Operations Research (Master's Degree Course)

7.3 Problems on Graphs: Flows in Networks

Silvano Martello
DEI "Guglielmo Marconi", Università di Bologna, Italy

Flows in Networks: The Maximum Flow Problem

- **Network =** Directed graph $G = (V, A)$ having **capacities** associated with the arcs:

$$q_{ij} = \textbf{capacity} \text{ of } (v_i, v_j) \text{ (integer).}$$

- Arcs are seen as "pipes" in which some material can "flow": q_{ij} = maximum flow in (v_i, v_j).
- **Problem: given $s, t \in V$ (*source* and *sink*), send the maximum flow from s to t.**
- Example: road map, with q_{ij} expressed in number of vehicles/hour; find the maximum quantity of traffic that is possible between two locations (and detect bottleneck roads).
- Given $G = (V, A)$ with *capacities* q_{ij}, a set of values

$$\xi_{ij} \; (= \text{flow in } (v_i, v_j)\text{), } i, j = 1, \ldots, n)$$

is called a **feasible flow of value z from s to t** if

$$0 \leq \xi_{ij} \leq q_{ij} \;\forall\, (v_i, v_j) \in A; \qquad (\alpha)$$

$$\sum_{v_j \in \Gamma^+(v_i)} \xi_{ij} - \sum_{v_k \in \Gamma^-(v_i)} \xi_{ki} = \begin{cases} +z & \text{if } v_i = s \\ -z & \text{if } v_i = t \\ 0 & \text{otherwise} \end{cases} \qquad \forall v_i \in V \qquad (\beta)$$

- An **s-t cut** is a partition of V in $(V_1, V_2) : s \in V_1,\ t \in V_2$.
- The **value of an s-t cut** is $\displaystyle\sum_{(v_i, v_j)\ :\ v_i \in V_1, v_j \in V_2} q_{ij}$.

Max-flow min-cut theorem (Ford-Fulkerson, 1956)

The value z of the maximum flow from s to t is equal to the minimum value of an s-t cut.

Proof z cannot obviously be greater than the value of any s-t cut. We will show, through a **constructive proof** that a flow with such value exists.

- Given a feasible flow ξ_{ij}, let us execute the following procedure:

  $V_1 := \{s\}$;

  while $\exists\ v_i \in V_1$ e $v_j \notin V_1 : \xi_{ij} < q_{ij}$ or $\xi_{ji} > 0$ **do** $V_1 := V_1 \bigcup \{v_j\}$.

- **Case 1:** $t \in V_1$: $\Rightarrow$ $\exists$ sequence of arcs connecting s to t

  ($s \longrightarrow v_i \longrightarrow v_j \longleftarrow \bigcirc \longrightarrow v_l \longleftarrow v_k \longrightarrow t$

  $\bigcirc \longrightarrow \bigcirc$ *forward arc* ; $\bigcirc \longleftarrow \bigcirc$ *backward arc*)

  such that: $\xi_{ij} < q_{ij}\ \forall$ forward arc (v_i, v_j) **and** $\xi_{kl} > 0\ \forall$ backward arc (v_k, v_l).

- Such a sequence is called an **augmenting chain**. Indeed:

  $$\left.\begin{array}{lcll} \delta_1 & = & \min\{(q_{ij} - \xi_{ij}) : & (v_i, v_j) \text{ is a forward arc}\} \\ \delta_2 & = & \min\{\xi_{kl} : & (v_k, v_l) \text{ is a backward arc}\} \end{array}\right\} \delta = \min(\delta_1, \delta_2).$$

  adding (resp. subtracting) δ flow units to each forward (resp. backward) arc, new flow that

  - satisfies (α) $\Leftarrow$ definition of δ
  - satisfies (β) $\Leftarrow$ $\xrightarrow{+\delta}\bigcirc\xrightarrow{+\delta}$ $\xleftarrow{-\delta}\bigcirc\xleftarrow{-\delta}$ $\xrightarrow{+\delta}\bigcirc\xleftarrow{-\delta}$ $\xleftarrow{-\delta}\bigcirc\xrightarrow{+\delta}$
  - has a value increased by δ units.

- New flow $\rightarrow$ new augmenting chain ... until:

- **Case 2:** $t \notin V_1$: $\Rightarrow \begin{cases} \xi_{ij} = q_{ij} & \forall\ (v_i, v_j)\ :\ v_i \in V_1, v_j \in V_2 = V \setminus V_1 \\ \xi_{kl} = 0 & \forall\ (v_k, v_l)\ :\ v_k \in V_2, v_l \in V_1 \end{cases} \Rightarrow$

  flow value $= \displaystyle\sum_{(v_i,v_j)\ :\ v_i \in V_1, v_j \in V_2} \underbrace{\xi_{ij}}_{=q_{ij}} - \sum_{(v_k,v_l)\ :\ v_k \in V_2, v_l \in V_1} \underbrace{\xi_{kl}}_{=0}$

  $=$ value of an s-t cut $\Rightarrow$ optimum.
- At each iteration of Case 1 the flow increases by at least one unit $\Rightarrow$ convergence. $\square$

Number of iterations bounded by any upper bound on z, e.g., $U = \displaystyle\sum_{v_j \in \Gamma^+(s)} q_{sj} \Rightarrow$

Time: $O(T \cdot U)$ (T= time needed by an iteration).

Algorithm outline:

- start with a feasible flow (e.g., $\xi_{ij} = 0\ \forall\ i, j$), and increase it through augmenting chains; when no augmenting chain exists, we have a maximum flow.
- For finding the augmenting chains we attach **labels** to the vertices. A vertex can be
  - **unlabeled**;
  - **labeled** ($\Leftrightarrow \in V_1$) and **unscanned**;
  - **labeled** and **scanned** ($\Leftrightarrow$ used for trying to increase V_1).
- the label of vertex v_i has the form $\begin{cases} [+v_k, \delta] & \Leftrightarrow \quad \xi_{ki} \text{ can be increased;} \\ [-v_k, \delta] & \Leftrightarrow \quad \xi_{ik} \text{ can be decreased;} \end{cases}$

  with δ = maximum additional flow that can be sent from s to v_i.

Ford-Fulkerson Algorithm

procedure MAX_FLOW:
begin
  for $i := 1$ **to** n **do** **for** $j := 1$ **to** n **do** $\xi_{ij} := 0$;
  $opt :=$ false;
  while $opt =$ false **do**
    begin
      label s by $[+s, +\infty]$;
      repeat
        let v_i be a labeled $([\pm v_k, \delta(v_i)])$ unscanned vertex;
        for each $v_j \in \Gamma^+(v_i) : v_j$ is unlabeled **and** $\xi_{ij} < q_{ij}$ **do**
          label v_j by $[+v_i, \min(\delta(v_i), q_{ij} - \xi_{ij})]$;
        for each $v_j \in \Gamma^-(v_i) : v_j$ is unlabeled **and** $\xi_{ji} > 0$ **do**
          label v_j by $[-v_i, \min(\delta(v_i), \xi_{ji})]$;
        mark v_i as scanned
      until t is labeled **or** no new vertex can be labeled;
      if t is unlabeled **then** $opt :=$ true
      else

...

Ford-Fulkerson Algorithm (cont'd)

...

```
        if t is unlabeled then opt := true
        else
            begin
                δ* := δ(t);  x := t;
                repeat
                    if the label of x is [+y, δ(x)] then ξ_yx := ξ_yx + δ*
                    else (i.e., the label is [−y, δ(x)]) ξ_xy := ξ_xy − δ*;
                    x := y
                until x = s;
                cancel all labels
            end
    end;
  comment: the minimum cut is given by V_1 = {v_i : v_i is labeled},  V_2 = V \ V_1
end.
```

In the **course web page**: **applet** for executing the Ford-Fulkerson algorithm.

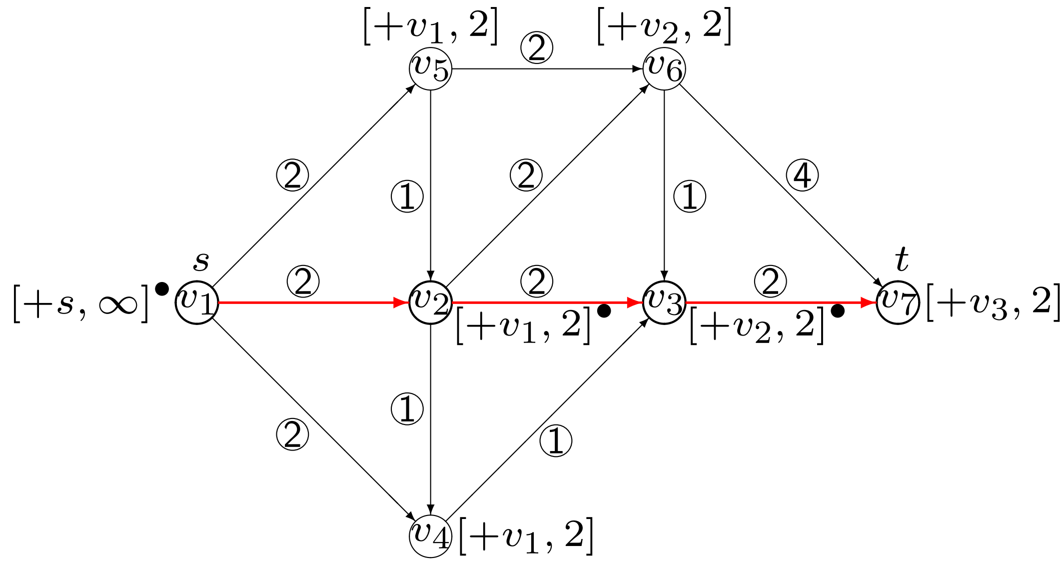
Remind: Number of iterations bounded by any upper bound on z, e.g., $U = \sum_{v_j \in \Gamma^+(s)} q_{sj} \Rightarrow$

time: $O(T \cdot U)$ (T= time needed by an iteration), **pseudo-polynomial**

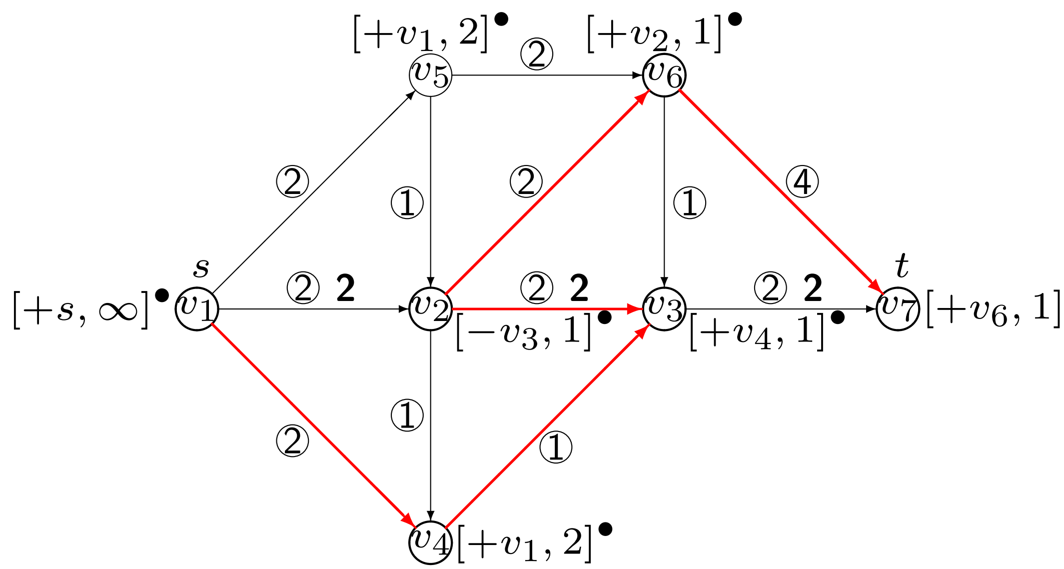
With this implementation, one iteration takes $O(n^2)$ time $\Rightarrow$ **overall time $O(n^2 U)$**.

Example: $s = v_1$, $t = v_7$. Capacities within circles. Start with nil flow.
Vertices considered by increasing index. Symbol '•' marks the scanned vertices:

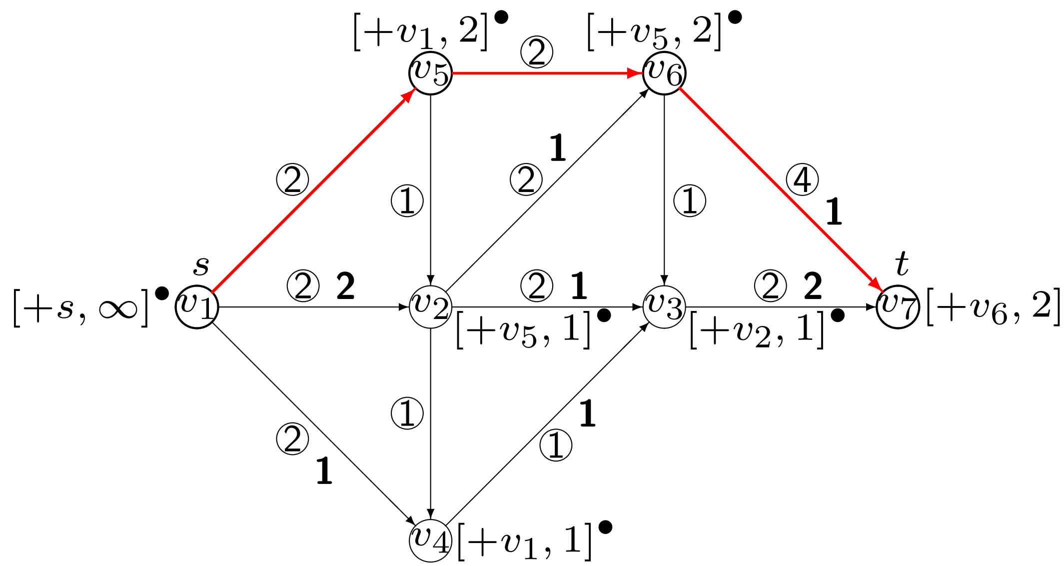
1st iteration:



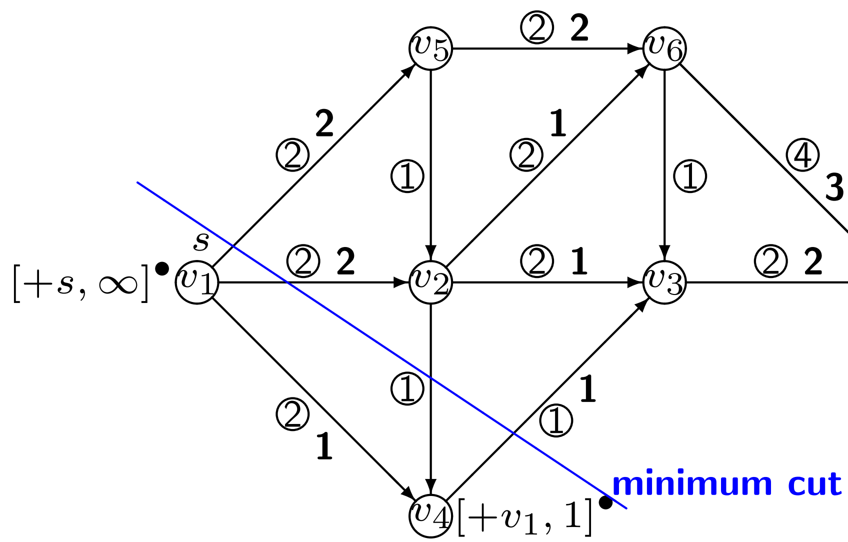
2nd iteration:



3rd iteration:

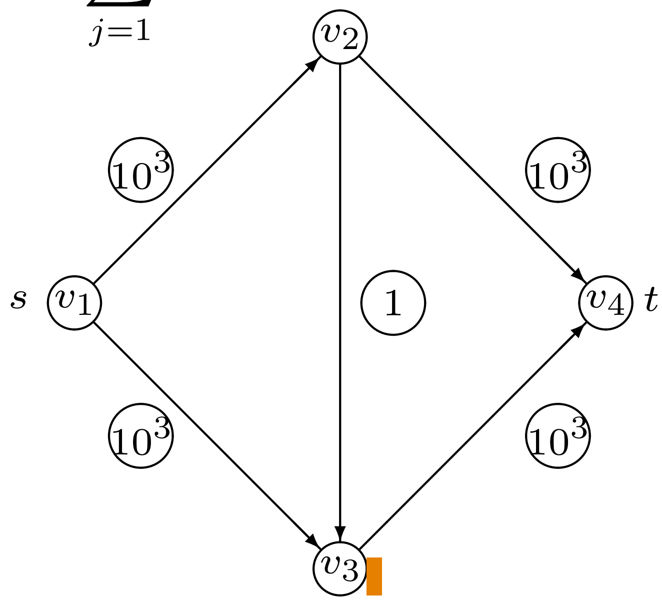


4th iteration:



Complexity of maximum flow

- Ford-Fulkerson algorithm: $O(n^2)$ time per augmenting chain; at most z augmenting chains;
- upper bound on z: $\sum_{j=1}^{n} q_{sj}$ $\Rightarrow$ Complexity $O(n^2 z)$, pseudo-polynomial. **Example:**



Possible sequence:

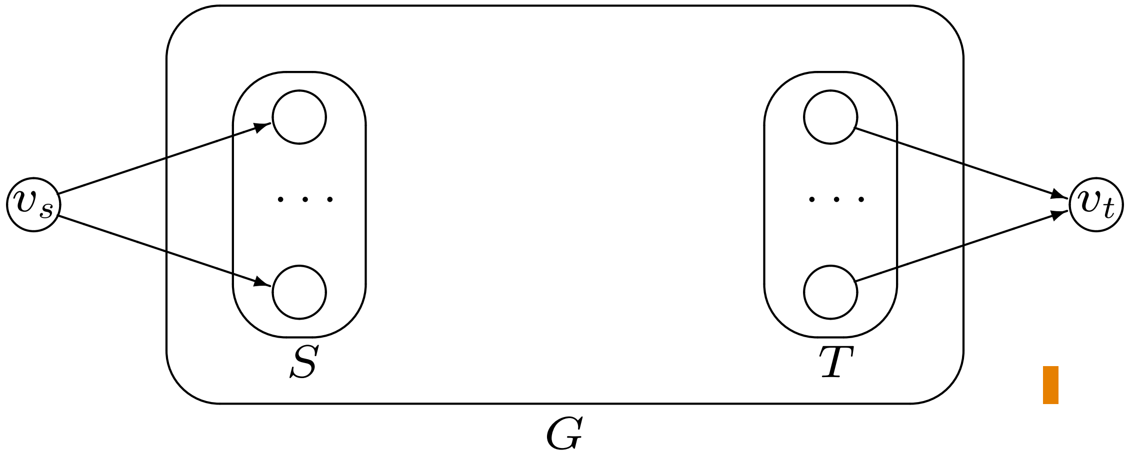
1st augmenting chain:	v_1, v_2, v_3, v_4 ;	$\delta = 1$;
2nd augmenting chain:	v_1, v_3, v_2, v_4 ;	$\delta = 1$;
3rd augmenting chain:	v_1, v_2, v_3, v_4 ;	$\delta = 1$;
. . . (2000 iterations)		

- Edmonds and Karp (1972): by selecting, at each iteration, the **shortest augmenting chain**, at most n^3 iterations; $\Rightarrow$ $O(n^5)$ algorithm;
- Karzanov (1974): $O(n^3)$ algorithm.

Other max-flow problems

Multiple sources and sinks:

- **Problem:** Given $G = (V, A)$, with capacities q_{ij}, and $S,\ T \subset V\ (S \cap T = \emptyset)$, send the maximum total flow from all sources of S to all sinks of T.
- **Solution:** define a new graph $G' = (V', A')$: with
  $V' := V \cup \{v_s, v_t\}$;
  $A' := A \cup \{(v_s, v_i) : v_i \in S\} \cup \{(v_j, v_t) : v_j \in T\}$



  and send the maximum flow from v_s to v_t.
- If there is no additional constraint, **capacities** $q_{si} = q_{jt} = +\infty\ \forall\ i, j$,
  otherwise: q_{si} = limit on the supply at source v_i, q_{jt} = limit on the demand at sink v_j.

Other max-flow problems (cont'd)

Arc and vertex capacities:

- **Problem:** given $G = (V, A)$, with $q_{ij} =$ capacity of arc (v_i, v_j) and $p_i =$ capacity of vertex v_i, and $s,\ t \in V$, send the maximum flow from s to t by satisfying the additional constraint

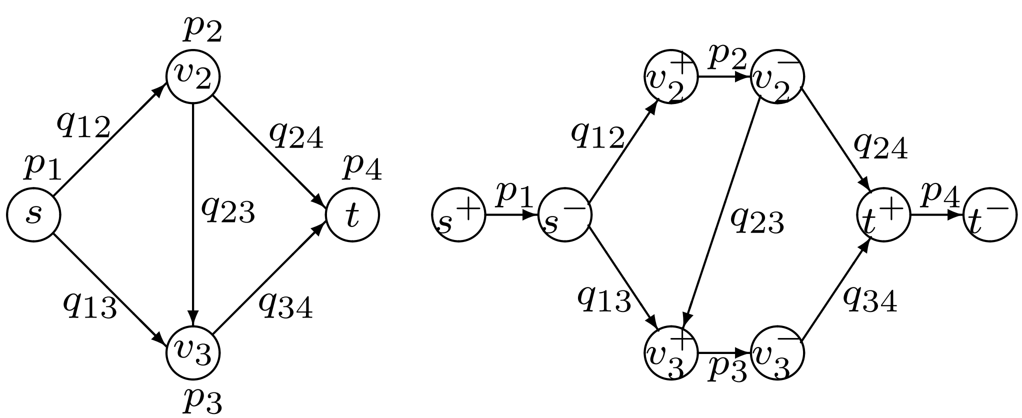
$$\sum_{v_k \in \Gamma^-(v_i)} \xi_{ki} \leq p_i \quad \forall\ v_i \in V \qquad (\gamma).$$

- **Solution:** new graph $G' = (V', A' \cup A'')$ with

  $V' := \{v_i^+ : v_i \in V\} \cup \{v_i^- : v_i \in V\}$;

  $A' = \{(v_i^-, v_j^+) : (v_i, v_j) \in A\}$ with capacity of $(v_i^-, v_j^+) = q_{ij}$;

  $A'' = \{(v_i^+, v_i^-) : v_i \in V\}$ with capacity of $(v_i^+, v_i^-) = p_i$



  and send the maximum flow from s^+ to t^-.

- The total flow entering vertex v_i^+ must travel along arc (v_i^+, v_i^-) $\Rightarrow$ it must satisfy (γ).

Flows in Networks: The Minimum Cost Flow Problem

- **Max-Flow** and **Min-Cost Flow** are the two most relevant flow problems.
- **Network** $G = (V, A)$ having **two** positive integer values associated with each arc (v_i, v_j):
  - **capacity** q_{ij} = maximum flow along (v_i, v_j);
  - **cost** c_{ij} = cost **per unit of flow** along (v_i, v_j).
- **Problem: given** $s, t \in V$, send a **flow of prefixed value** u from s to t at **minimum cost**.
- **Algorithm**: **Phase 1.** find a feasible flow of value u;

  Phase 2. iteratively modify the flow preserving its value and decreasing its cost.
- **Phase 1:**
  - use the Ford-Fulkerson algorithm halting it when the current flow has value u, i.e.,
  - ...

    begin

    $\delta^* := \delta(t)$; $x := t$;

    let z $(z < u)$ be the value of the current flow;

    if $z + \delta^* > u$ **then** $\delta^* := u - z$

    repeat

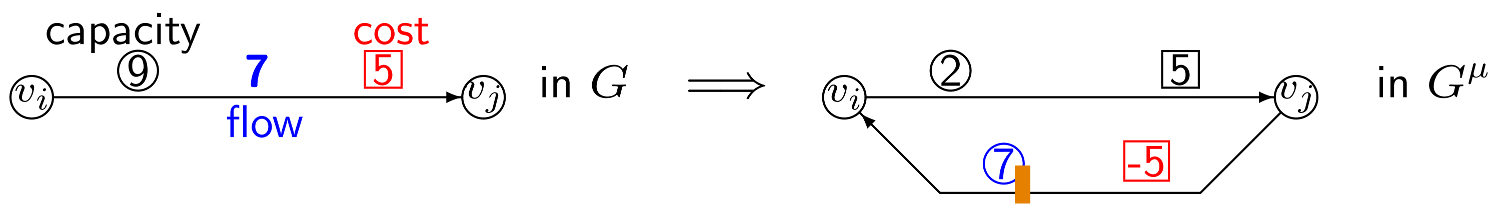
    ...

    until $x = s$;

    if z (updated) $= u$ **then stop else** cancel all labels

The Minimum Cost Flow Problem: Phase 2

- Given a flow ξ_{ij} $(i, j = 1, \dots, n)$ of value u in $G = (V, A)$, let us define the **incremental graph** $G^\mu = (V, A^\mu)$ as:
  - $A^\mu = A^\mu_f \cup A^\mu_b$;
  - $A^\mu_f = \{(v_i, v_j) \in A \;:\; \xi_{ij} < q_{ij}\}$;
  - $A^\mu_b = \{(v_j, v_i) \in A \;:\; \xi_{ij} > 0\}$;
  - $\forall\ (v_i, v_j) \in A^\mu_f$, set capacity $q^\mu_{ij} = q_{ij} - \xi_{ij}$ and cost $c^\mu_{ij} = c_{ij}$;
  - $\forall\ (v_j, v_i) \in A^\mu_b$ set capacity $q^\mu_{ji} = \xi_{ij}$ and cost $c^\mu_{ji} = -c_{ij}$.

- 

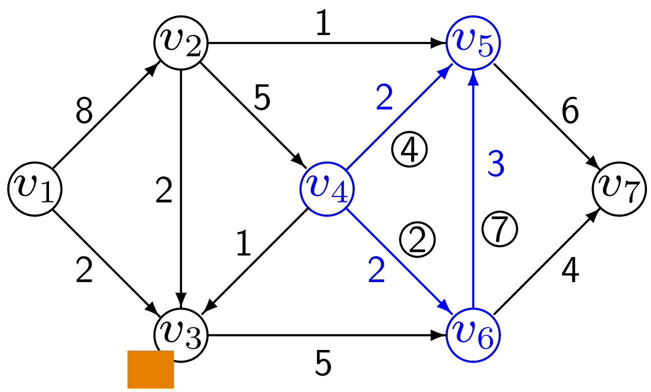
- The incremental graph represents the possibilities of increasing/decreasing the current flow.

- **Observation:**
  - The labeling procedure of the Ford-Fulkerson algorithm can be seen as a method of finding a path in the incremental graph;
  - the **augmenting chain** is a path where the forward arcs correspond to arcs of A^μ_f, and the backward arcs correspond to arcs of A^μ_b.

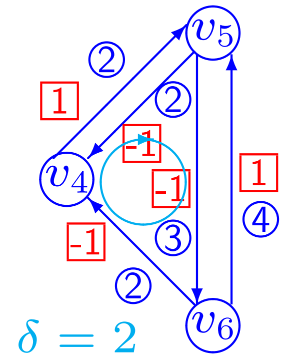
The Minimum Cost Flow Problem: Phase 2 (cont'd)

- **Key observation:** If the incremental graph G^μ contains a **circuit** Φ such that the sum of the arc costs in Φ is **negative** then we can send the maximum possible flow δ around the circuit.

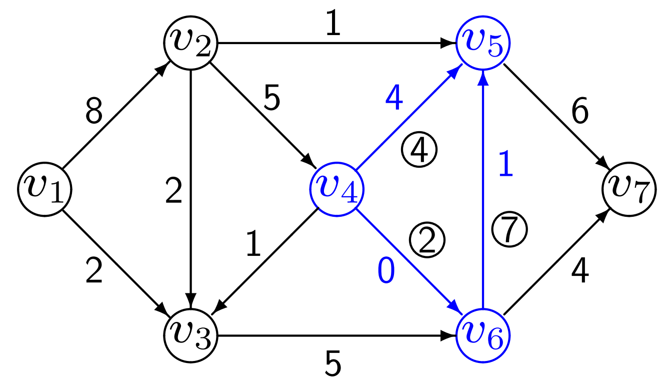
  Example: flow from v_1 to v_7. Assume all costs are $\boxed{1}$.



Flow of value 10
Cost for the blue arcs = 7



In the incremental graph



New flow of value 10
Cost for the blue arcs = 5

- By sending such a flow:
  - the overall flow from the source to the sink remains **unchanged** and **feasible** (the equilibrium at each vertex is preserved);
  - the cost is reduced by δ.

- **Property** *A flow of value u is a minimum cost flow if and only if the incremental graph G^μ contains no circuit such that the sum of the costs in its arcs is negative.*

  Proof The "only if" part is obvious. The "if" part is omitted.

The Minimum Cost Flow Problem: Algorithm

Observation: the incremental graph can contain arcs with negative cost;
a negative cost circuit can be detected by an appropriate $O(n^3)$ shortest path algorithm (modified Dijkstra, or Floyd–Warshall).

procedure Minimum_Cost_Flow:
begin
  use a maximum flow algorithm to find a feasible flow $[\xi_{ij}]$ of value u;
  opt$:=$ false;
  while *opt* $=$ false **do**
    construct the incremental graph G^μ corresponding to the current flow $[\xi_{ij}]$;
    use an appropriate shortest path algorithm to find a negative cost circuit Φ (if any) in G^μ;
    if no such circuit exists **then** *opt*$:=$ true
    else
      begin
        $\delta := \min_{(v_i,v_j)\in\Phi}\{q^\mu_{ij}\}$
        for each $(v_i, v_j) \in \Phi$ with $c^\mu_{ij} < 0$ **do** $\xi_{ji} := \xi_{ji} - \delta$;
        for each $(v_i, v_j) \in \Phi$ with $c^\mu_{ij} > 0$ **do** $\xi_{ij} := \xi_{ij} + \delta$;
      end
  endwhile.
end.

Complexity of the Minimum Cost Flow Problem

- The algorithm Minimum_Cost_Flow is the most simple and elegant approach, but not the most efficient one.
- Each iteration takes $O(n^3)$ time for detecting a negative circuit;
- the number of iterations can be proportional to $Q = \max_{ij}\{q_{ij}\}$ and $C = \max_{ij}\{c_{ij}\}$, and hence the overall time complexity is pseudo-polynomial:

$$O(n^3\, Q\, C).$$

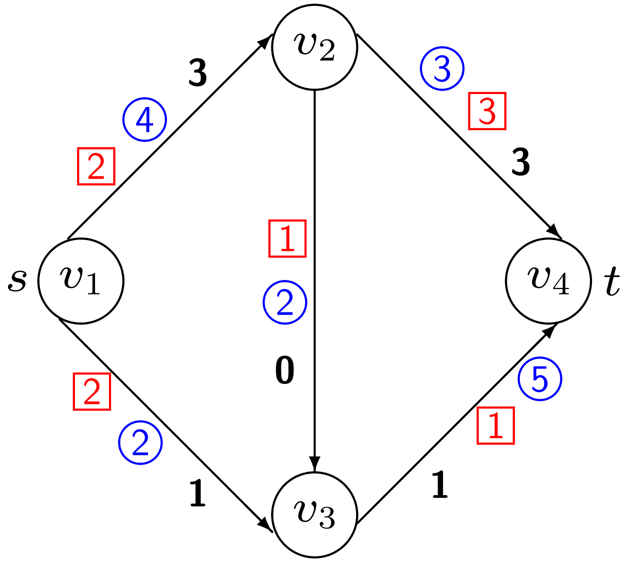
- Other, somehow similar, algorithms are known, that have better pseudo-polynomial time complexity.
- A different family of algorithms, based on scaling techniques, have polynomial time complexity, like, e.g.,

$$O(n^3 \, \log Q \, \log(nC));$$

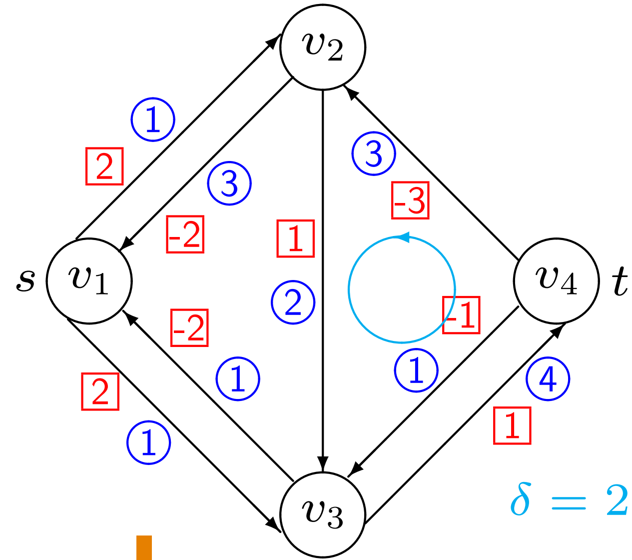
- such time complexities are however called **weakly polynomial**, because they still depend on the magnitude of the input values.
- A third family of (very complicated) algorithms have instead **fully polynomial** time complexity, like, e.g.,

$$O(\,(n^2 \, \log n)\,(n^2 + n \log n)).$$

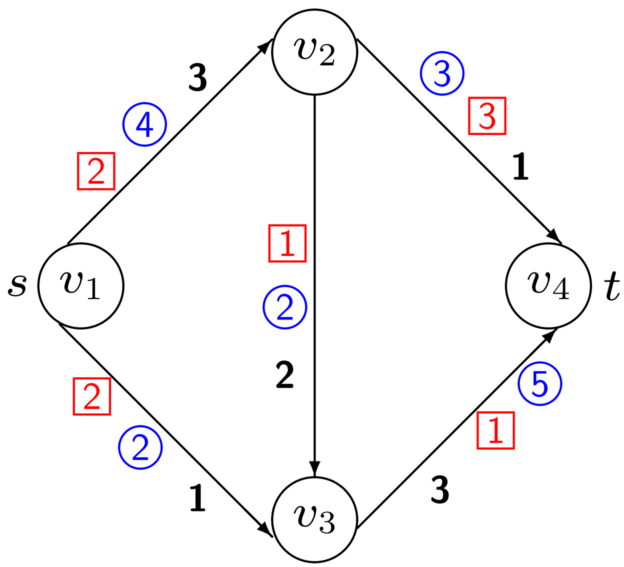
Example



flow of value 4, cost =18

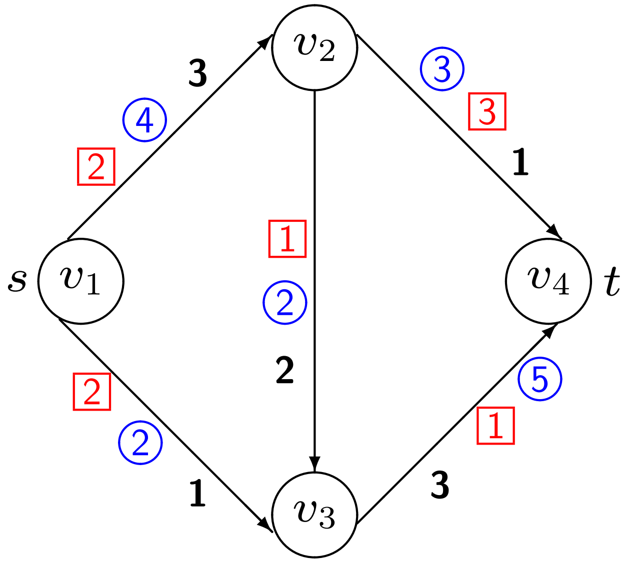


incremental graph

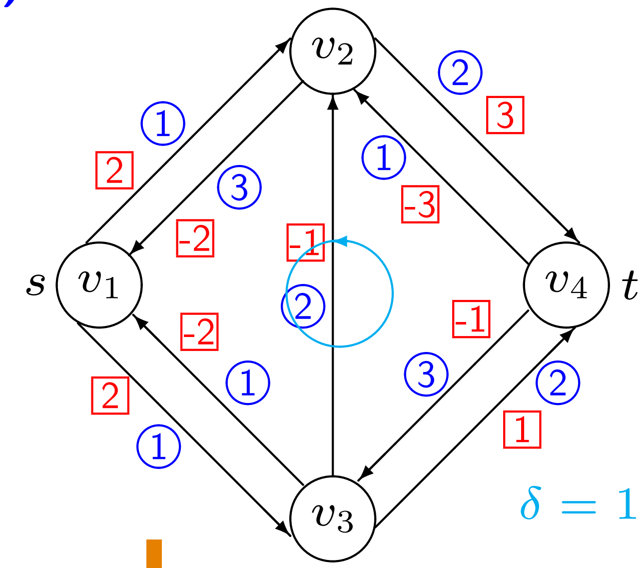


flow of value 4, cost =16

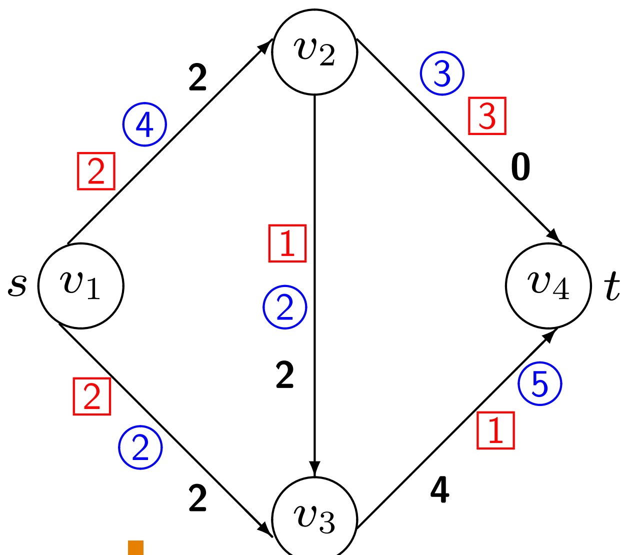
Example (cont'd)



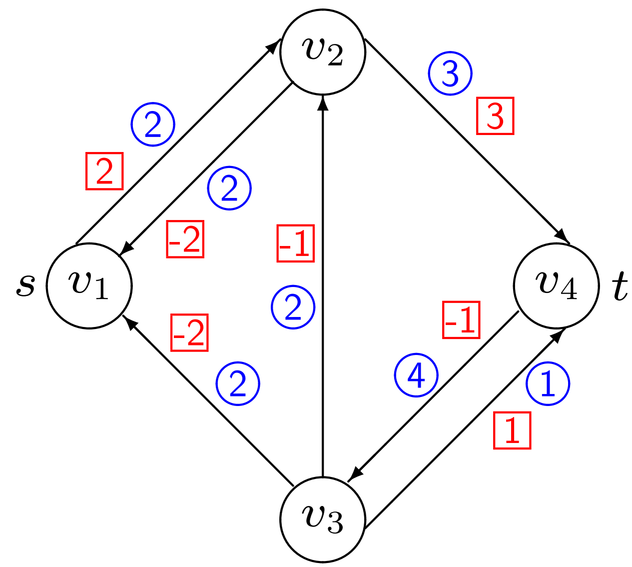
flow of value 4, cost =16



incremental graph



flow of value 4, cost =14



incremental graph
no negative cost circuit

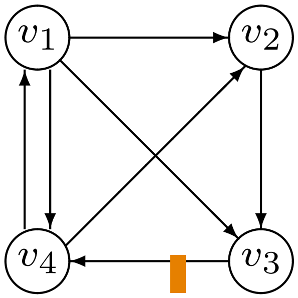
Mathematical models

- A special data structure for representing graphs in mathematical models:
- **Directed graphs:** let a_i $(i = 1, \dots, m)$ be the arcs in the graph.

  Incidence matrix: an $n \times m$ matrix $A = [a_{ij}]$ with

$$a_{ik} = \begin{cases} +1 & \text{if arc } a_k \text{ emanates from vertex } v_i \text{ ;} \\ -1 & \text{if arc } a_k \text{ enters vertex } v_i \text{ ;} \\ 0 & \text{otherwise .} \end{cases}$$

- **Example:**



$$[a_{ij}] = \begin{bmatrix} +1 & +1 & +1 & 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & +1 & 0 & 0 & -1 \\ 0 & -1 & 0 & -1 & +1 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & +1 & +1 \end{bmatrix}$$

Shortest paths and linear programming

- Shortest path from s to t reformulated as:
- send, at minimum cost, a flow of material from s to t:
- ξ_k = quantity of material that flows along arc a_k;
- c_k (length of arc a_k) = cost to be paid for sending one unit of ξ_k along arc a_k;
- shortest s to t path = find a minimum cost flow to send one unit of material from s to t.
- row i of A: a "$+1$" for each arc emanating from v_i, a "-1" for each arc entering v_i:
- flow conservation (β): $a_i'\xi = 0$ for $v_i \neq s, t \Rightarrow$

$$
\text{(P)} \quad \min c'\xi \qquad\qquad A\xi = \begin{array}{c c}
\begin{bmatrix} +1 \\ -1 \\ 0 \\ \dots \\ 0 \end{bmatrix} & \begin{array}{c} s \\ t \\ \\ \\ \\ \end{array}
\end{array} \qquad \xi \geq 0
$$

$$
\text{(D)} \max \pi_s - \pi_t \qquad \pi' A \leq c' \qquad \pi \gtreqless 0 \qquad \Leftrightarrow \qquad A_k = \begin{array}{c c}
\begin{bmatrix} +1 \\ -1 \\ 0 \\ \dots \\ 0 \end{bmatrix} & \begin{array}{c} v_i \\ v_j \\ \\ \\ \\ \end{array}
\end{array}
$$

- (P): a constraint per vertex; (D): a constraint per arc, $\pi_i - \pi_j \leq c_{ij}$= cost of arc (v_i, v_j).
- **Could ξ be fractional?** Yes but: **Intuitively**, different paths must have the same unit cost. **Formally**, is A a TUM matrix?.

A sufficient unimodularity condition

Theorem *An integer matrix A with $a_{ij} \in \{0, +1, -1\}$ $\forall\ i, j$ is TUM if*

1. *no column has more than two non-zero elements,* **and**
2. *the rows can be partitioned into two sets, I_1 and I_2 such that*

- *if a column has two entries of the same sign, their rows are in different sets;*
- *if a column has two entries of different sign, their rows are in the same set.*

Proof By induction. **Base:** any square submatrix of A of order 1 is UM.
Induction: we show that:

if every submatrix of order $k-1$ is UM, then every submatrix of order k is UM.

Let C be a submatrix of order k. Three possibilities exist:

(a) C has a column of all zero entries: then $\det(C) = 0$;

(b) C has a column with a single non-zero entry c_{ij}: then $\det(C) = \pm 1 \cdot$(minor of c_{ij}).
The minor of c_{ij} is a submatrix of order $k-1$, hence UM. It follows that C is UM.

(c) C all columns of C have two non-zero entries: for each column j we have

$$\sum_{i \in I_1} a_{ij} = \sum_{i \in I_2} a_{ij}.$$

Hence a linear combination of the rows of C has value zero, i.e., $\det(C) = 0$. $\square$

Some important consequences

1. *The incidence matrix of a directed graph is TUM.*

   Proof It satisfies the hypothesis with $I_2 = \emptyset$ $\square$.

2. *The shortest path problem can be solved through linear programming.*
   (In addition, it can be proved that the Dijkstra algorithm is equivalent to a specialized version of an algorithm for Linear Programs, known as **Primal-Dual**.)

3. *The constraint matrix of the Assignment Problem* (AP) *is* TUM.

   Proof Constraints $\sum_{j=1}^{n} x_{ij} = 1 (i = 1, \ldots, n)$ and $\sum_{i=1}^{n} x_{ij} = 1 (j = 1, \ldots, n)$ model an LP in standard form with n^2 variables:

$$
\underbrace{\boxed{}}_{A}
\underbrace{\begin{bmatrix} x_{11} \\ x_{12} \\ \cdots \\ x_{1n} \\ \cdots \\ x_{n1} \\ x_{n2} \\ \cdots \\ x_{nn} \end{bmatrix}}_{x}
=
\underbrace{\boxed{}}_{b}
\Rightarrow
\begin{array}{c|cccccccccc|}
 & x_{11} & \ldots & x_{1n} & x_{21} & \ldots & x_{2n} & \ldots & x_{n1} & \ldots & x_{nn} \\
\hline
1 & 1 & \ldots & 1 & & & & & & & \\
 & & & & 1 & \ldots & 1 & & & & \\
 & & & & & & & \ldots & & & \\
n & & & & & & & & 1 & \ldots & 1 \\
n+1 & 1 & & & 1 & & & & 1 & & \\
 & & \ddots & & & \ddots & & \ldots & & \ddots & \\
2n & & & 1 & & & 1 & & & & 1 \\
\hline
\end{array}
\quad
\underbrace{\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}}_{b}
$$

$$A$$

   which satisfies the hypothesis with $I_1 = \{1, \ldots, n\}$, $I_2 = \{n + 1, \ldots, 2n\}$. $\square$

4. It follows that we can solve the AP by solving through LP its continuous relaxation, obtained by replacing constraints $x_{ij} \in \{0, 1\}$ with $x_{ij} \geq 0$ (constraints $x_{ij} \leq 1$ are redundant).

Maximum flows and linear programming

- Incidence matrix A. Flow of value z from s to t:

$$\begin{aligned} \xi &\le q \\ A\xi &= \begin{cases} +z & \text{row } s \\ -z & \text{row } t \\ 0 & \text{rows } \neq s, t \end{cases} \\ \xi &\ge 0\,. \end{aligned}$$

- By introducing a vector d of n constants: $d_i = \begin{cases} -1 & \text{if } i = s \\ +1 & \text{if } i = t \\ 0 & \text{if } i \neq s, t \end{cases}$, $A\xi + dz = 0$.

- *The weaker form $A\xi + dz \le 0$ is equivalent to $A\xi + dz = 0$.*

  Proof For a vertex v_i, the quantity $[a_i'\xi]$ means [(flow-out v_i) $-$ (flow-in v_i)]. Hence:

  (a) row s: $A\xi + dz \le 0 \Rightarrow$ (flow-out s) $\le z$;

  (b) row t: $A\xi + dz \le 0 \Rightarrow$ (flow-in t) $\ge z$;

  (c) row $i \neq s, t$: $A\xi + dz \le 0 \Rightarrow$ (flow-out v_i) $\le$ (flow-in v_i).

  [(b) and (c)] $\Rightarrow$ (a) (flow-out s) $= z$;

  [(a) and (c)] $\Rightarrow$ (b) (flow-in t) $= z$;

  [(b) and (c)] $\Rightarrow$ (c) (flow-out v_i) $=$ (flow-in v_i) $\forall\ v_i \neq s, t$. $\square$

Maximum flows and linear programming (cont'd)

- Resulting LP model in $m + 1$ variables:

$$
\begin{array}{llcccl}
(D) & \max & & z & & \\
& & A\xi & + dz & \leq & 0 \\
& & \xi & & \leq & q \\
& & -\xi & & \leq & 0 \\
& & \xi & , \; z & \gtreqless & 0 \; .
\end{array}
$$

- (D) can be interpreted as the dual of a primal in standard form.
- It can be shown that the primal of (D) is equivalent to finding the minimum cut, and that
- the Ford-Fulkerson algorithm is equivalent to a specialized version of a particular LP algorithm, the *Primal-Dual Algorithm*.
- The **Primal-Dual Algorithm** provides the theoretical foundations of a number of important algorithms for graphs and networks.