

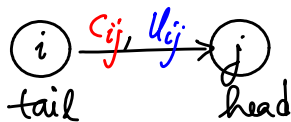
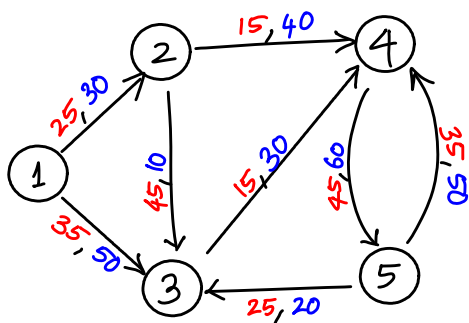
MATH 566: Lecture 5 (09/08/2026)

Today: * forward star representation
* network transformations

Forward Star Representation

A list of info about arcs, set as a table. There is one row for each arc $(i, j) \in A$, and 3-5 columns listing the tail (i), head (j), and c_{ij} , l_{ij} , and u_{ij} values.

Illustration



T	H	COST	UB
1	2	25	30
1	3	35	50
2	3	45	10
2	4	15	40
3	4	15	30
4	5	45	60
5	3	25	20
5	4	35	50

Assume $l_{ij} = 0 \ \forall (i, j) \in A$

The main matrix is $m \times 3$, or $m \times 4$ or $m \times 5$, depending on which of the parameters c_{ij} , l_{ij} , and u_{ij} are given. The $b(i)$ values are stored in a separate n -vector.

Notice that the information is stored in the order of the arcs. To extract the outarc list efficiently from this matrix notation, we use one more vector named the **point**. Most algorithms run steps of operations on the outarcs or inarcs of each node. Hence it is important to have these lists readily available, or easily extractable.

→ lexicographic order

	T	H	COST	VB
→ 1	1	2	25	30
2	1	3	35	50
→ 3	2	3	45	10
4	2	4	15	40
→ 5	3	4	15	30
→ 6	4	5	45	60
→ 7	5	3	25	20
8	5	4	35	50

point(i)

1	1
2	3
3	5
4	6
5	7
(nti) → 6	9

m+1 (#arcs+1)

$$\text{point}(\text{nti}) = m+1 \text{ (always)}$$

For $1 \leq i \leq n$, $\text{point}(i)$ stores the row # (index) of the first arc (in lex order) that has i as its tail. $\text{point}(\text{nti})$ is always set to $m+1$ (#arcs+1).

The outarcs of node i are in rows indexed $\text{point}(i)$ to $\text{point}(\text{iti})-1$, for $1 \leq i \leq n$.

For instance, for node 2, $\text{point}(2)=3$, $\text{point}(\text{iti}) = \text{point}(3)=5$. Hence, the outarcs of node 2 are in rows 3 to 5-1, i.e., rows 3 and 4.

The matrix ($m \times 3, m \times 4, \dots$) of arcs list along with the point vector (and b(.) vector) is the **forward star representation** of G .

Note: If node i has no outarcs, we set $\text{point}(i) = \text{point}(\text{iti})$, so that the indices of outarcs of node i is an empty set.

Here is a quick example: we reverse arc (3,4) in the example network. Rest of the network remains unchanged. Note that node 3 now has no outarcs. The corresponding point(.) vector is shown to the right.

	T	H	COST	UB
1	1	2	25	30
2	1	3	35	50
3	2	3	45	10
4	2	4	15	40
5	4	3	15	30
6	4	5	45	60
7	5	3	25	20
8	5	4	35	50

point(i)

1	1
2	3
3	5
4	5
5	7
6	9

Notice how the outarcs of node 2 are still recorded correctly: arcs $\text{point}(2)=3$ to $\text{point}(3)-1=4$.

Equivalently, we could have a matrix of inarcs along with the associated parameters (c_{ij}, u_{ij}, l_{ij}) , and create the **rpoint** vector, which is the **reverse point** vector. The in-arcs of node i are precisely the arcs indexed $\text{rpoint}(i)$ to $\text{rpoint}(i+1)-1$.

For small or moderately sized networks, it may be simpler to initialize the $A\{\cdot\}$ lists as a cell array (outarc list). For large networks, the use of **point** is recommended.

Here are some basic Matlab commands and files

netdata.m

```
%% Math 566 (Fall 2026)
%% Matlab code to extract out- and in-arc lists
%% from the forward star representation

%T: TAIL, H: HEAD
%DEGO: outdegree, DEGI: indegree
%n: number of nodes, m: number of arcs
%A{i}: out-arc list at node i, cell array
%AI{i}: in-arc list at node i, cell array

DEGO=zeros(1,n);
DEGI=DEGO;
A{1,n} = [];
AI{1,n} = [];
for k=1:m
    i=T(k);
    j=H(k);
    pos=DEGO(i)+1;
    DEGO(i)=pos;
    A{i}(pos)=k;

    posI=DEGI(j)+1;
    DEGI(j)=posI;
    AI{j}(posI)=k;
end%for
```

Example1.m

```
%% Math 566 (Fall 2026)
%%
%% Network from Lecture 4, which is a slightly modified
%% version of the network in AMO Figure 2.13, page 32.

data=[1 1 2 25 30
      2 1 3 35 50
      3 2 3 45 10
      4 2 4 15 40
      5 3 4 15 30
      6 4 5 45 60
      7 5 3 25 20
      8 5 4 35 50];

% The first column is just the arc index (or number)
% It is redundant info, and is listed here just for
% the sake of convenience.

% data=data(3:end,:);
T=data(:,2);
H=data(:,3);
m=length(T);
n=max(max(T), max(H));

netdata

DEGO
celldisp(A)

DEGI
celldisp(AI)
```

Output from Matlab:

```
% Matlab session
% Lecture 5, Sep 08, 2026

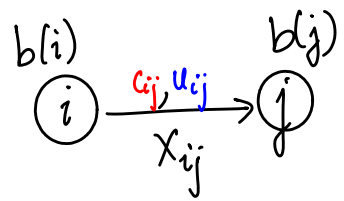
>> example1
DEGO =
     2     2     1     1     2
A{1} =
     1
A{2} =
     3     4
A{3} =
     5
A{4} =
     6
A{5} =
     7     8
DEGI =
     0     1     3     3     1
AI{1} =
     []
AI{2} =
     1
AI{3} =
     2     3     7
AI{4} =
     4     5     8
AI{5} =
     6
>>
```

We will implement several algorithms we discuss in class in Matlab.

Network Transformations

Recall the model for min-cost flow:

We discuss several ways to transform networks. The goal is to make the network amenable to algorithms requiring certain structure, while not changing the problem entirely.



$$\begin{aligned} \min \quad & \sum_{(i,j) \in A} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_j x_{ij} - \sum_j x_{ji} = b(i) \quad \forall i \\ & l_{ij} \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A. \end{aligned}$$

1. Removing nonzero lower bounds

(We typically take $l_{ij} = 0$. But if $l_{ij} > 0$, we could transform the problem to an equivalent problem with $l_{ij} = 0$).

$$l_{ij} - l_{ij} \leq x_{ij} - l_{ij} \leq u_{ij} - l_{ij}$$

$$0 \leq \overset{\text{new flow}}{x'_{ij}} \leq \overset{\text{new } u_{ij}}{u'_{ij}}$$

$$\begin{aligned} x_{ij} - l_{ij} &= x'_{ij} \\ \text{so, } x_{ij} &= x'_{ij} + l_{ij}, \text{ and} \\ u'_{ij} &= u_{ij} - l_{ij}. \end{aligned}$$

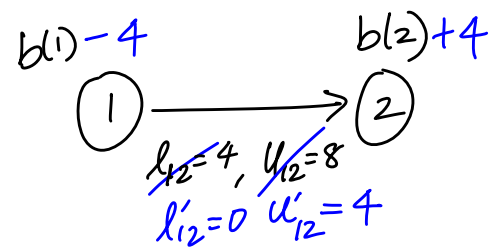
The flow balance constraint for node i becomes

$$\sum_{(i,j)} (x'_{ij} + l_{ij}) - \sum_{(j,i)} (x'_{ji} + l_{ji}) = b(i)$$

$$\Rightarrow \sum_{(i,j)} x'_{ij} - \sum_{(j,i)} x'_{ji} = b(i) - \underbrace{\sum_{(i,j)} l_{ij} + \sum_{(j,i)} l_{ji}}_{b'(i) \text{ new } b(i) \text{ value}}$$

The $b(i)$ values are updated as follows. We **subtract** the l_{ij} values for all **outarcs** of node i , and **add** the l_{ji} values for all **in arcs** of node i .

Here is an illustration on a single arc:
If we have to send at least 4 units along $(1,2)$, we could imagine that much flow being "taken out of" $b(1)$, and being added to $b(2)$.



What about the objective function?

$$\sum_{(i,j)} c_{ij} x_{ij} = \sum_{(i,j)} c_{ij} (x'_{ij} + l_{ij}) = \sum_{(i,j)} c_{ij} x'_{ij} + \underbrace{\sum_{(i,j)} c_{ij} l_{ij}}_{\text{constant; does not depend on } x_{ij} \text{ values.}}$$

Adding a constant to the objective function does not change the optimal solution. Hence we will indeed find the optimal solution to the original problem.

Once you have the optimal solution x'_{ij} , we can recover the corresponding optimal solution x_{ij} by computing $x'_{ij} + l_{ij}$.