**ISE 532: Network Flows**

# 1: Introduction, Notation, and First Algorithms

*Lecturer: Fernando Ordóñez*

## 1 Introduction to Network Flows

Networks arise in many applications in everyday life: The Internet, traffic networks, logistic networks, supply chains, etc. In Network Flows we consider optimization problems over a given network. The notation to represent the network is by no means unique.

For example consider the following network, or directed graph, or digraph, or graph:

| A.M.O | alternate |
|---|---|
| $G = (N, A)$ | $G = (V, E)$ |
| Node set $N = \{1, 2, 3, 4\}$ | Vertex set $V = \{1, 2, 3, 4\}$ |
| Arc set $A =$ | Edge set $E =$ |
| $\{(1, 2), (1, 3), (3, 2), (3, 4), (4, 1)\}$ | $\{1 - 2, 1 - 3, 3 - 2, 3 - 4, 4 - 1\}$ |

A network will have integer numbers associated with the nodes and arcs of the graph above. The cost per unit flow $c_{ij}$, capacity $u_{ij}$, and lower bound $l_{ij}$ can be associated to arc $(i, j)$. A supply/demand $b(i)$ can be associated to node $i \in N$. If $b(i) > 0$ it is a supply node, if $b(i) < 0$ it is a demand node, and if $b(i) = 0$ it is a transshipment node.

Some classic optimization problems on networks are:

- *Minimum cost flow problem.* Given a network with capacities and costs on its arcs, and that a certain amount of flow has to be transported from sources/supply nodes to sinks/demand nodes in the network. What is the least expensive way to transport the

flow? Let $x_{ij}$ be the flow on arc $(i,j)$, the problem can be expressed by the following linear program:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \quad \text{for all } i \in N \\ & l_{ij} \le x_{ij} \le u_{ij} \qquad\qquad\qquad\qquad \text{for all } (i,j) \in A\ . \end{aligned}$$

- *Shortest path problem.* What is the shortest way to go from a specified node $s$ to a specified node $t$ in a network. How do we write this as a minimum cost flow problem?

- *Maximum flow problem.* Given a network with arc capacities. How much flow can be sent from a specified source/supply node $s$ to a specified sink/demand node $t$?

  This problem can be represented by

$$\begin{aligned} \max \quad & v \\ \text{s.t.} \quad & \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \quad \text{for all } i \in N \\ & l_{ij} \le x_{ij} \le u_{ij} \qquad\qquad\qquad\qquad \text{for all } (i,j) \in A\ , \end{aligned}$$

  where

$$b(i) = \begin{cases} v & \text{if } i = s \\ -v & \text{if } i = t \\ 0 & \text{if } i \neq s,t\ . \end{cases}$$

  How can we transform this problem to a minimum cost flow problem?

- *Assignment problem.* We partition the set of nodes into two equally sized sets $N_1$ and $N_2$ and define a collection of pairs $A \subseteq N_1 \times N_2$ representing the possible assignments, each with an associated cost $c_{ij}$. We want to find the set of assignments of minimum total cost that associates each node in $N_1$ to a node in $N_2$. This is a particular case of the minimum cost flow problem.

- *Circulation problem.* A special case of minimum cost flow problem where there are no supply or demand nodes. In other words $b(i) = 0$ for all $i \in N$. We obtain the minimum cost feasible circulation of flow on the network.

- *Minimum spanning tree problem.* A spanning tree is a tree (i.e. a connected acyclic graph) that spans (reaches) all the nodes of an undirected network. The cost of a spanning tree is the sum of the costs of all the arcs in the tree. In the minimum spanning tree problem we wish to identify the spanning tree of minimum cost.

- *Matching problem.* A matching in a network $G = (N, A)$ is a set of arcs with the property that every node in $N$ is incident to at most one arc in the matching. In a matching every node is matched to at most one other node, and some nodes might not be matched with any other node. We look for the matching that has maximum cardinality or maximum weight.

- *Traveling Salesman Problem.* Starting from node 1 design a tour that visits all nodes $2, \ldots, n$ exactly once and returns to 1 with minimum total cost.

# 2 Applications

## 2.1 Hopping airplane problem

A small commuter airline uses a plane, with a capacity to carry at most $p$ passengers, on a hopping flight. This hopping flight visits the cities $1, 2, \ldots, n$ in a fixed sequence. The plane can pick up passengers at any node and drop them at any other node. Let $b_{ij}$ denote the passengers at node $i$ that are going to node $j$, and let $f_{ij}$ denote the fare per passenger from node $i$ to node $j$. Set up a minimum cost flow problem that will determine the number of passengers to pick up at every city in order to maximize revenue on the hopping plane.

## 2.2 Assortment of Structural Steel Beams

A construction company needs structural steel beams of a uniform cross section but of varying lengths. For each $i = 1, \ldots, n$ let $D_i > 0$ denote the demand of the steel beam of length $L_i$. Assume $L_1 < L_2 < \ldots < L_n$. Given that there exists a cost to set up an inventory for each beam length, it can be economical not to carry an inventory of all beam lengths. When a length not in inventory is needed, a longer beam can be cut to the desired length. The remainder length will be discarded as unusable steel scrap. Assume that $K_i$ is the cost to set up inventory of beams of length $L_i$, and $C_i$ is the cost of a beam of length $L_i$.

Model this problem as a shortest path network problem that will determine the lengths of beams that must be carried in inventory in order to minimize inventory costs.

### 2.3 Machine Set-up Costs

Assume that we want to schedule $n$ jobs in a machine. We incur in a set-up cost of $c_{ij}$ if we schedule job $j$ right after job $i$. What problem do we solve in order to determine the job schedule with least total set-up cost.

## 3 Notation and Definitions

To illustrate the following definitions consider the following network

- **Directed graphs and networks:** A directed graph is a set of nodes $N = \{1, 2, \ldots, n\}$ and a set of arcs which are ordered pairs of distinct nodes. For example, for the graph above
  $N =$
  $A =$

  A directed network is a directed graph whose nodes and/or arcs have associated numerical values (capacities, costs, and/or supplies and demands).

- **Undirected graphs and networks:** An undirected graph is a set of nodes $N = \{1, 2, \ldots, n\}$ and a set of edges which are pairs of distinct nodes. The only difference with a directed graph is that the edges do not have a direction. Flow can traverse the edge both ways.

- **Degrees:** The indegree of a node is the number of incoming arcs, and its outdegree is the number of its outgoing arcs. The degree of a node is the sum of its indegree and outdegree.

  For example, node 5 in the graph above has
  indegree = , outdegree = , and degree = .

  Show that the sum of indegrees of all nodes = sum of outdegrees of all nodes = number of arcs in the graph.

- **Adjacency list:** The arc adjacency list $A(i)$ is the set of arcs emanating from node $i$, that is $A(i) = \{(i, j) \in A \mid j \in N\}$. The node adjacency list $A(i)$ is the set of nodes that are adjacent to node $i$, that is $A(i) = \{j \in N \mid (i, j) \in A\}$.

  Note that $\sum_{i \in N} |A(i)| = m$.

- **Subgraph:** A graph $G' = (N', A')$ where $N' \subseteq N$ and $A' \subseteq A$.

  Draw the subgraph $G'$ induced by $N' = \{1, 2, 5, 6\}$.

  Draw a spanning subgraph of $G$.

- **Walk, Directed walk:** A walk is a subgraph formed by a sequence of nodes and arcs $i_1, a_1, i_2, a_2, \ldots, a_{r-1}, i_r$ such that for $1 \leq k \leq r-1$ either $a_k = (i_k, i_{k+1}) \in A$ or $a_k = (i_{k+1}, i_k) \in A$. In a directed walk $a_k = (i_k, i_{k+1}) \in A$ for every $1 \leq k \leq r-1$. For example:

- **Path, Directed path:** A path (directed path) is a walk (directed walk) that does not repeat nodes. For example:

- **Cycle, Directed cycle:** A cycle (directed cycle) is a path (directed path) with the additional arc $(i_r, i_1)$ or $(i_1, i_r)$ (only $(i_r, i_1)$ if directed). For example:

- **Connectivity, Strong Connectivity:** Nodes $i$ and $j$ are connected if there exists a path from $i$ to $j$. A graph $G$ is connected if every pair of nodes in the graph are connected, otherwise the graph is disconnected. A graph is strongly connected if from every node there is a directed path to every other node in the graph.

  Draw a connected graph that is not strongly connected.

Draw a disconnected graph with a strongly connected component.

- **Cut:** A cut is a partition of the set $N$ into two parts $S$ and $\bar{S} = N - S$. The set of arcs from $S$ to $\bar{S}$ is denoted by $[S, \bar{S}]$.

- **Tree, Forest, Subtree:** A tree is a connected graph that contains no cycle. A forest is a graph that contains no cycle, a possibly disconnected graph with every connected component a tree. A subtree is a connected subgraph of a tree.

  **Proposition 1.** *1. A tree on $n$ nodes contains exactly $n - 1$ arcs.*

  *2. A tree has at least two leaf nodes (i.e. nodes with degree 1).*

  *3. Every two nodes of a tree are connected by a unique path.*

  **proof:**

- **Bipartite Graph:** A graph $G = (N, A)$ is bipartite if we can partition $N$ into $N_1$ and $N_2$ such that for every $(i, j) \in A$ either $i \in N_1$ and $j \in N_2$ or $i \in N_2$ and $j \in N_1$. Draw a bipartite graph.

# 4 Complexity Analysis

All the network problems discussed so far can be solved by enumerating all possible solutions and picking the one with the optimal objective function value. This procedure will be usually unpractical, since the number of possible solutions can be astronomical.

**Example** What is the running time of the following algorithm that adds two $m \times n$ matrices:

```
for i = 1 to m do
    for j = 1 to n do
        c(i, j) = a(i, j) + b(i, j)
```

Additions
Assignments
Comparisons
Look up cost

Some simplifications to determine algorithmic complexity:

- Ignore constants in the running time.
- Running times are stated in terms of relevant problem parameters.
- Use worst-case analysis.
- All arithmetic operations are assumed to take one step.
- Assume a *Random Access Machine* (RAM). Can use arrays, and select any element of an array in 1 step.
- All numbers are integral.

Revisit complexity of sum of matrices.

For network flow problems on $G = (N, A)$, the relevant problem parameters are

| | |
|---|---|
| $n := \|N\|$ | number of nodes |
| $m := \|A\|$ | number of arcs |
| $U$ | upper bound on arc flow capacities |
| $C$ | upper bound on cost coefficients |

### 4.1 Complexity of Algorithms

An algorithm to solve an instance of a network flow problem runs in

- **pseudo-polynomial time** iterations are bounded by a polynomial in $n,\ m,\ U,\ C$.
- **polynomial time** iterations are bounded by a polynomial in $n, m, \log U$, and $\log C$.
- **strongly polynomial time** iterations are bounded by a polynomial in $n$ and $m$.

Exponential growth

| Function | | | |
|---|---|---|---|
| $n$ | 10 | 100 | 1000 |
| $n \log n$ | | | |
| $n^3$ | | | |
| $10^6 n^8$ | | | |
| $2^n$ | | | |

| | | | | |
|---|---|---|---|---|
| 10 | 100 | 1000 | 10000 | 1.0E+05 |
| 10 | 200 | 3000 | 40000 | 5.0E+05 |
| 1000 | 1.0E+06 | 1.0E+09 | 1.0E+12 | 1.0E+15 |
| 1.0E+14 | 1.0E+22 | 1.0E+30 | 1.0E+38 | 1.0E+46 |
| 1024 | 1.3E+30 | 1.1E+301 | #NUM! | #NUM! |

en mins, 2 GHz

| | | | | |
|---|---|---|---|---|
| 8.3E-11 | 1.7E-09 | 2.5E-08 | 3.3E-07 | 4.2E-06 |
| 8.3E-09 | 8.3E-06 | 8.3E-03 | 8.3E+00 | 8.3E+03 |
| 8.3E+02 | 8.3E+10 | 8.3E+18 | 8.3E+26 | 8.3E+34 |
| 8.5E-09 | 1.1E+19 | 8.9E+289 | #NUM! | #NUM! |

## 5 Graph Search

*Breadth First Search Algorithm.*

Input: A Directed graph $G = (N, A)$, and a starting node $s \in N$.
Output: An set of nodes that can be reached from $s$.
Define Arrays: Reachable[$n$], Prior[$n$].

Set Reachable[$s$] = yes and Reachable[$j$] = no for all $j \in N - \{s\}$. LIST = {s}
while LIST != empty do
    select $f$ the first node from LIST
    find $j \in A(f)$ such that Reachable[$j$] == no
    if $j$ exists then
        Reachable[$j$] = yes, Prior[$j$] = $f$, add $j$ to LIST.
    else remove $f$ from LIST
end

What is the output of this algorithm? Its complexity? What is the complexity to find all connected components?

*Depth First Search Algorithm.*

Set Reachable[$s$] = yes and Reachable[$j$] = no for all $j \in N - \{s\}$. LIST = {s}
while LIST != empty do
    select $l$ the last node from LIST
    find $j \in A(l)$ such that Reachable[$j$] == no
    if $j$ exists then
        Reachable[$j$] = yes, Prior[$j$] = $l$, add $j$ to LIST.
    else remove $l$ from LIST
end

What is the output of this algorithm? Its complexity? What is the complexity to find all connected components?

# 6 Flow Decomposition

Any feasible flow on a network can be decomposed into the sum of flows around cycles plus the sum of flows from nodes with supply to nodes with demand.

Consider the following network and circulation:

*Flow decomposition algorithm*

1. Locate a cycle $C$ with positive flow using depth first search
2. Determine max flow through the cycle
3. Remove flow through the cycle from network

Complexity? Why can we start this algorithm?

**Lemma 2.** *If a directed graph has no node with indegree 0, then there is a directed cycle.*

**proof:**

How do we adapt the previous algorithm to consider general flows?

## 6.1 Application of Flow decomposition

Consider the problem of finding a shortest path from node 1 to all other nodes. Assume all arc costs are non-negative.

In order to tackle this problem, solve

$$\begin{array}{ll} \min & c^t x \\ \text{s.t.} & Nx = \begin{cases} n-1 & \text{if } i = 1 \\ -1 & \text{if } i \neq 1 \end{cases} \\ & x \geq 0 \ . \end{array}$$

# 7 Network Transformations

## 7.1 Removing Nonzero Lower Bounds

## 7.2 Arc Reversal

## 7.3 Removing Arc Capacities

### 7.4 Node Splitting

### 7.5 Residual Networks