Graphs and Network Flows
IE411

Lecture 2

Dr. Ted Ralphs

References for Today's Lecture

- Required reading
  - Sections 17.2-17.5
- References
  - AMO Sections 2.3
  - CLRS Section 22.1

Network Representation

- Our goal is to develop "efficient" algorithms $\rightarrow$ reasonable computation time.
- The main factors affecting efficiency are
  - The underlying algorithm
  - Data structure for storing the network
- The same algorithm may behave much differently with different graph data structure.
- What information do we need to store?
  - network topology (structure of nodes and arcs)
  - associated data (costs, capacities, supplies/demands)
- What are the important operations we might need to perform with a network data structure?

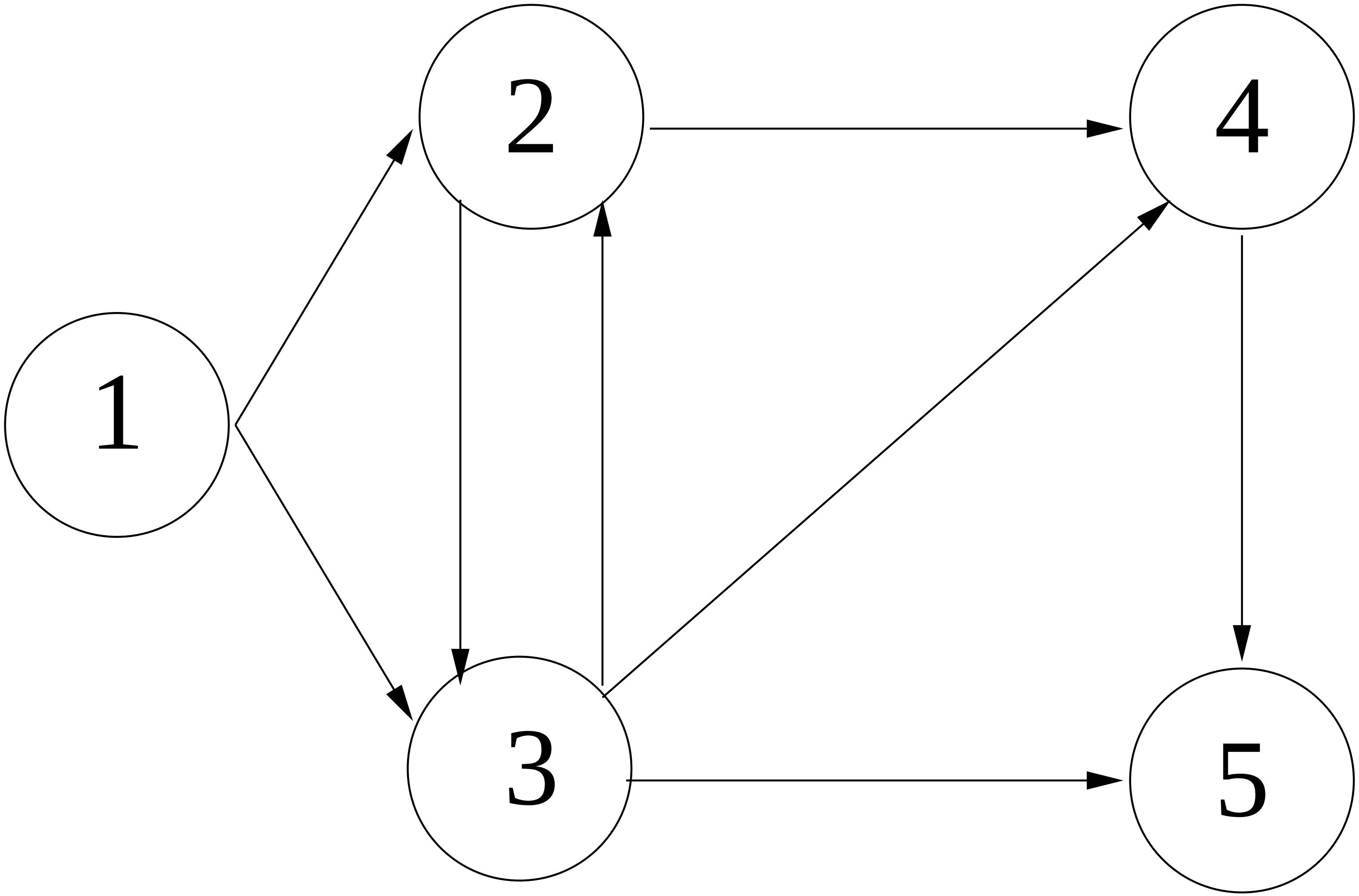
Common Representations

- Data structures
  - Node-Arc Incidence Matrix
  - Node-Node Adjacency Matrix
  - Adjacency List
  - Forward Star (Reverse Star)
- How do we evaluate a data structure?

Aside: Multiarcs and Loops

- *Multiarcs* are two or more arcs with the same tail and head nodes.
- A *loop* is an arc with the property that its tail and head nodes are the same.
- Generally we will assume that our networks do not contain parallel arcs or loops.
- The existence of such arcs can cause problems with standard data structures.

Example Graph



(Node-Arc) Incidence Matrix

- $n \times m$ matrix denoted $\mathcal{N}$.
- One row for each node and one column for each arc.
- For each arc (i, j), put +1 in row i and -1 in row j.

	(1,2)	(1,3)	(2,3)	(2,4)	(3,2)	(3,4)	(3,5)	(4,5)
1								
2								
3								
4								
5								

(Node-Arc) Incidence Matrix

- What is the size of the matrix?
- How many entries are non-zero?
- What information do we get by reading across a row?
- Is this a space efficient representation?
- How about other operations?

(Node-Node) Adjacency Matrix

- $n \times n$ matrix denoted $\mathcal{H}$
- one row for each node and one column for each node
- entry $h_{ij} = 1$ if arc $(i, j) \in A$ (0 otherwise)

	1	2	3	4	5
1					
2					
3					
4					
5					

(Node-Node) Adjacency Matrix

- What is the size of the matrix?
- How many entries are non-zero?
- What data structures might we use to store arc costs and capacities?
- Is this a space efficient representation?
- What operations are most efficient with this data structure?

Adjacency List

- Adjacency list of node i, $A(i)$, is a list of the nodes j for which $(i, j) \in A$
- List stored as a *linked list*.
- Need one linked list of length $|A(i)|$ for each node.
- Cell can store additional fields such as arc cost and capacity
- Is this a space efficient representation?
- What operations are most efficient with this data structure?

Forward Star

- Stores node adjacency list of each node in one large array
- Associates a unique sequence number with each arc using a specific order starting with arcs outgoing from node 1, then node 2, etc.
- Stores tail information about each arc in **tail** array, head information in **head** array, etc.
- Maintains a pointer for each node that indicates the smallest numbered arc in the arc list for that node.
- For consistency, set pointer(1) to 1 and pointer($n+1$) to $m+1$.
- What are the advantages of this representation?

Reverse Star

- Similar to a forward start except that arcs are sequenced starting with arcs incoming from node 1.
- The two representations can be maintained side-by-side if necessary.

Miscellaneous Issues

- Parallel Arcs
  - Why would we need parallel arcs?
  - Which representation(s) could accommodate them?
- Undirected Network
  - What needs to change?
    - Node-Arc Incidence Matrix
    - Node-Node Adjacency Matrix
    - Adjacency List
  - What needs to happen when we update (i, j)?

Summary of Representations

Representation	Storage Space	Features
Incidence Matrix	nm	1. Space inefficient 2. Expensive to manipulate 3. MCFP constraint matrix
Adjacency Matrix	kn^2	1. Suited to dense networks 2. Easy to implement
Adjacency List	$k_1n + k_2m$	1. Space efficient 2. Efficient to manipulate 3. Suited to dense and sparse
Forward Star	$k_3n + k_4m$	1. Space efficient 2. Efficient to manipulate 3. Suited to dense and spare

Table 1: From Ahuja et al. Figure 2.25