GRAPH THEORY and APPLICATIONS

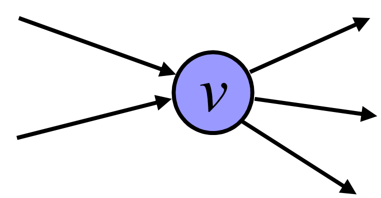
Networks an Flows

Network

- **Network**: A finite connected digraph in which:
  - one vertex x, with $d^+(x) > 0$ is called the source.
  - one vertex y, with $d^-(y) > 0$ is called the sink.
- A **flow** for the network N, associates:
  - a non-negative integer $f(u,v)$,
  - with each edge (u,v) of N, such that,

  for all vertices v, other than x and y:

$$\sum_{u} f(u,v) = \sum_{u} f(v,u)$$



- Conservation of flow at each vertex.

Capacity

- A network is a model for the flow of material leaving a single departure point, and arriving at a single destination.
- In practise, there is an upper bound on the possible flow along any edge.
- For each edge (u,v):
  - $c(u,v)$: capacity of the edge (a non-negative integer)
- Henced, for each edge (u,v):

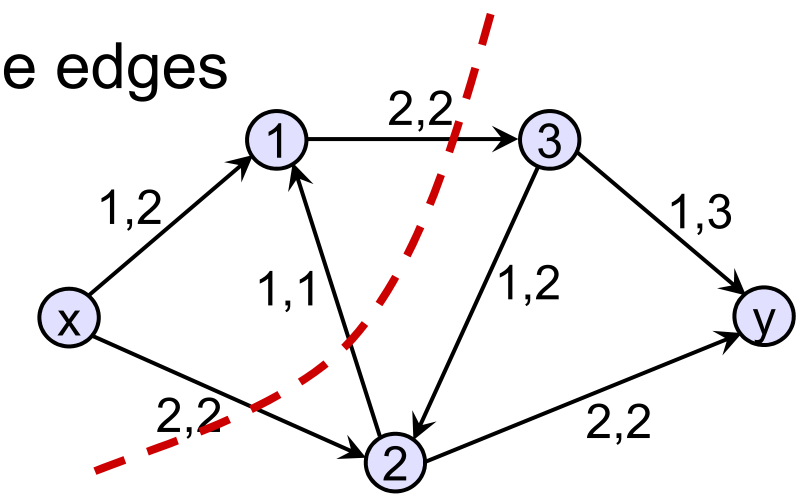
$$0 \leq f(u,v) \leq c(u,v)$$

Cut

- A cut of N=(V,E) is a cut-set of the underlying graph.
  - Denoted by $(P, \overline{P})$ where $x \in P, y \in \overline{P}$

    $P \cap \overline{P} = \varnothing \quad P \cup \overline{P} = V$
- The capacity of a cut $(P, \overline{P})$:
  - Denoted by $K(P, \overline{P})$
  - Sum of the capacities of those edges
    - incident from vertices in P, and
    - incident to vertices in $\overline{P}$.

$$K(P, \overline{P}) = \sum_{u \in P, v \in \overline{P}} c(u, v)$$



Value of a flow

- The value of the flow $F(N)$ for a network is the *net flow* leaving the source x:

$$F(N) = \sum_{v} f(x,v) - \sum_{v} f(v,x)$$

Theorem: For an arbitrary cut of the network N, the value of the flow is given by:

$$F(N) = \sum_{u \in P, v \in \overline{P}} f(u,v) - \sum_{u \in \overline{P}, v \in P} f(u,v)$$

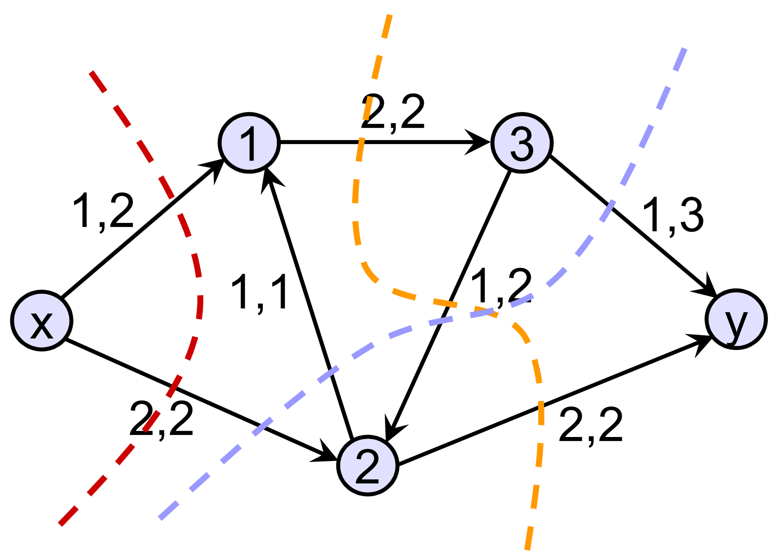
$$= (\text{flow from P to } \overline{\text{P}}) - (\text{flow from } \overline{\text{P}} \text{ to P})$$

Value of a flow

Corollary: The value of the flow for any network cannot exceed the capacity of any cut:

$$F(N) \leq \min(K(P, \overline{P}))$$

Example:



A path in a network

- The corollary provides an upper bound for the maximum flow in a network.
- We focus on finding a flow of maximum value in any given network.
- Path: A sequence of distinct vertices $Q = (v_0, v_1, \ldots, v_k)$ from the source x to the sink y, where,
  - v_0 = x,
  - v_k = y, and
  - Q is a path in the underlying graph of N.
- For any two consecutive vertices v_i and v_{i+1} of Q, either $(v_i, v_{i+1}) \in$ E or $(v_{i+1}, v_i) \in$ E.
  - (v_i, v_{i+1}) is called a ***forward***-edge.
  - (v_{i+1}, v_i) is called a ***reverse***-edge.

Augmenting path

- Augmenting path: For a given flow F(N), a path Q of N such that for each $(v_i, v_{i+1}) \in Q$:
  - if (v_i, v_{i+1}) is a forward-edge, then:

    $$\Delta_i = c(v_i, v_{i+1}) - f(v_i, v_{i+1}) > 0$$

  - if (v_i, v_{i+1}) is a reverse-edge, then:

    $$\Delta_i = f(v_{i+1}, v_i) > 0$$

- If Q is an augmenting path then we define Δ as follows:

  $$\Delta = \min \Delta_i > 0$$

Augmenting the flow

- Each (v_i, v_{i+1}) of Q, for which $\Delta_i = \Delta$ is called a bottleneck-edge relative to F(N) and Q.
- For a given network and flow F(N):
  - If the augmenting path Q exists, then we can construct a new flow F'(N).
  - The value of F'(N) is equal to the value of F(N) plus Δ.
    - If (v_i, v_{i+1}) is a forward-edge then:

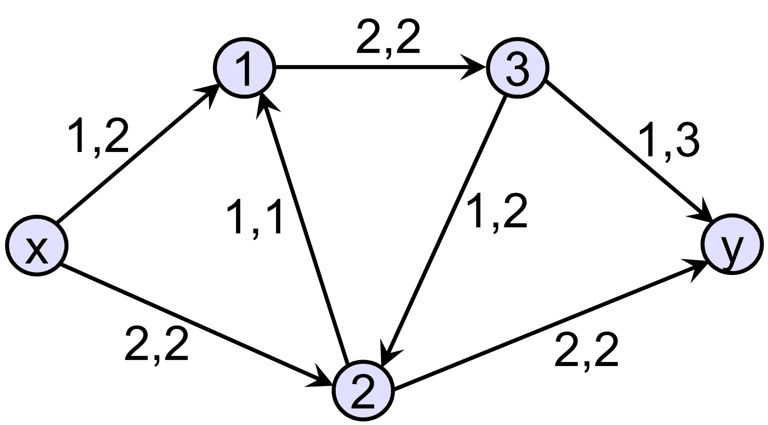
      $$f(v_i, v_{i+1}) \leftarrow f(v_i, v_{i+1}) + \Delta$$

    - If (v_i, v_{i+1}) is a reverse-edge then:

      $$f(v_{i+1}, v_i) \leftarrow f(v_{i+1}, v_i) - \Delta$$

Augmenting the flow

- The addition of Δ along an augmenting path preserves the conservation of flow requirement, at each vertex except x and y.
- The net flow from x is increased by the addition of Δ to the flow along (x,v_1).



Q = (x,1,2,3,y)

Forward-edges: (x,1) and (3,y)
Reverse-edges: (1,2) and (2,3)
Bottleneck edges: All except (3,y)
Δ = 1
Assign:

f(x,1) = 2 f(1,2) = 0
f(2,3) = 0 f(3,y) = 2

Maximum-flow problem

- The idea of augmenting path forms a basis for an algorithm: Ford-Fulkerson
- Start from an initial flow $F_0(N)$
  - Could be a zero flow
- Construct a sequence of flows $F_1(N)$, $F_2(N)$, …
  - $F_{i+1}(N)$ is constructed from $F_i(N)$ by finding an augmenting path.
- Termination is guaranteed, because:
  - $F_{i+1}(N)$ is greater than $F_i(N)$, and bounded.
- If no augmenting path exists then $F_i(N)$ is maximum. (proof: Gibbons, p.100)

Max-flow min-cut theorem

- The outlined algorithm shows that it is always possible to attain a flow value F(N) equal to:

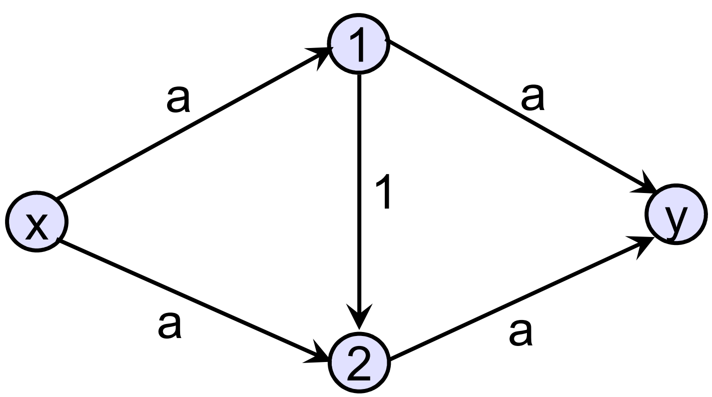
$$\min(K(P,\bar{P}))$$

Theorem: (Max-flow min-cut by Ford and Fulkerson)
For a given network the maximum possible value of the flow is equal to the minimum capacity of all cuts.

$$\max F(N) = \min(K(P,\bar{P}))$$

How to find an augmenting path?

- Assume: each augmentation increases the flow from x to y by one unit.
  - Number of augmentations: $K(P,\overline{P})$
  - No relation to network size.



Select alternatively:
P1 = (x,1,2,y)   P2 = (x,2,1,y)

- Each augmentation enhances the flow by 1 unit.
- Overall 2a augmentations will be required.

How to find an augmenting path?

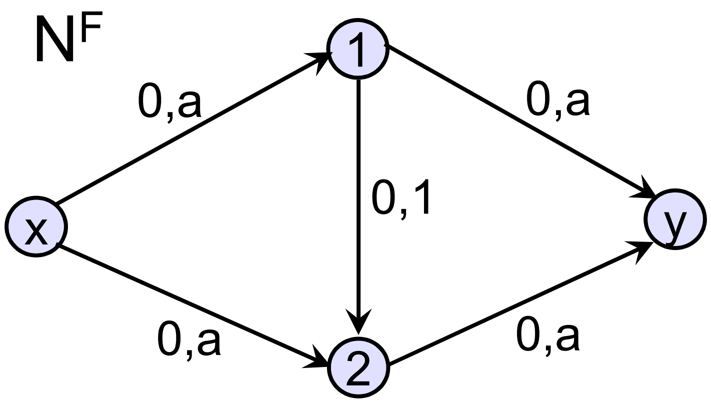
- An algorithm of Edmonds & Karp.
- Polynomially dependent upon network size only.
- Given N=(V,E) with a flow, construct an associated network N^F=(V, E'):
  - N and N^F have the same vertex set.
  - For any two vertices u and v, (u,v) is an edge of N^F if and only if, either:

$$(u,v) \in E \text{ and } c(u,v) - f(u,v) > 0$$

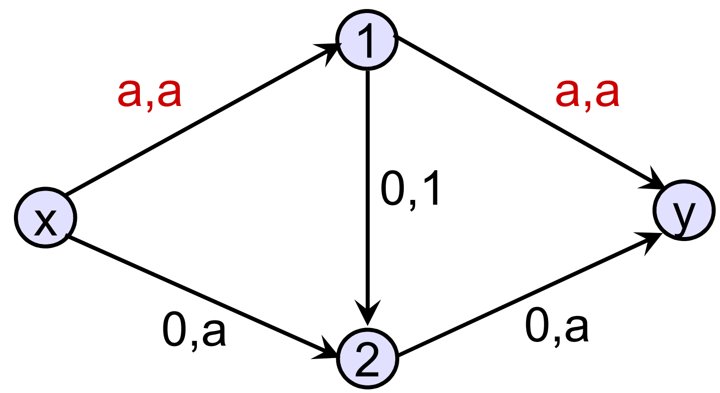
or

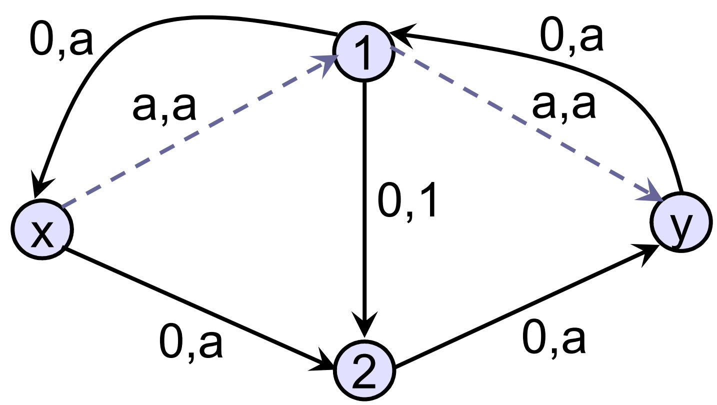
$$(v,u) \in E \text{ and } f(v,u) > 0$$

Example



Shortest path: (x,1,y)





Determining augmenting path

- Finding an augmentation path
  $\Rightarrow$ Finding a directed path from x to y in N^F
- P^F: a directed path in N^F
- To determine P^F:
  - Each vertex v is labeled $L(v)$:
    Minimum distance from x to v.
    $L(v) = 0$ if there is no path
  - If a path exists from x to y, choose the minimum-length path.
  - Trace the path backwards from y to x.

Finding edge-connectivity

- $p_e(u,v)$: Number of edge disjoint paths between u and v.
- $c_e(u,v)$: Smallest cardinality of those cutsets which partition the graph, so that:
  - u is in one component
  - v is in the other component.

A variation of Menger's theorem: Let G be an undirected graph with $u,v \in V$, then:

$c_e(u,v) = p_e(u,v)$

Proof

- From G construct a network N:
  - N contains the same vertex set as G
  - For each edge (u,v) of G, N contains (u,v) and (v,u).
  - For each edge e of N, assign a capacity c(e) = 1.
- Thus, any flow in N is either 0 or 1.
- F: Maximum value of a flow from a source to a sink.
- Show that: F = $p_e(x,y)$.
  - $p_e(x,y)$ edge-disjoint paths from x to y in G
    $\Rightarrow$ $p_e(x,y)$ edge-disjoint paths from x to y in N.
  - Each such path can transport 1 unit of flow.
  - Thus, $F \geq p_e(x,y)$

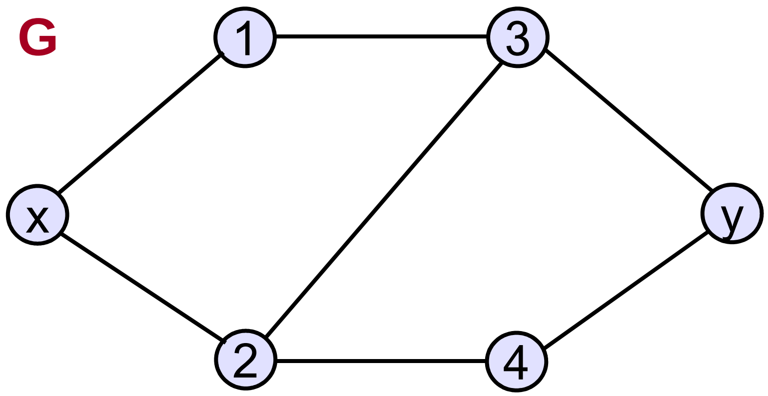
Proof

- For a maximum flow in N, we can assume that:
  - for each edge (u,v), not both of f(u,v) and f(v,u) are 1.
  - If they were, we could replace each flow by 0.
- Then, flow F consists of unit flows corresponding to edge-disjoint paths in G.
- Thus, $F \leq p_e(x,y)$.

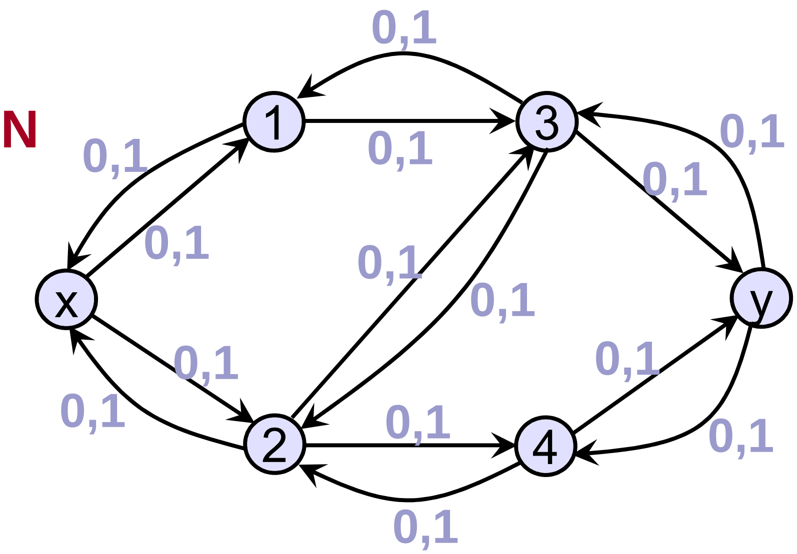
- Max-flow min-cut theorem
  $\Rightarrow$ F = the capacity of a minimum cut-set.
- Every path from x to y uses at least one edge of the cut.
- This cut would disconnect G, so, cut-set has cardinality F.

Example

G

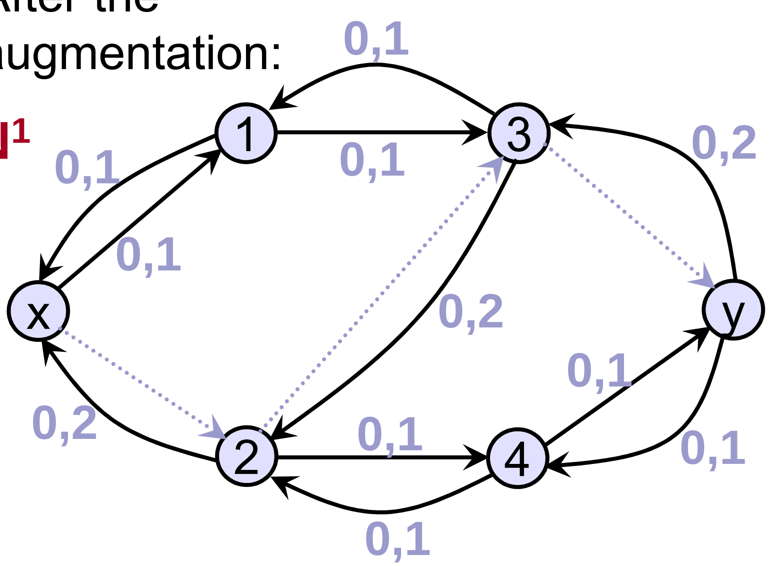


N

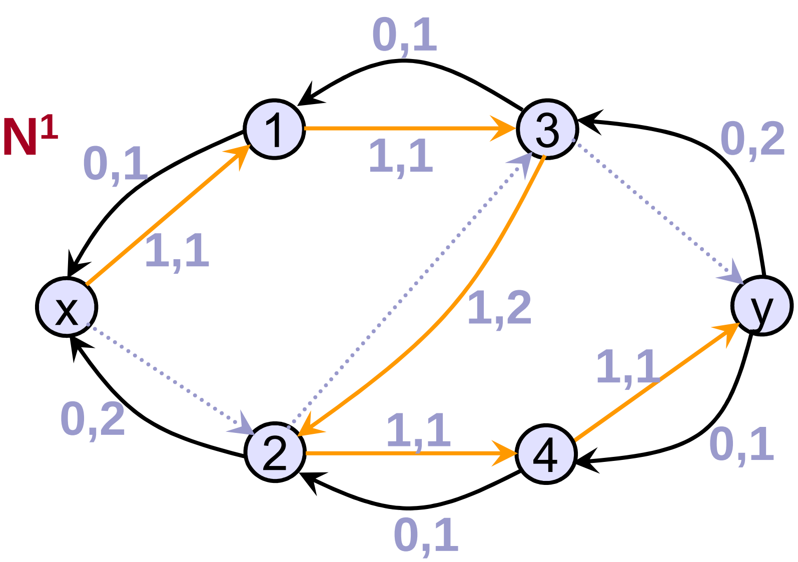


After the augmentation:

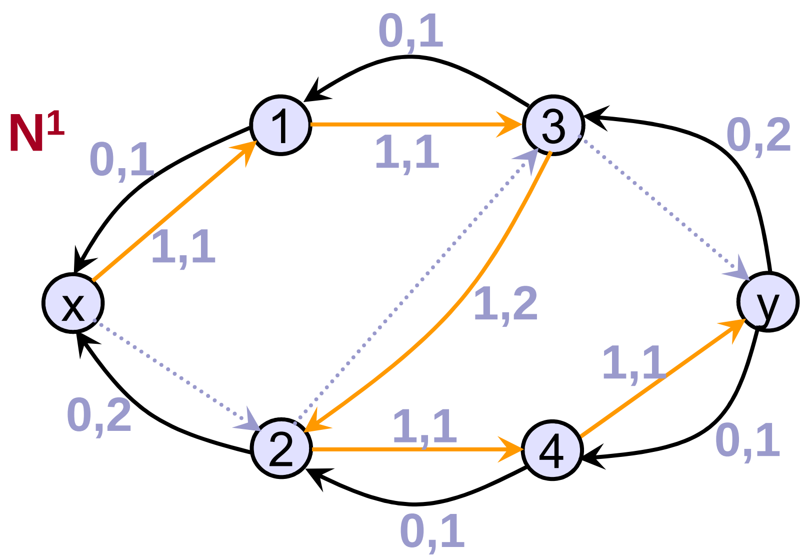
N¹



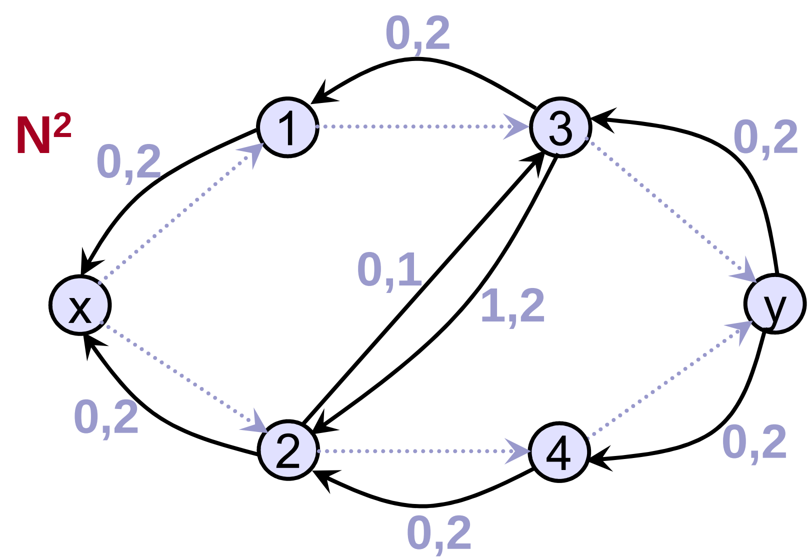
N¹



Example



After the augmentation:



Max flow = 2

Edge-connectivity

- From the definition of edge-connectivity κ'(G), and $c_e(u,v)$:

$$\mathrm{K}'(G) = \min_{u,v \in V} c_e(u,v)$$

- We can find κ'(G), by solving the maximum flow problem for a series of networks, derived from G, as in the proof.

Edge-connectivity algorithm

```
Input G and construct G';
Specify u;

K' = |E|
for all v in V-{u} do
  find F between (u,v) for G';
  if F < K' then K' = F;
endfor
output K';
```

- The overall algorithm requires a polynomial-time complexity.

Why O(n) maximizations?

- Do we need $O(n^2)$ maximizations?
  - for n(n-1) node pairs
- No. O(n) maximizations will suffice.
  - If (P, P') is a cut-set of minimum cardinality, with $u \in P$ and $v \in P'$, then $\kappa' = c_e(u,v)$
  - So, κ' can be found by solving max-flow problem for a particular vertex, say u as the source.
  - The remaining vertices are taken as sink in turn.

Finding vertex-connectivity

- $p_v(u,v)$: Number of vertex-disjoint paths between u and v.
- $c_v(u,v)$: Smallest cardinality of those vertex-cuts which partition the graph, so that:
  - u is in one component
  - v is in the other component.

Theorem: Let G be an undirected graph with $x,y \in V$, and $(x,y) \notin E$ then:
$c_v(u,v) = p_v(u,v)$

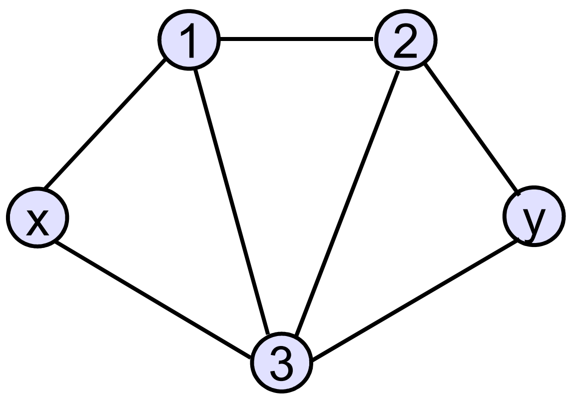
Road to a proof

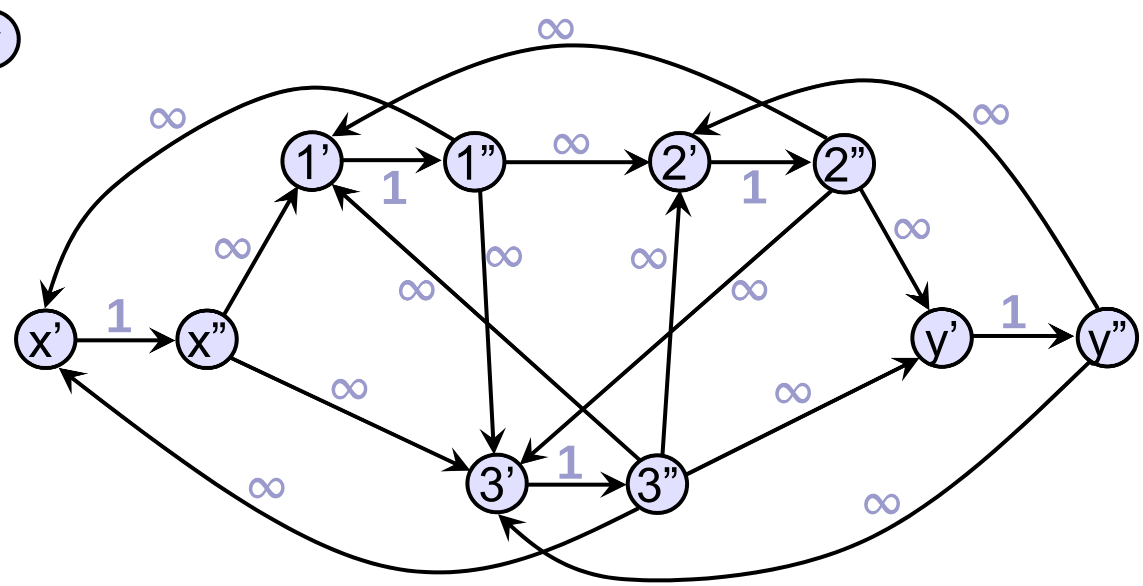
- Given G, construct a digraph G' as follows:
  - For every vertex v of G, create
    - two vertices v', and v"
    - an edge (v',v") called internal edge.
  - For every edge (u,v) of G, create two edges:
    - (u",v') and (v",u')

    called external edges.
- Define a network N, consisting of digraph G',
  - source is x"
  - sink is y'
  - capacity of internal edges = 1
  - capacity of external edges = infinite

Example

- The value of maximum flow in N is:
  $F = c_v(u,v) = p_v(u,v)$





Vertex-connectivity

- The algorithm is based on finding vertex-connectivity of pair of vertices in the graph G'.
- We need to solve the max-flow problem for:
  - v_1 as the source and v_2, v_3, …,v_n as the sinks in turn
  - v_2 as the source and v_3, …,v_n as the sinks in turn
  - …
  - v_{K+1} as the source and v_{K+2}, …,v_n as the sinks in turn
    K: vertex-connectivity found so far.

Vertex-connectivity algorithm

```
Input G and construct G';
K = n;
i = 0;

while K ≥ i do
  i = i+1;
  for j = i+1 to n do
    if (v_i,v_j)∉E then
      find F for (v_i,v_j) in G';
    if F < K then
      K = F;
  endfor
endwhile
output K;
```

Minimum-cost flows

- Most fundamental network flow problem.
- Determine:
  - a least cost shipment of a commodity through a network
  - to satisfy demands at certain nodes
  - from available supplies at other nodes.
- Few example applications:
  - Distribution of a product
  - Flight scheduling
  - Job scheduling with flexible deadlines

Minimum-cost flows

- Special cases of minimum-cost flows:
  - Shortest-path problems
    - Arc costs, but no arc capacities
  - Maximum-flow problem
    - Arc capacities, just simple, equal arc costs

Notation

- G = (N,A) a directed network
  - c_{ij} : cost of arc (i,j)
  - u_{ij} : capacity of arc (i,j)
  - $b(i)$: supply(+) or demand(-) of node i
- Problem definition:

Minimize: $z(x) = \sum_{(i,j)\in A} c_{ij} x_{ij}$ x_{ij} : flow variables

subject to:

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b(i) \quad \forall i \in N$$

$$0 \le x_{ij} \le u_{ij} \quad \forall (i,j) \in A$$

Assumptions

- All data are integral.
  - cost, supply/demand, capacity
- The network is directed.
- The supplies/demands at nodes satisfy:

$$\sum_{i \in N} b(i) = 0$$

- All costs are nonnegative.

Residual network

- G(x): Residual network corresponding to flow x.
- Replace each arc (i,j) by two arcs:
  - (i,j) with cost c_{ij}, residual capacity $r_{ij} = u_{ij} - x_{ij}$
  - (j,i) with cost $-c_{ij}$, residual capacity $r_{ij} = x_{ij}$
  - G(x) consists only of arcs with positive residual capacity.

Cycle-canceling algorithm

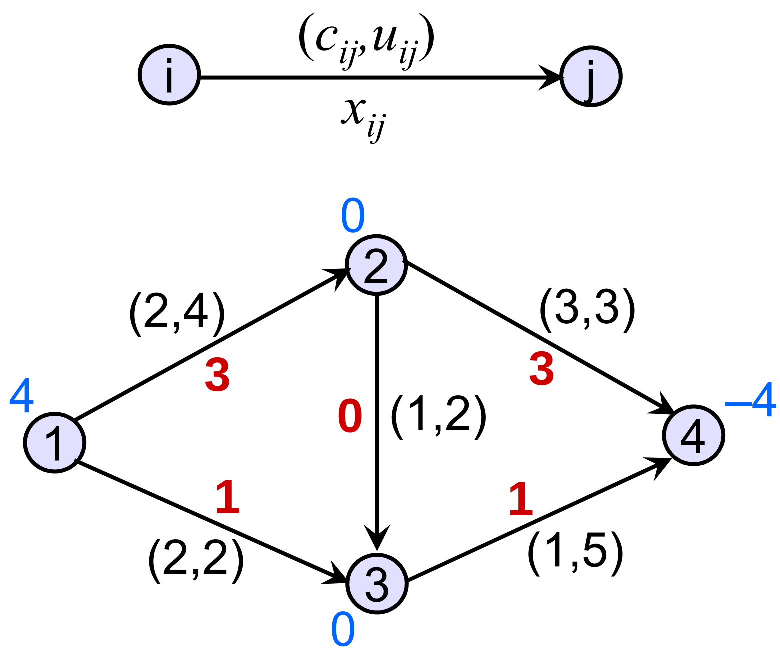
- A simple approach.
- Maintains a feasible solution.
- At every iteration, attempts to improve its objective value.
- First establishes a feasible flow x, by solving maximum flow problem.
- Then, iteratively:
  - finds negative cost directed cycles, and
  - augment flows along these cycles.
- Terminates when the residual network contains no negative cycle.

Cycle-canceling algorithm

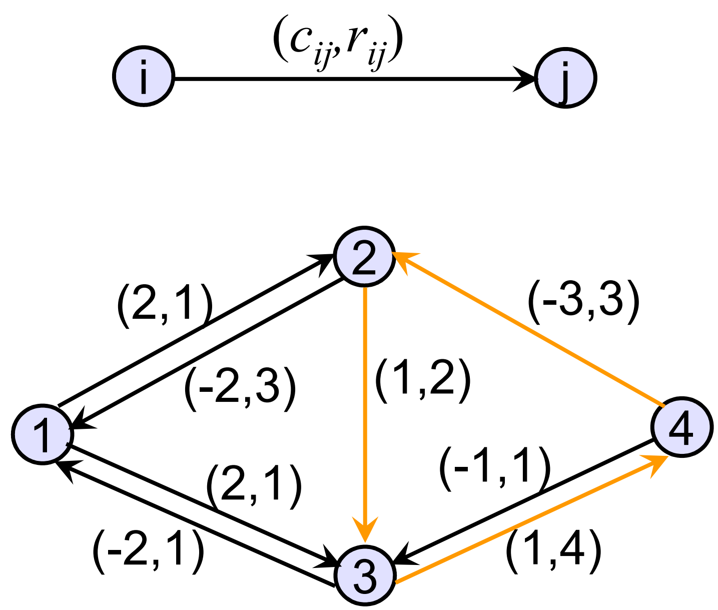
```
Find a feasible flow x in the network;

while G(x) contains a negative cycle do
  Use an algorithm to find a negative cycle W;
  D = min{r_ij: (i,j)∈ W};
  Augment D units of flow in the cycle W;
  Update G(x);
endwhile
```

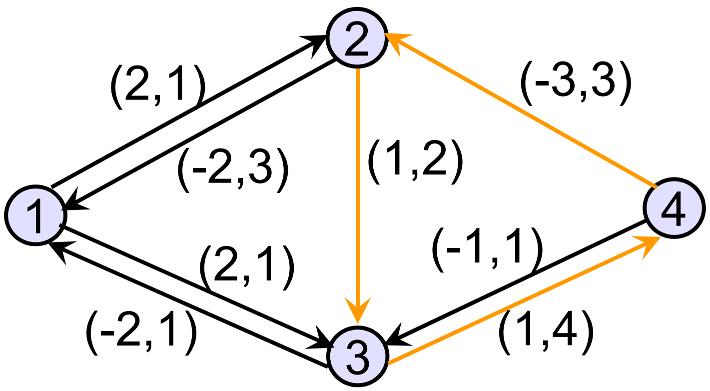
Example

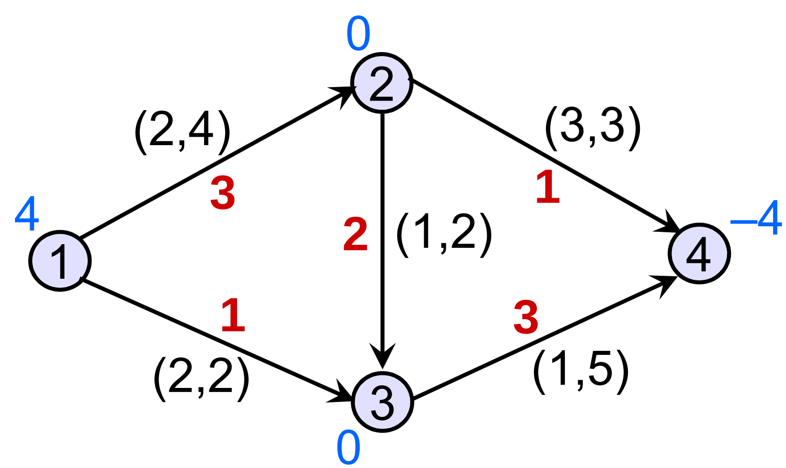


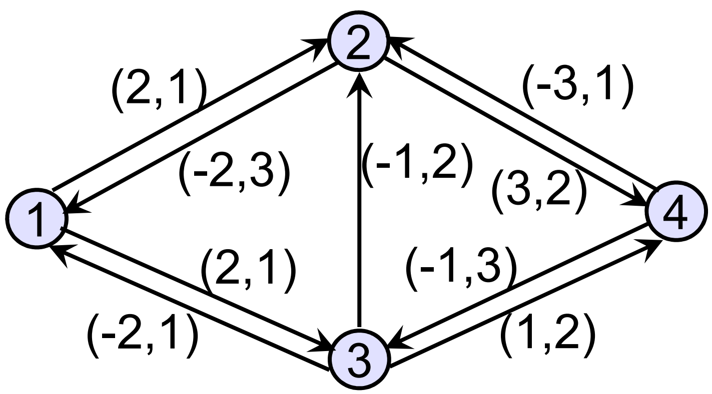
A network with a feasible flow.



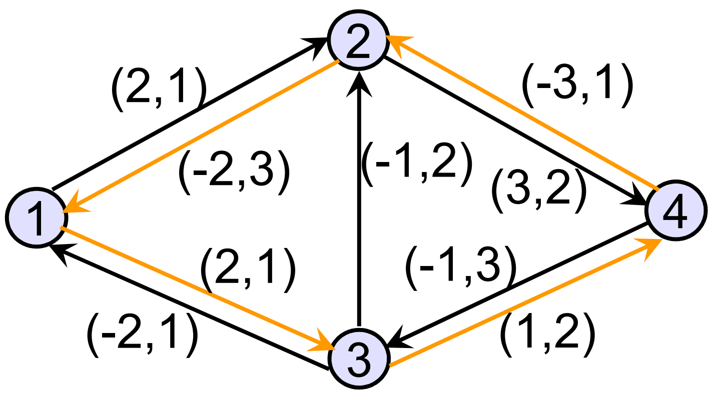
The residual network.
D = 2


1
2
3
4
(2,1)
(-2,3)
(1,2)
(-3,3)
(2,1)
(-2,1)
(-1,1)
(1,4)


1
2
3
4
4
0
0
–4
(2,4)
3
(3,3)
1
2
(1,2)
1
(2,2)
3
(1,5)

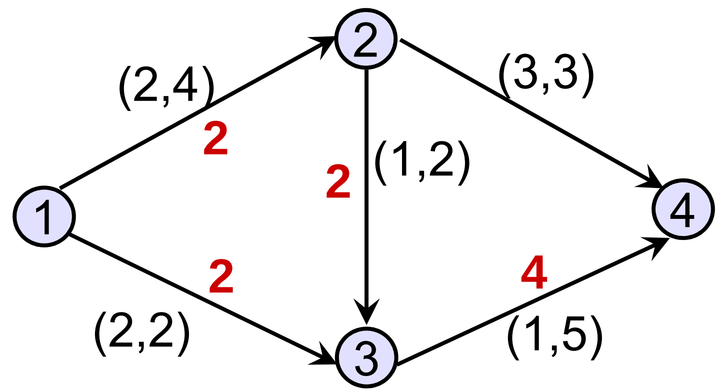

1
2
3
4
(2,1)
(-2,3)
(-3,1)
(-1,2)
(3,2)
(2,1)
(-2,1)
(-1,3)
(1,2)

Example

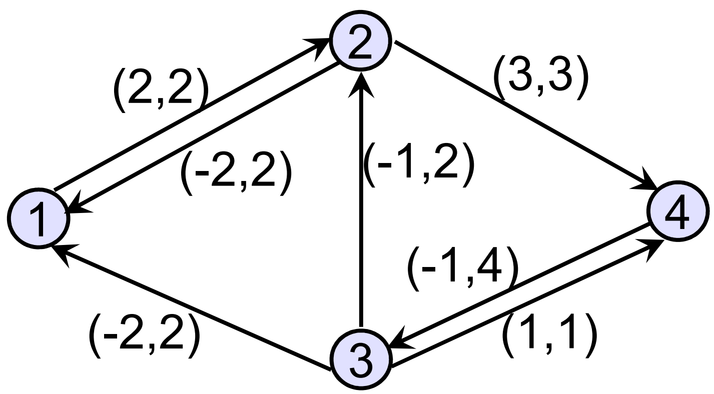


1
2
3
4
(2,1)
(-2,3)
(-1,2)
(-3,1)
(3,2)
(2,1)
(-1,3)
(-2,1)
(1,2)

D = 1



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



1
2
3
4
(2,2)
(-2,2)
(-1,2)
(3,3)
(-1,4)
(-2,2)
(1,1)

Successive Shortest Path

- Maintains optimality of the solution at each step.
- The intermediate solutions
  - maintain the capacity constraint, but
  - violates the mass balance constraint.
- At each step, the algorithm:
  - selects a node s with excess supply
  - selects a node t with unfulfilled demand
  - sends flow from s to t along a shortest path in the residual network.
- Terminates, when node balance constraints are achieved.

Pseudoflow

- For any pseudoflow x, we define the imbalance of a node i:

$$e(i) = b(i) + \sum_{j:(j,i)\in A} x_{ji} - \sum_{j:(i,j)\in A} x_{ij} \quad \forall i \in N$$

  - If e(i) > 0, refer e(i) as the excess of i
  - If e(i) < 0, refer -e(i) as the deficit of i
  - If e(i) = 0, node i is balanced.
- E: Set of excess nodes
- D: Set of deficit nodes
- Notice:

$$\sum_{i\in N} e(i) = \sum_{i\in N} b(i) = 0 \quad \text{and} \quad \sum_{i\in E} e(i) = -\sum_{i\in D} e(i)$$

Notations

- If the network contains an excess node, it must also contain a deficit node.
- Residual network is defined the same way.
- Node potentials π, are used to maintain non-negative arc lengths.
- Reduced cost:

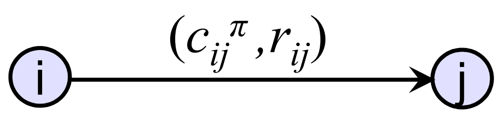
$$c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$$

- d(i,j): distance of nodes i and j.

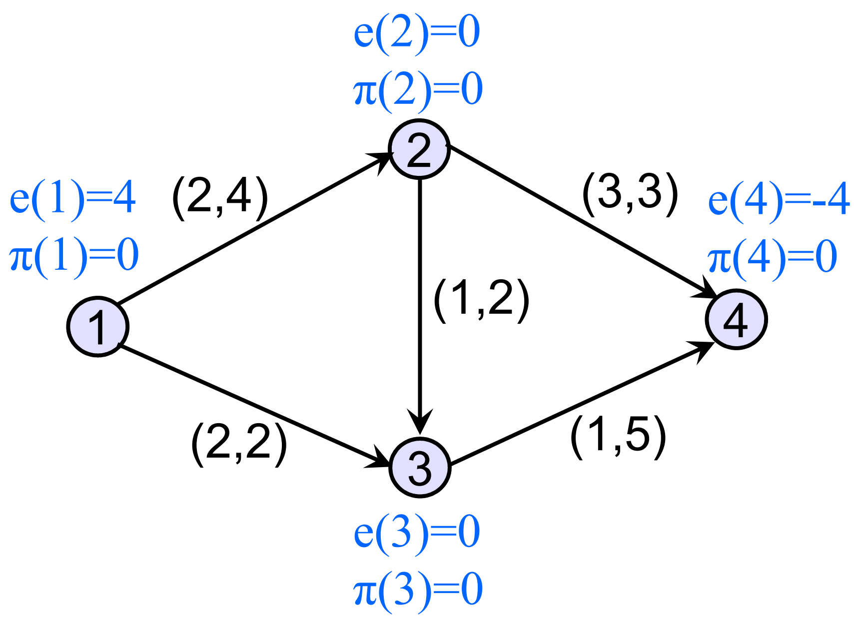
Successive shortest path algorithm

```
for all edges do x(i,j) = 0;
for all nodes do
  π(i) = 0;
  e(i) = b(i);
endfor
initialize the sets:
  E = {i | e(i) > 0} and D = {i | e(i) < 0}
while E ≠ Ø do
  select nodes k ∈ E and l ∈ D;
  determine shortest paths from k to all nodes using reduced
                                                         costs;
  Let P = shortest (k,l)-path;
  for all i do π(i) = π(i) - d(i);
  for all (i,j) do update reduced costs;
  D = min{e(k), -e(l), min{r_ij: (i,j)∈ P}};
  Augment D units of flow along P;
  Update x,G(x),E,D, and reduced costs;
endwhile
```

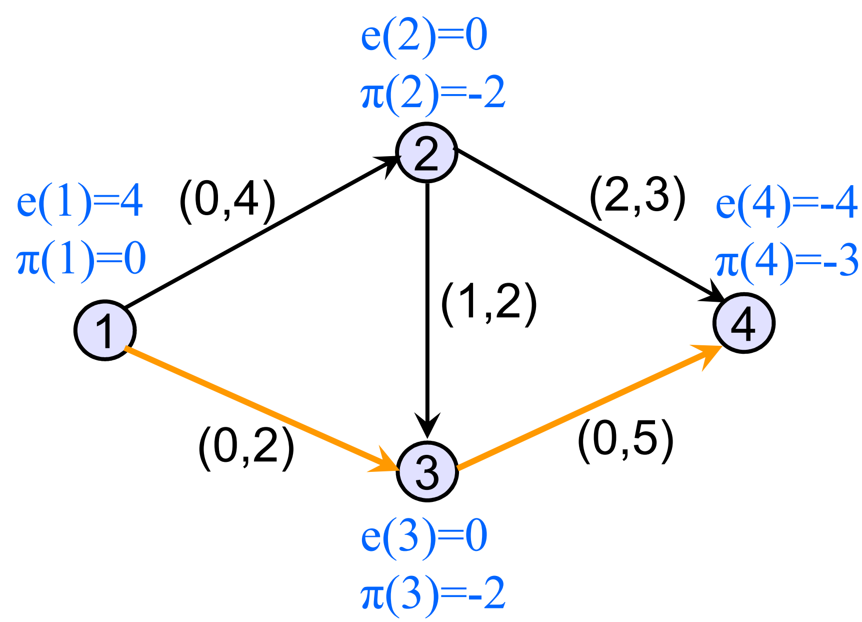
Example



i
j
(c_{ij}^{π}, r_{ij})

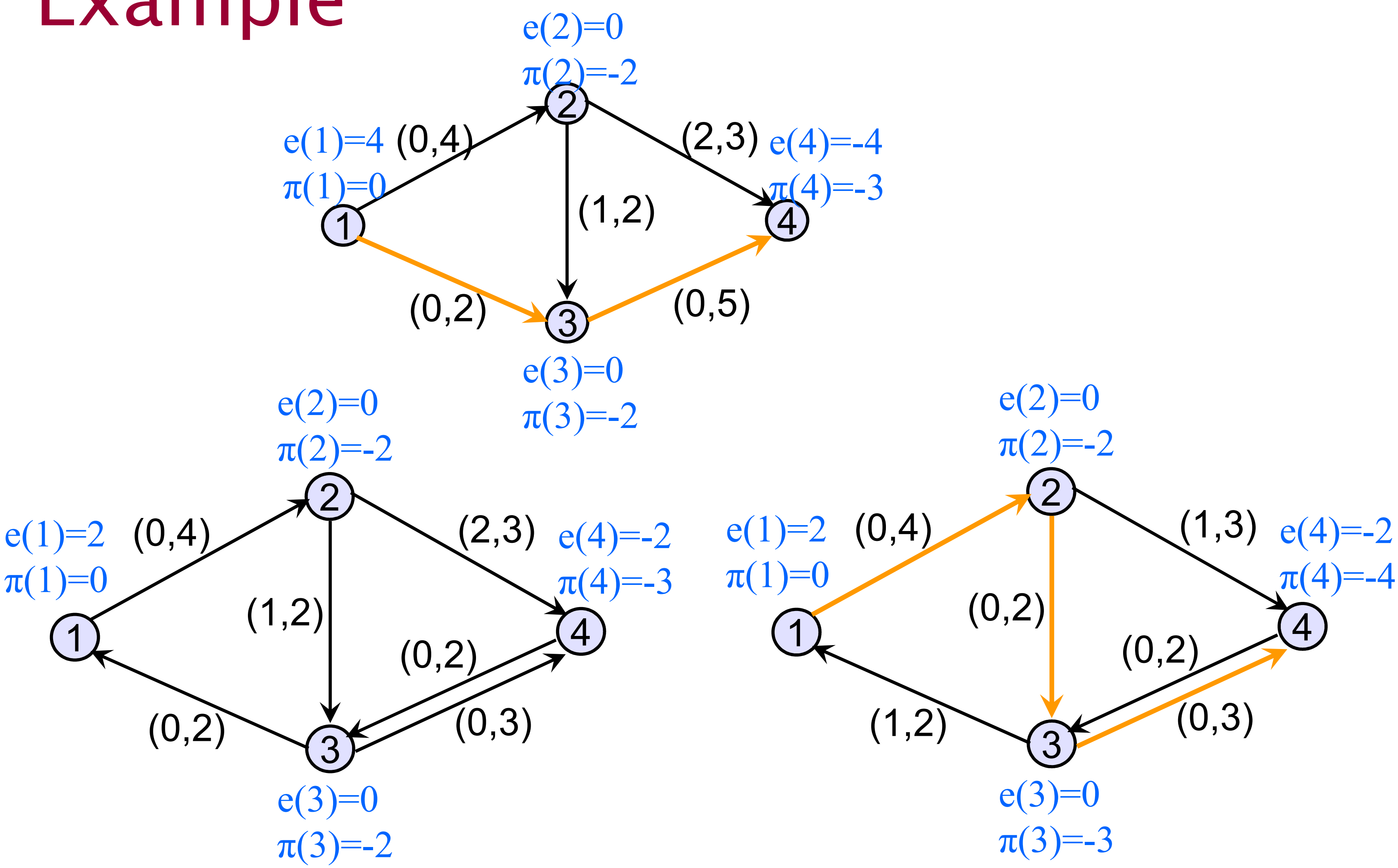


e(2)=0
π(2)=0
e(1)=4
π(1)=0
(2,4)
(3,3)
e(4)=-4
π(4)=0
(1,2)
(2,2)
(1,5)
e(3)=0
π(3)=0
1
2
3
4

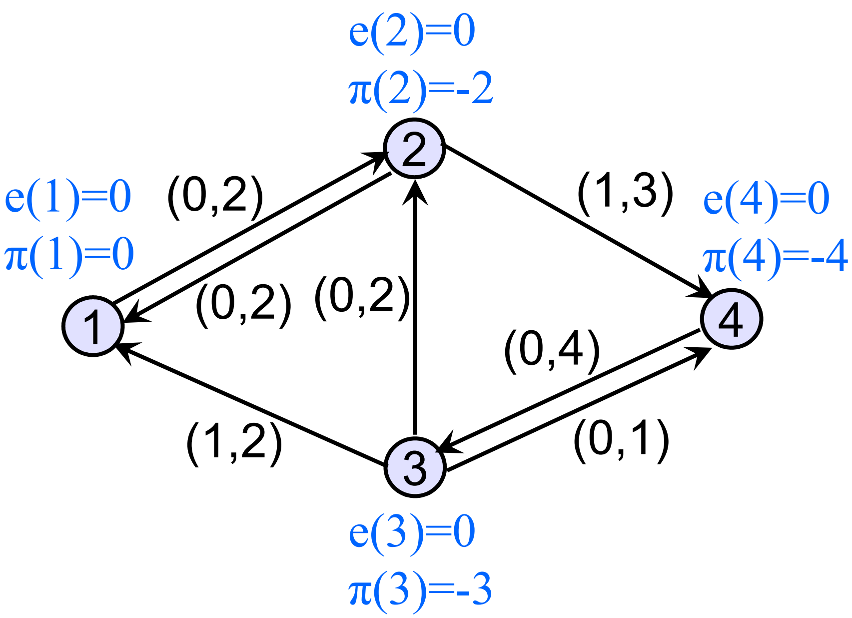


e(2)=0
π(2)=-2
e(1)=4
π(1)=0
(0,4)
(2,3)
e(4)=-4
π(4)=-3
(1,2)
(0,2)
(0,5)
e(3)=0
π(3)=-2
1
2
3
4

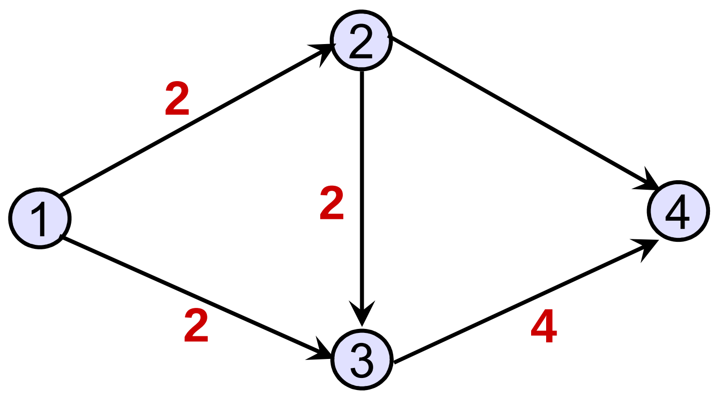
Example


e(2)=0
π(2)=-2
e(1)=4
π(1)=0
(0,4)
(2,3)
e(4)=-4
π(4)=-3
(1,2)
1
2
3
4
(0,2)
(0,5)
e(3)=0
π(3)=-2
e(2)=0
π(2)=-2
e(1)=2
π(1)=0
(0,4)
(2,3)
e(4)=-2
π(4)=-3
(1,2)
(0,2)
(0,2)
(0,3)
e(3)=0
π(3)=-2
e(2)=0
π(2)=-2
e(1)=2
π(1)=0
(0,4)
(1,3)
e(4)=-2
π(4)=-4
(0,2)
(0,2)
(1,2)
(0,3)
e(3)=0
π(3)=-3

Example

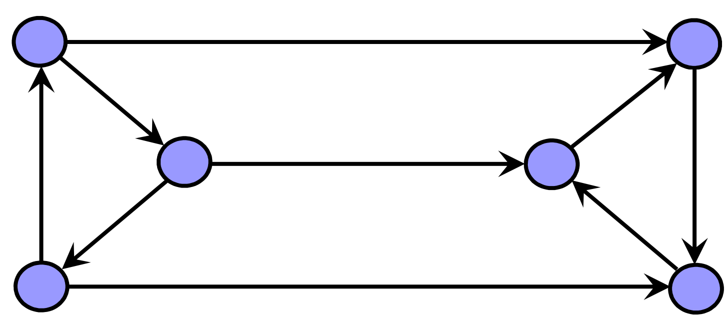


Final solution:



Chinese postman problem in digraphs

- If the digraph is connected and balanced, then the solution is a directed Euler circuit.
- If the graph is not Eulerian we need another method to solve the problem.
- Not all connected digraphs contain a solution.



Theorem: A digraph has a Chinese postman's tour iff it is strongly connected.

Chinese postman in digraphs

- A postman’s circuit for non-eulerian digraph involves repeated edges.
- Number of times that the edge (u,v) is repeated: r(u,v)
- G”: the digraph obtained by adding r(u,v) copies of each edge.
- A postman’s circuit in G corresponds an Euler circuit in G”.
- Repeated edges must form paths between vertices whose in-degree is not equal to their out-degree.

Chinese postman in digraphs

- For any such path:
  - $d^-(u) - d^+(u) = D(u) > 0$
  - $d^-(v) - d^+(v) = D(v) < 0$
  - If $D(u) > 0$, then $D(u)$ paths of repeated edges must start from u.
  - If $D(v) < 0$, then $-D(v)$ paths must end at v.
- The problem reduces to:
  Choosing a set of paths such that G" is balanced.

Solution using flows

- Each vertex u, for which D(u) > 0, can be thought as a source.
- Each vertex v, for which D(v) < 0, can be thought as a sink.
- A path from u to v can be thought as:
  - A unit flow
  - with a cost equal to the sum of the edge-weights.
- We wish to send:
  - D(u) units of flow from u
  - -D(v) units of flow to v
  - At minimum cost.

Solution

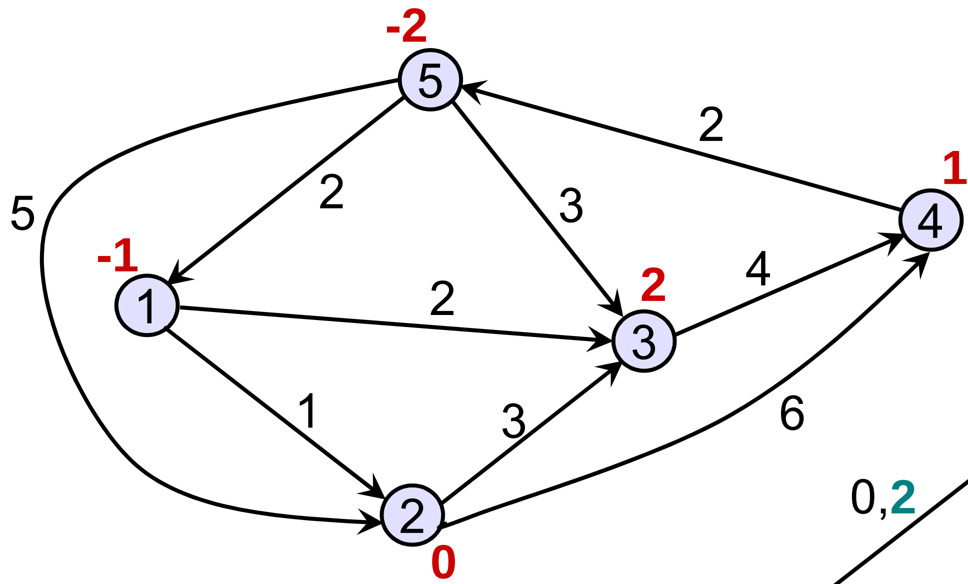
- Single source X:
  - An edge from X to a source u
  - capacity = +D(u)
  - cost = 0
- Single sink Y:
  - An edge from a sink v to Y
  - capacity = -D(v)
  - cost = 0
- All other edges have capacity = infinity

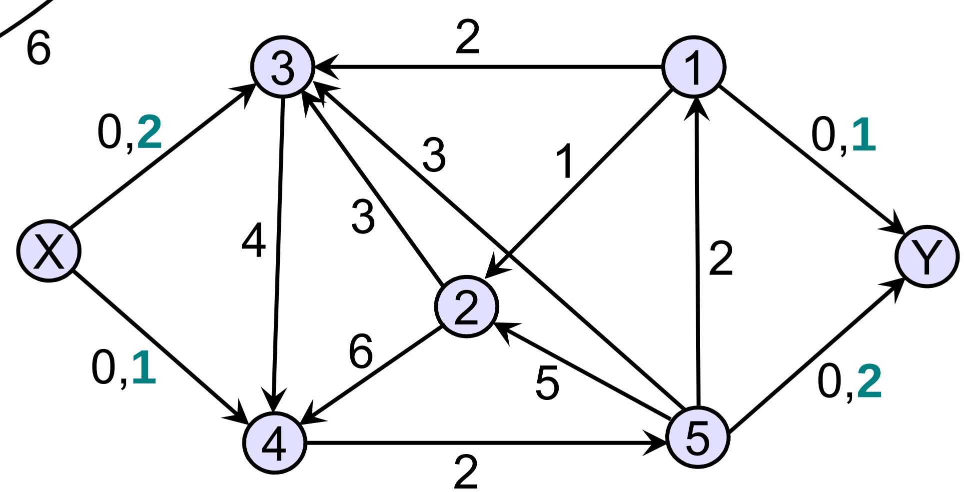
Algorithm

```
Construct network G′;
Find a maximum flow at minimum cost in G′;
Construct G″;
Find an Eulerian circuit of G″;
```

- Eulerian circuit of G” is a minimum-weight postman’s circuit of G.

Example

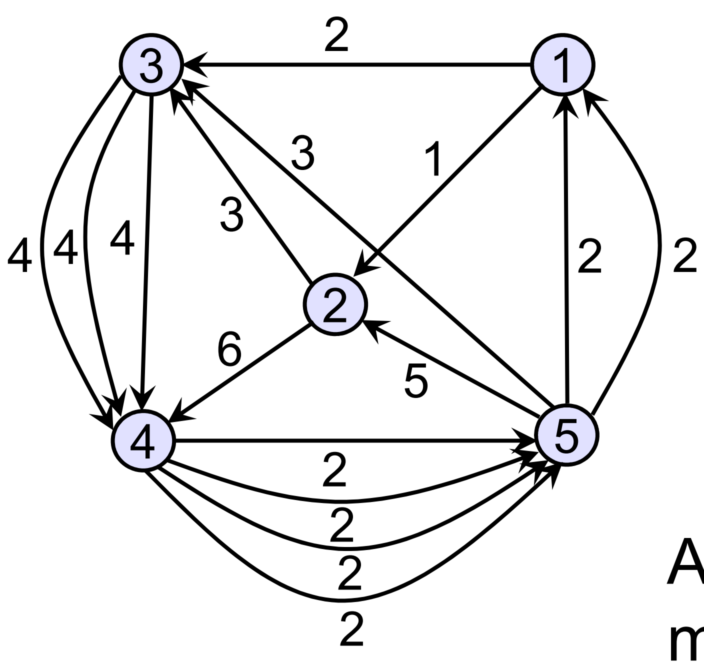




Maximum flow at minimum cost is:

- 2 units along (X,3,4,5,Y)
- 1 unit along (X,4,5,1,Y)

Example



An Eulerian circuit of G” and a minimum cost postman’s circuit of G:
(1,2,3,4,5,2,4,5,3,4,5,1,3,4,5,1)