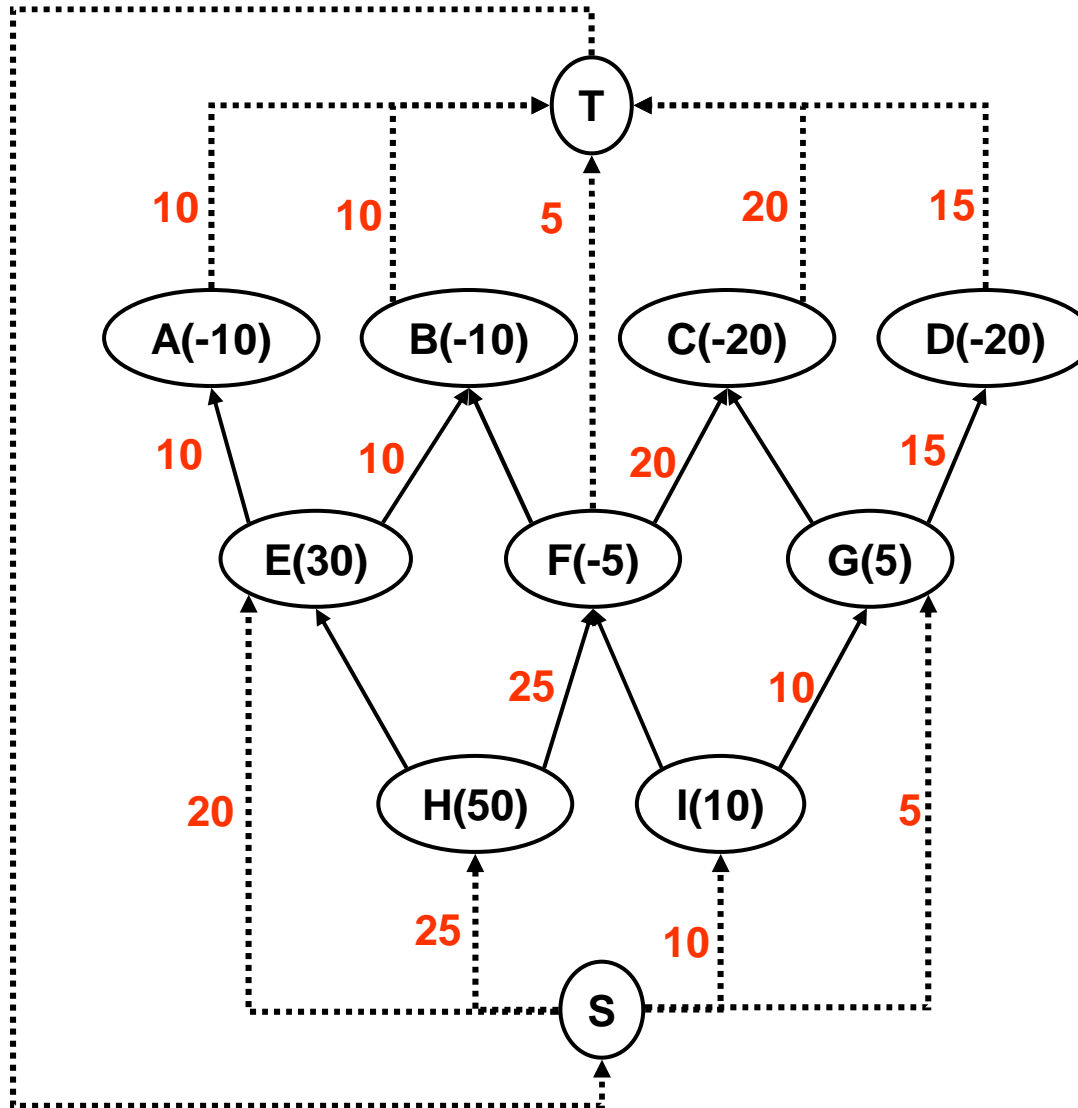
- Add two new nodes,  $s$  and  $t$
- Connect all nodes with positive payoffs with arcs *from* node  $s$
- Connect all nodes with negative payoffs arcs *to* node  $t$
- Make the upper bound on all new arcs the absolute value of the weight of the node
- Make the upper bound on the original arcs infinite
- Solve a maximum flow problem with this network

# The Maximum Weight Closure as a Max Flow



Upper bound on flow for all new arcs is the absolute value of the node they connect to (except for T to S)

# Getting the Answer



Objective function value: **60**

Flows shown on diagram

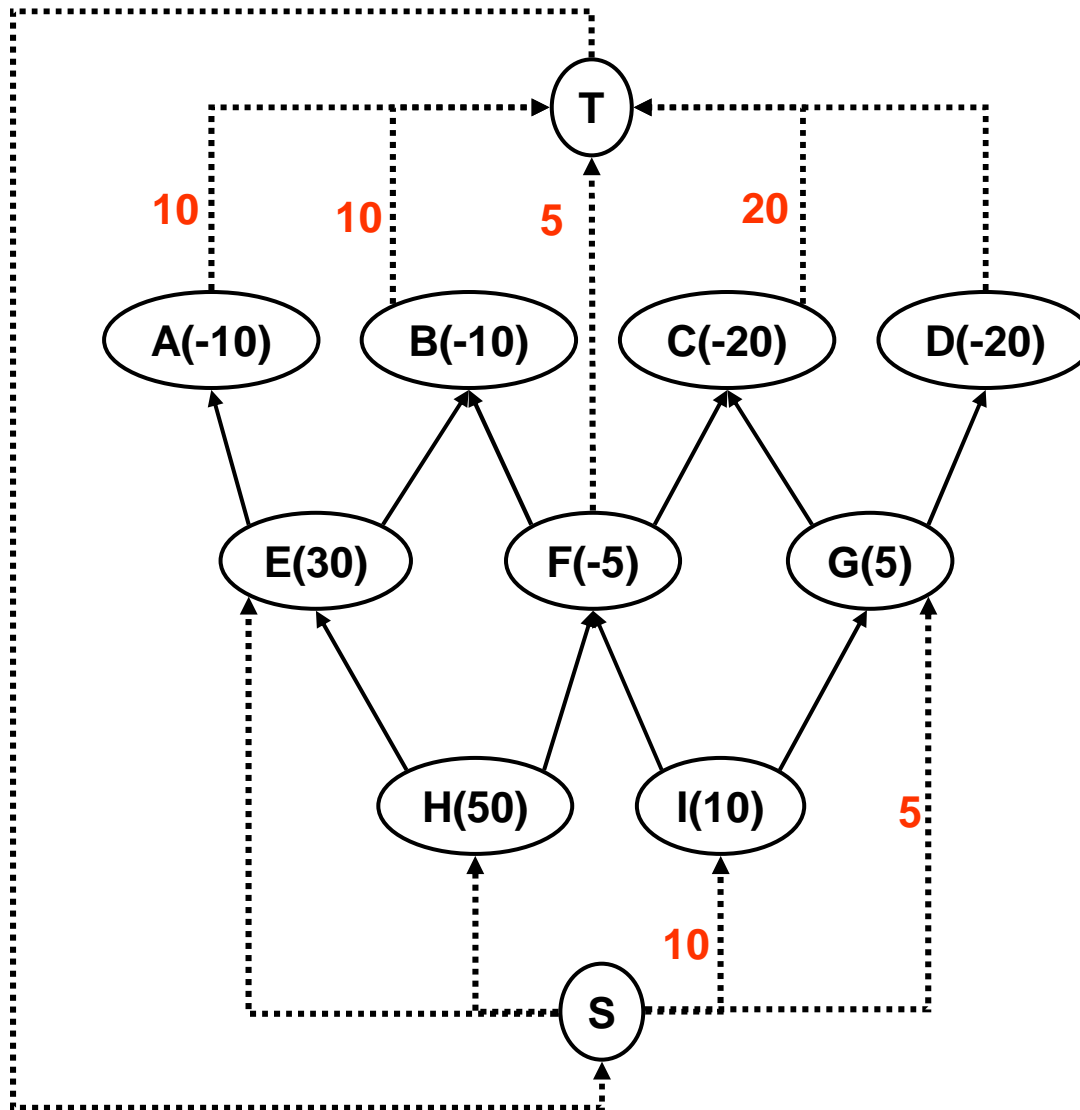
But what nodes are in the closure? And what's the weight?

# Aside: the Max-Flow Min-Cut Theorem

---

- In a maximum flow problem, the optimal flow is equal to the capacity of the minimum “cut”
  - A *cut* is a set of arcs that divides the network into two sets of nodes, one containing the source (S) and the other the sink (T)
  - Call these sets of nodes  $N_1$  and  $N_2$
  - Each arc in the cut set has one endpoint in  $N_1$  and another in  $N_2$
- Consequences:
  - Solving the max flow problem also gives the minimal set of arcs that can “disconnect” the network
  - The arcs in the cut will all be at their upper bounds
  - A large network can have many cutsets
  - May have to resort to a separate algorithm to find them all

# Finding the Min Cut in the Closure Problem



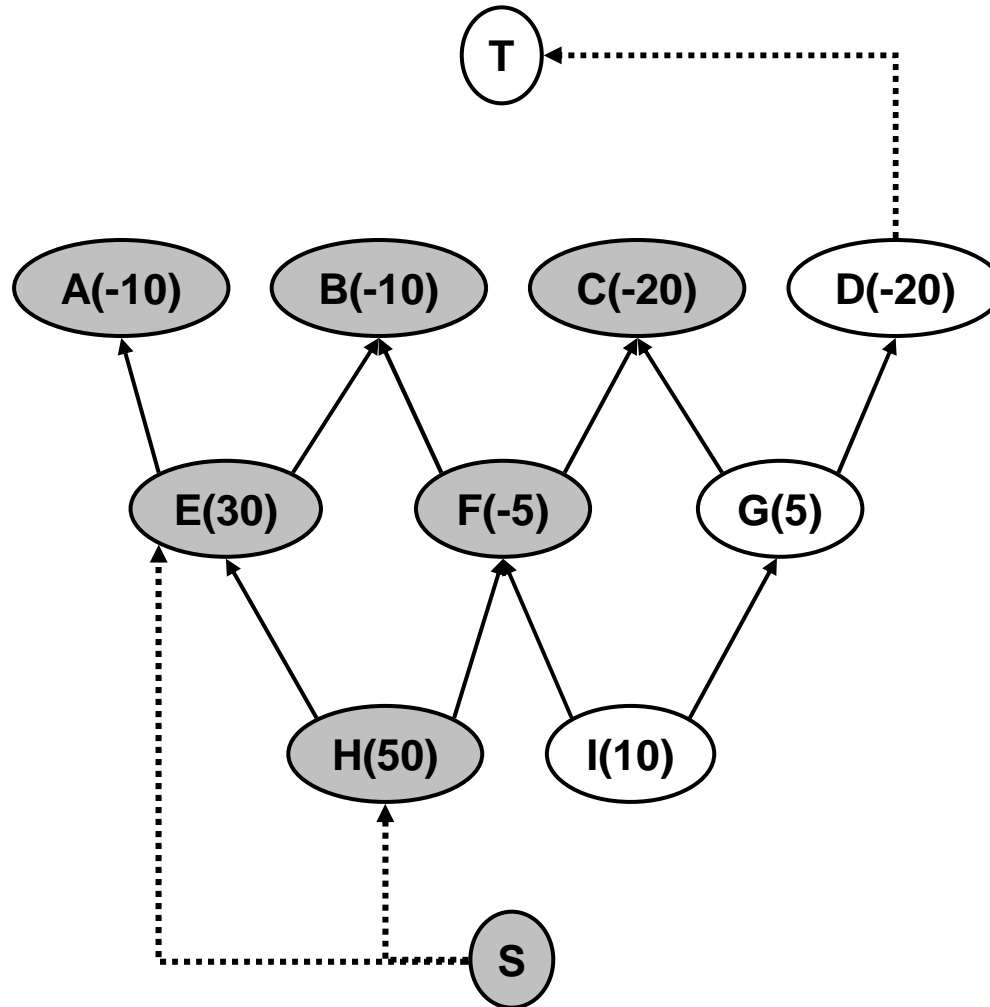
Marked arcs are at their upper bounds

Note that the sum of those bounds is the max flow



# Delete the Arcs in the Min Cut

---



Now we see the partition of the nodes ...

Which partition is the maximum weight closure?  
And what is its weight?

# Some Final Notes

---

- We can solve the max weight closure problem directly:

$$\begin{array}{l} \max z = \sum_i W_i x_i \\ \text{subject to} \\ x_i \leq x_j \text{ for all } i, j \in \text{ARCS}(i, j) \\ x_i \in \{0,1\} \text{ for all } i \end{array}$$

- People convert it to a network because:
  - There are special max flow algorithms available that do not require expensive LP solvers
  - It's relatively easy to code these algorithms and they run quickly
- However, you must do added work to find the solution
- See <http://128.32.125.151/riot/index.html> (the Remote Interactive Optimization Testbed) website for a demo

# The Critical Path Method (CPM)

---

- Recent evolution of project scheduling
  - Methodology depended on who was in charge
  - After WW I, the Gantt (bar) chart became a popular method
  - But, bar charts had limited ability to depict complex relationships
- DuPont and Remington Rand Univac developed a new method in the late 1950's
  - Approach was to depict the project as a network
  - Aim was provide a means to investigate tradeoffs in project cost and duration
  - Came to be known as CPM

# CPM Formulation

---

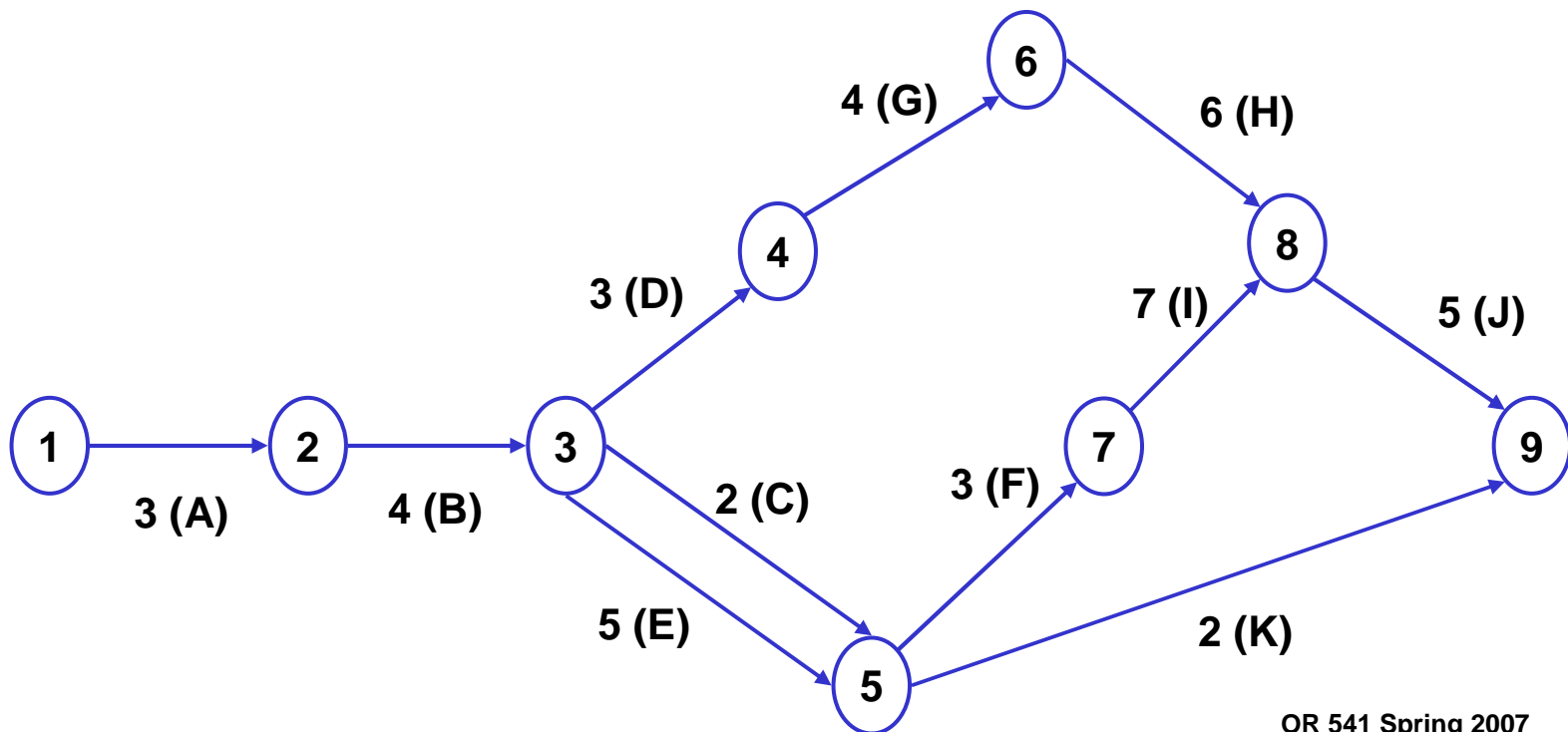
- CPM is essentially a longest-path problem (and can be depicted as an MCNFP)
- Consider the following example (from Schrage):

Activity	Job #	Time	Predecessors
Dig basement	A	3	none
Pour foundation	B	4	A
Pour basement floor	C	2	B
Install floor joists	D	3	B
Install Walls	E	5	B
Install rafters	F	3	C,E
Install flooring	G	4	D
Rough interior	H	6	G
Install roof	I	7	F
Finish interior	J	5	I,H
Landscape	K	2	C,E

# Network Representation (Activity-on-Arc, or AOA)

---

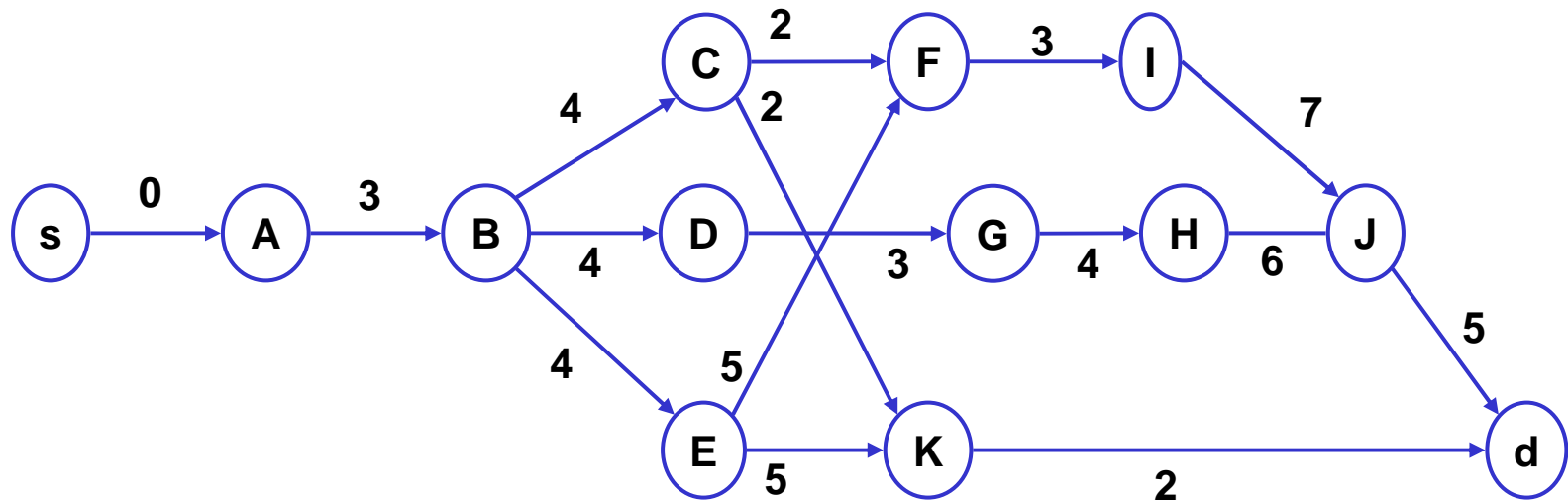
- Nodes represent precedences
- Arcs represent activities and completion times
- Project time is the *longest* path from 1 to 9
- What's the critical path?



## A Better Representation: Activity on Node (AON)

---

- Here is the same problem with the nodes as activities



- This representation is *far superior* to AOA
  - AOA frequently requires dummy arcs to depict precedences
  - Minimizing the number of dummy arcs is a difficult problem
  - We will *not* use AOA representations in this course

# CPM as a Longest Path Problem

---

- Just maximize the shortest path formulation:

$$\begin{aligned} \max z = & \sum_{i,j \in ARCS(i,j)} C_{ij} * x_{ij} \\ \text{subject to} & \\ \left[ \sum_{j \in ARCS(i,j)} x_{ij} \right] - \left[ \sum_{j \in ARCS(j,i)} x_{ji} \right] = & \begin{cases} 1, i = s \\ 0, i \neq s \text{ and } i \neq d \\ -1, i = d \end{cases} \begin{pmatrix} u_s \\ u_i \\ u_d \end{pmatrix} \\ 0 \leq x_{ij} \leq 1 & \text{ for all } ARCS(i, j) \end{aligned}$$

- However, we will work (for now) with the *dual* of this problem
  - The indicies  $i, j$  (with start  $s$  and finish  $d$ ) now represent jobs
  - The variable  $u_i$  is the start time for each job
  - Let  $C_i$  be the completion time of job  $i$

# Dual Formulation

---

- Here's what the dual looks like:

$$\min z = u_d - u_s$$

subject to

$$u_j - u_i \geq C_i \text{ for all } i, j \in ARCS(i, j)$$

$u_i$  unrestricted for all  $i$

- The dual is not a network!
  - The total time is the difference between  $u_d$  and  $u_s$
  - The rest of the constraints enforce precedences, completion times
  - The dual is easier to formulate (and extend) than the MCNFP
- This formulation *does* let us get at what the original researchers wanted to investigate, though ...



# Project Crashing

---

- Addresses trades between expenditures, completion time
- Assume that:
  - You know the cost per unit time to “crash” a job ( $CC_i$ )
  - You know the minimum job completion time ( $MIN_i$ )
  - $TOT$  is the total desired project time
- Formulation, where  $cr_i$  is the amount a job is crashed:

$$\begin{aligned} \min z &= \sum_i CC_i * cr_i \\ \text{subject to} \\ u_d - u_s &\leq TOT \\ u_j &\geq u_i + C_i - cr_i \text{ for all } i, j \in ARCS(i, j) \\ u_i &\text{ unrestricted for all } i \\ 0 &\leq cr_i \leq C_i - MIN_i \end{aligned}$$

# Just-In-Time Scheduling

---

- In this model, some jobs must start within a certain amount of time of other jobs
- Let  $S_{ij}$  be the max length of time between the start of job  $i$  and the start of job  $j$
- How do we modify the formulation to handle this?

$$\begin{aligned} \min z &= u_d - u_s \\ \text{subject to} \\ u_j - u_i &\geq C_i \text{ for all } i, j \in ARCS(i, j) \\ u_j &\leq u_i + S_{ij} \text{ for all } i, j \text{ with } S_{ij} \geq 0 \\ u_i &\text{ unrestricted for all } i \end{aligned}$$

# Another Twist

---

- Suppose instead we penalize the time difference between the completion of job  $i$  and the start of job  $j$
- Let:
  - $P_{ij}$  be the late penalty per unit time
  - $TOT$  be the total desired project time
- The following formulation minimizes these penalties:

$$\begin{aligned} \min z = & \sum_{i,j \in ARCS(i,j)} (u_j - u_i - C_i) * P_{ij} \\ \text{subject to} & \\ & u_d - u_s \leq TOT \\ & u_j - u_i \geq C_i \text{ for all } i, j \in ARCS(i, j) \\ & u_i \text{ unrestricted for all } i \end{aligned}$$

# How Do You Find the Critical Path?

---

- Suppose you solve the example problem in MPL
- You get task start times, but don't know which ones are critical
- The key is to look at the dual values of the constraints, which represent the arcs
- Any arc with a *nonzero dual value* is on the critical path

i	j	Slack	Shadow Price
s	A	0	1
A	B	0	1
B	C	-3	0
B	D	-2	0
B	E	0	1
C	F	0	0
C	K	-13	0
D	G	0	0
E	F	0	1
E	K	-13	0
F	I	0	1
G	H	0	0
H	J	0	0
I	J	0	1
J	d	0	1
K	d	0	0

# MPL Code

---

## TITLE

```
CPM; { Schrage CPM example; MPL must be }  
      { in case sensitive mode! }
```

## INDEX

```
node := (s,A,B,C,D,E,F,G,H,I,J,K,d);  
i     := node;  
j     := node;
```

## DATA

```
{ prec is used to define precedence arcs }
```

```
prec[i,j] :=
```

```
[s,A,1,  
A,B,1,  
B,C,1, B,D,1, B,E,1,  
C,F,1, C,K,1,  
D,G,1,  
E,F,1, E,K,1,  
F,I,1,  
G,H,1,  
H,J,1,  
I,J,1,  
J,d,1,  
K,d,1];
```

```
{ note 0 (dummy) duration times for s and d }
```

```
dur[i] := (0,3,4,2,3,5,3,4,6,7,5,2,0);
```

## VARIABLES

```
u[node];
```

## MODEL

```
min span = u["d"] - u["s"];
```

## SUBJECT TO

```
precedence[i,j] where prec[i,j]>0:
```

```
u[node:=j] - u[node:=i] > dur[i];
```

## END