Lecture 9: Maximum Flow in a Network

Prof. Krishna R. Pattipati
Dept. of Electrical and Computer Engineering
University of Connecticut
Contact: krishna@engr.uconn.edu; (860) 486-2890

Outline

- LP formulation and its dual
  - Maximum flow ≡ Minimum cut
  - A historical perspective on maximum flow algorithms
- Ford-Fulkerson labeling algorithm
- Dinic-Malhotra-Pramodh Kumar-Maheswari (DMKM) algorithm
  - Push-pull algorithm
  - Wave method
- Applications of maximum flow
  - Mapping problem
  - PERT networks

Preliminaries

- Suppose have a graph $G = < V, E >$ with two distinguished (designated) nodes s and t
  - s = source node; t = terminal node
- Consider edge between nodes i and j
  - Edge $< i, j >$ permits flow in both directions . . . undirected
    - Edge $< i, j >$ has a capacity c_{ij} in the forward direction and c_{ji} in the backward direction
    - $c_{ij} \geq 0$ and $c_{ji} \geq 0$
    - Usually, we assume $c_{ij} = c_{ji}$ (symmetric)
  - Edge $< i, j >$ permits flow from node i to node j *only*
    - Capacity $c_{ij} \geq 0$
    - $c_{ji} = 0 \rightarrow$ no flow allowed in reverse direction
- Since any undirected graph can be converted into a directed graph, we assume that G is directed

Preliminaries

- Let x_{ij} be the flow of commodity (oil, messages, vehicles) from i to j
  - By definition $x_{ji} = -x_{ij} \rightarrow$ flow matrix is *skew symmetric*
- $x_{ij} \leq c_{ij}$ and $x_{ji} \leq c_{ji} \rightarrow$ flows satisfy <u>capacity constraints</u>
- For any $< i, j >$ if $x_{ij} = c_{ij}$ or $x_{ji} = c_{ji} \Rightarrow$ edge $< i, j >$ is <u>saturated</u>
- If don't have an edge $< i, j > \Rightarrow c_{ij} = c_{ji} = x_{ij} = x_{ji} = 0$
- We can also look at flows in a network in terms of path flows
  - Indeed, we can establish an equivalence between arc flows and path flows
  - Let P be the set of paths in the network
  - Let y_p be the flow on path p

Let $\delta_{ij}(p) = \begin{cases} 1 & \text{if arc } <i, j> \text{ is on path } p \\ 0 & \text{otherwise} \end{cases}$ $\Rightarrow$ $x_{ij} = \sum_{p \in P} y_p \delta_{ij}(p)$

Conservation of Flow

- Flow conservation constraints
  - $\forall$ node $i \neq s, t,$ we have

$$flow\ in \equiv flow\ out$$

$$\sum_{j=1}^{n} x_{ji} \equiv \sum_{k=1}^{n} x_{ik} \forall i \neq s, t$$

$$\sum_{<j,i>\in E} x_{ji} \equiv \sum_{<i,k>\in E} x_{ik} \forall i \neq s, t$$

- Flow in the network

$$f = \sum_{i=1}^{n} x_{si} - \sum_{k=1}^{n} x_{ks} \quad \text{... net flow out of source}$$

$$\text{(or)} \quad f = \sum_{k=1}^{n} x_{kt} - \sum_{i=1}^{n} x_{ti} \quad \text{... net flow into sink}$$

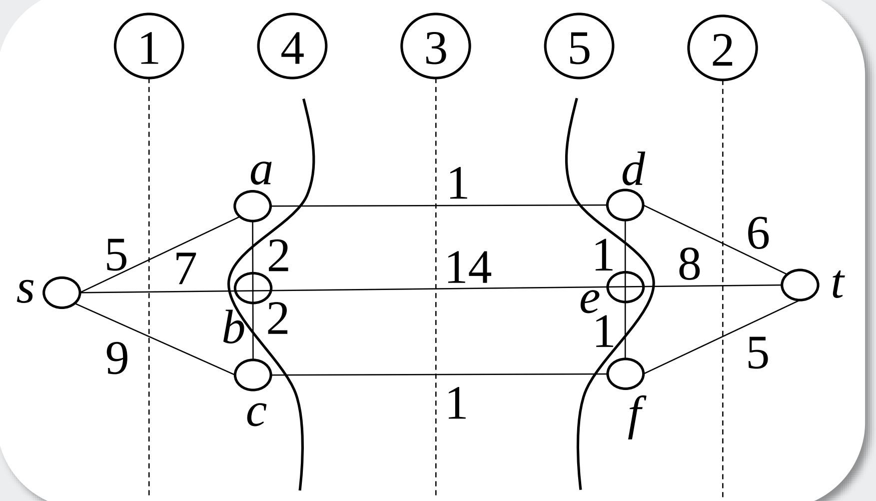
- Max. flow problem:
  - Want to find the maximum flow that the network can sustain from s to t
    - What is the capacity of the network?

Max. flow problem

- LP formulation

$$
\begin{aligned}
&\max \; f \\
&\text{s.t.} \sum_{i=1}^{n} x_{si} - \sum_{k=1}^{n} x_{ks} - f = 0 && \text{(source flow)} \\
&\sum_{j=1}^{n} x_{ij} - \sum_{k=1}^{n} x_{ki} = 0, \;\; \forall i \neq s,t && \text{(Kirchoff's law)} \\
&-\sum_{k=1}^{n} x_{kt} + \sum_{i=1}^{n} x_{ti} + f = 0 && \text{(sink flow)} \\
&0 \leq x_{ij} \leq c_{ij} && \text{(capacity constraints)}
\end{aligned}
$$

- Example:



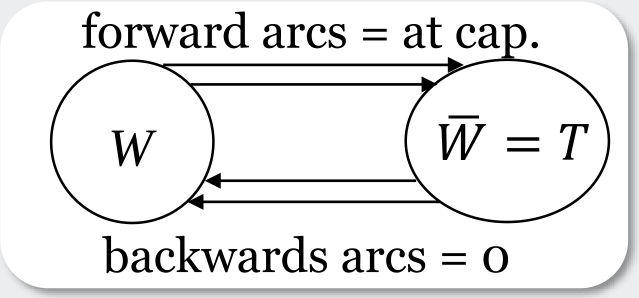
Capacity of a cut

- Capacities provide a bound on the flow
- At the source: can't send more than (5 + 7 + 9) = 21 units
- Can't send this because at the sink: can't receive more than (6 + 8 + 5) = 19 units
- Can't send 19 units either because at the center: can't move more than (14 + 1 + 1) = 16 units
- What we have defined are three *cuts*
  - Cut ≡ A partition (or separation) of nodes into two groups W and T such that $s \in W$ and $t \in T = \overline{W}$
  - *Capacity of the cut* is the sum of capacity of edges crossing from W to T

$$C\left(W, \overline{W}\right) = \sum_{\substack{<i,j> \in E: \\ i \in W,\, j \in \overline{W}}} c_{ij} \begin{cases} \text{cut at the source: 21} \\ \text{cut at the sink: 19} \\ \text{cut in the middle: 16} \end{cases}$$

Max Flow ≡ Min cut

- Know $f \leq C(W, \bar{W}), \forall (W, \bar{W})$ cut
  - Can't push through 16 units either!!
- Cut (4) → $7 + 2 + 1 + 2 + 1 = 13$
  - Can't push through 13 units either!!
- Cut (5) → $1 + 1 + 8 + 1 + 1 = 12$
  - Cut(5) →$W = \{s, a, b, c, e\}$; $\bar{W} = \{t, d, f\}$



- Property of a cut
  - Each cut corresponds to a feasible solution of the dual of max. flow problem …later
  - Properties of cut(5):
    - Every forward edge across the cut is saturated
    - It is a cut of maximum capacity

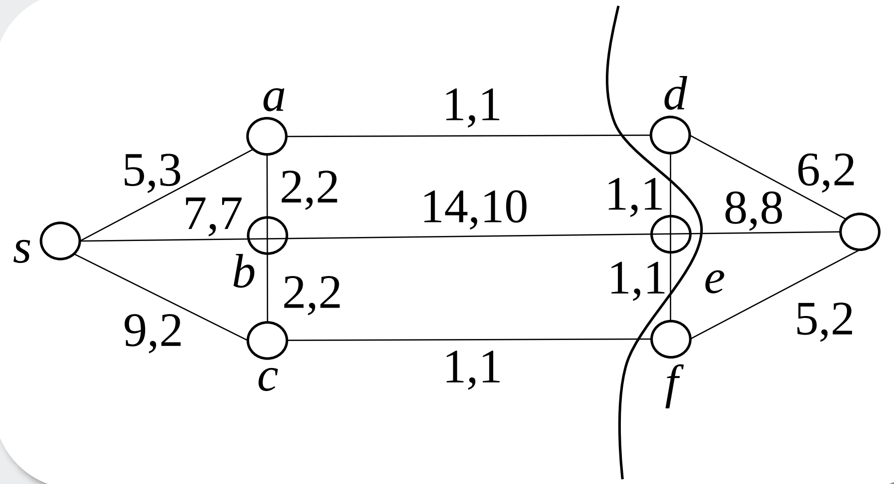
      → Max. flow = min cut (?)

      …Recall dual is a minimization problem!!

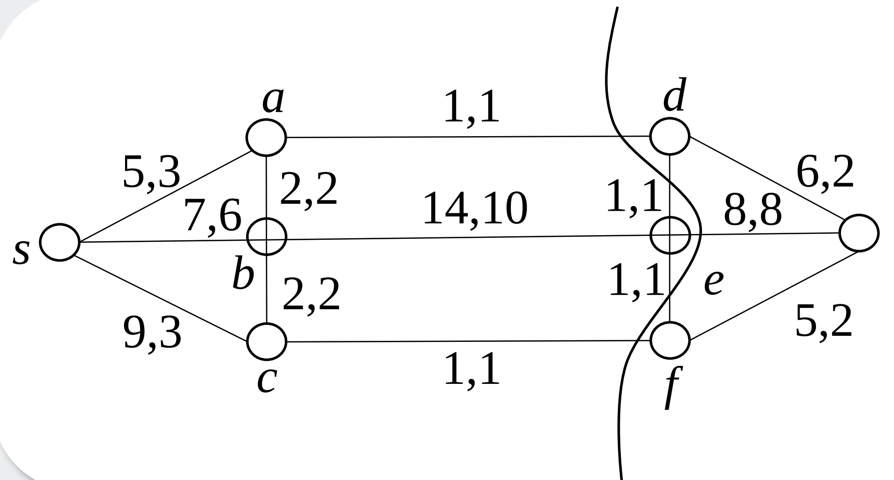
Some observations from example

- Minimum cut is not unique
  - Min. cut is not unique: e.g., if 14→10

    ⇒ a second min. cut



- Maximum flow pattern is not unique
  - Max. flow pattern is not unique. Degenerate bfs
  - Max. flow value f = 12 is <u>*unique*</u>: cap. of min cut is unique



Establishing feasibility

- Let us look at the dual to establish feasibility

Primal

$$\begin{aligned}
&\min\ -f\\
&\text{s.t.}\ \sum_{i=1}^{n} x_{si} - \sum_{k=1}^{n} x_{ks} - f = 0\\
&\sum_{j=1}^{n} x_{ij} - \sum_{k=1}^{n} x_{ki} = 0,\ \ \forall i \neq s,t\\
&\sum_{i=1}^{n} x_{ti} + \sum_{k=1}^{n} x_{kt} + f = 0\\
&-x_{ij} \geq -c_{ij};\ x_{ij} \geq 0
\end{aligned}$$

Dual

$$\begin{aligned}
&\max\ -\sum_{<i,j>\in E} \mu_{ij} c_{ij} = \min \sum_{<i,j>\in E} \mu_{ij} c_{ij}\\
&\text{s.t.}\ \ -\gamma_s + \gamma_t \leq -1\\
&\gamma_i - \gamma_j - \mu_{ij} \leq 0\\
&\gamma_i\ \text{unconstrained}\\
&\mu_{ij} \geq 0
\end{aligned}$$

- Let $\gamma_i = -\lambda_i, \forall i$
- Final Dual form

$$\begin{aligned}
&\Rightarrow \min \sum_{<i,j>\in E} \mu_{ij} c_{ij}\\
&\text{s.t.}\ \ \lambda_t - \lambda_s \geq 1\\
&\lambda_i - \lambda_j + \mu_{ij} \geq 0 \Rightarrow \lambda_j - \lambda_i \leq \mu_{ij}\\
&\mu_{ij} \geq 0
\end{aligned}$$

Establishing dual feasibility of a cut

- Every $s-t$ cut $(W,\bar{W})$ determines a dual feasible solution with cost $C(W,\bar{W})$ as follows:

$$\mu_{ij}=\begin{cases}1 & i\in W;\ j\in\bar{W}\\ 0 & \text{otherwise}\end{cases}$$

$$\Rightarrow \sum_{<i,j>\in E}\mu_{ij}c_{ij}=\sum_{\substack{<i,j>\in E\\ i\in W,\ j\in\bar{W}}}c_{ij}=C\left(W,\bar{W}\right)$$

$$\lambda_i=\begin{cases}0 & i\in W\\ 1 & i\in\bar{W}\end{cases}\quad \text{dual feasible}$$

$$\left.\begin{array}{l} i\in W,\ j\in W:\text{OK}\\ i\in\bar{W},\ j\in\bar{W}:\text{OK}\\ i\in W,\ j\in\bar{W}:\text{OK}\\ i\in\bar{W},\ j\in W:\text{OK}\end{array}\right\}\Rightarrow \text{feasible}$$

- $\Rightarrow$ note that $\lambda_t=1$ and $\lambda_s=0$ <u>always</u>

Max. flow ≡ Min. cut

- Flow x_{ij}^* and $(W, \bar{W})$ are jointly optimal iff
  - $x_{ij}^* = 0, \;\; \forall < i, j > \in E \ni i \in \bar{W}$ and $j \in W$

    $\Rightarrow$ Zero flows on backward arcs
  - $x_{ij} = c_{ij}, \;\; \forall < i, j > \;\; \in \; E \ni i \in W$ and $j \in \bar{W}$

    $\Rightarrow$ Saturated flows on forward arcs
- If $i \in \bar{W}$ and $j \in W$

  $\Rightarrow \lambda_i - \lambda_j + \mu_{ij} = 1 - 0 + 0 = 1 > 0 \Rightarrow x_{ij}^* = 0$
- If $i \in \; W$ and $j \in \bar{W}$

  $\Rightarrow \lambda_i - \lambda_j + \mu_{ij} = 0 - 1 + 1 = 0 \;\; \Rightarrow x_{ij}^* = c_{ij}$
- To see this duality more clearly, consider a graph with $c_{ij} = c_{ji} = 1$
- Minimal cut ≡ smallest number of edges across it ≡ # of edges from W to $\bar{W}$
- Maximal flow ≡ # of disjoint paths from s to t

  $\Rightarrow$ $\boxed{\text{Max. \# of disjoint paths from } s \text{ to } t \equiv \text{min. \# of edges across a cut}}$ (or)

  $\Rightarrow$ $\boxed{\text{Capacity of a network} \equiv \text{sum of capacities of its weakest links}}$

Historical perspective on max. flow algorithms

Year	Algorithm	Complexity
1956	Ford & Fulkerson	can be exponential
1969	Edmonds & Karp	$O(nm^2)$
1970	Dinic	$O(n^2m)$
1974	Karzanov	$O(n^3)$
1978	Malhotra, Kumar, Maheswari	$O(n^3)$
1977	Cherkaski	$O(n^2m^{1/2})$
1978	Galil	$O(n^{5/3}m^{1/2})$
1979	Galil, Naamad, Shiloach	$O(nm(\log n)^2)$
1980	Sleator & Tarjan	$O(nm \log n)$
1986,87	Goldberg & Tarjan	$O(n^3)$
1987	Bertsekas	$O(n^3)$
1989	Ahuja & Orlin	survey of max. flow algorithms

Historical perspective on max. flow algorithms

- Ford-Fulkerson & Edmonds & Karp
  - Try to push flow on one path at a time called an *augmentation path*
  - If can't find a path from s to t, we are done!!

- Other algorithms
  - Several paths at once
  - We construct a series of *layered Networks*
  - If can't construct a layered network from $s - t$, we are done!

- More recent algorithms
  - Work on arcs $\Rightarrow$ distributed computation

Idea of Ford-Fulkerson labeling algorithm

- Ford-Fulkerson labeling algorithm
  - Given: a directed graph $G = <V,E>$ and a feasible flow (x_{ij})
  - An *augmentation path* (or augmenting path) p is a path from s to t in the undirected graph resulting from G by ignoring edge directions with the following properties:
    - $\forall <i,j> \in E$ that is traversed by P in the forward direction (called forward arc $<i, j>$ or forward labeling of j), we have

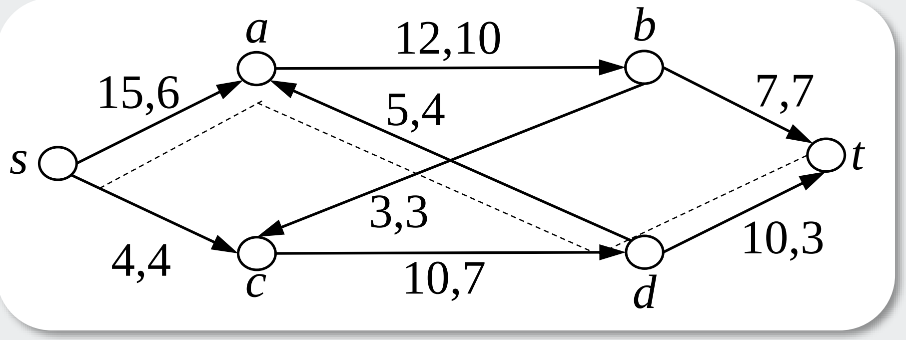
$$x_{ij} < c_{ij} \rightarrow x_{ij} \uparrow \begin{cases} \text{we can forward label } j \text{ if} \\ \bullet \ \ i \text{ is labeled and } j \text{ is not} \\ \bullet \ \ x_{ij} < c_{ij} \end{cases}$$

    - $\forall (j,i) \in E$ that is traversed by P in the backward direction (called backward labeling of j), we have

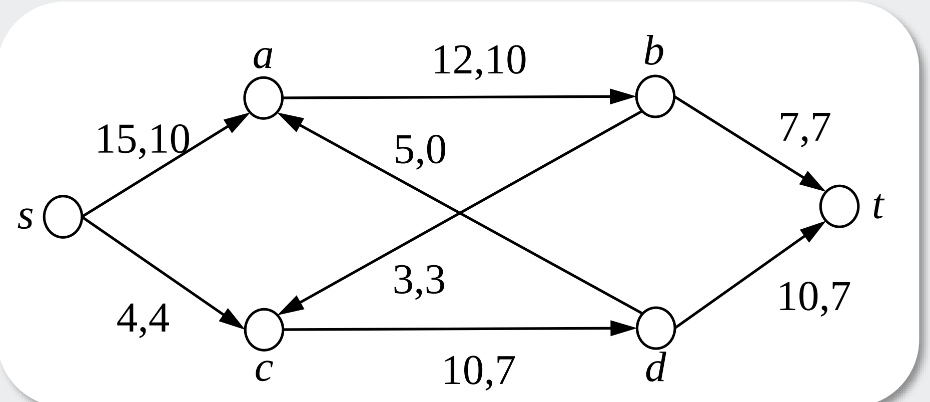
$$x_{ji} > 0 \rightarrow x_{ji} \downarrow \begin{cases} \text{we can backward label } j \text{ if} \\ \bullet \ \ i \text{ is labeled and } j \text{ is not} \\ \bullet \ \ x_{ji} > 0 \end{cases}$$

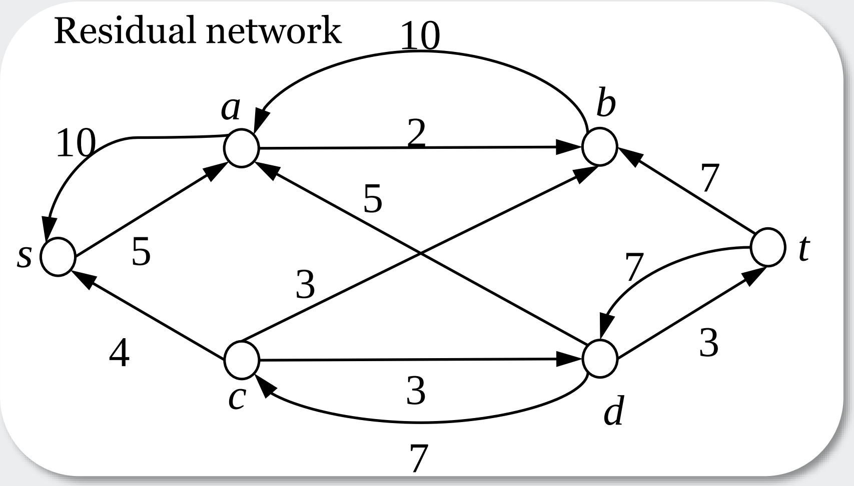
This idea is similar to Hungarian algorithm for the assignment problem

Example



- We can increase the flow on the augmenting path p until we violate the capacity constraint of a forward arc or empty a backward arc



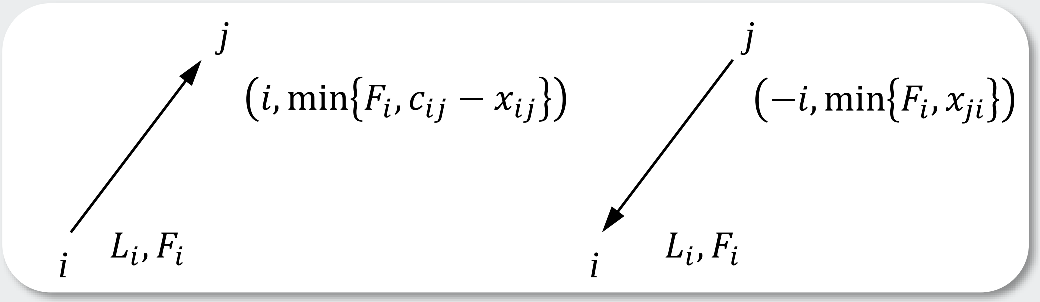


$$\delta = \min_{(i,j)\in P} \begin{cases} c_{ij} - x_{ij} & <i,j> \text{ forward} \\ x_{ji} & <j,i> \text{ backward} \end{cases}$$

$$\delta = \begin{cases} 9 \quad 7 \\ 4 \end{cases} \rightarrow \delta = 4$$

How to find augmentation paths?

- We propagate *labels* from s to t or get stuck
- Each node i has a two part label: label(i) = $< L_i , F_i >$
  - L_i = from where i was labeled $\begin{cases} \bullet \text{ Parent of } i \text{ for forward arc} \\ \bullet \text{ Son of } i \text{ for backward arc} \end{cases}$
  - F_i = amount of extra flow that can be brought to i from s



When label all nodes adjacent to i, we are said to scan i

- We add all nodes labeled by scanning i to a LIST
  - So, to find augmenting path, scan $s \xrightarrow{i=s}$ add to LIST all nodes labeled from $i \longrightarrow$ pick a node from LIST
- Outcome
  - t gets labeled $\Rightarrow$ found an augmentation path
  - LIST becomes empty $\Rightarrow$ can't find a path $\Rightarrow$ optimal

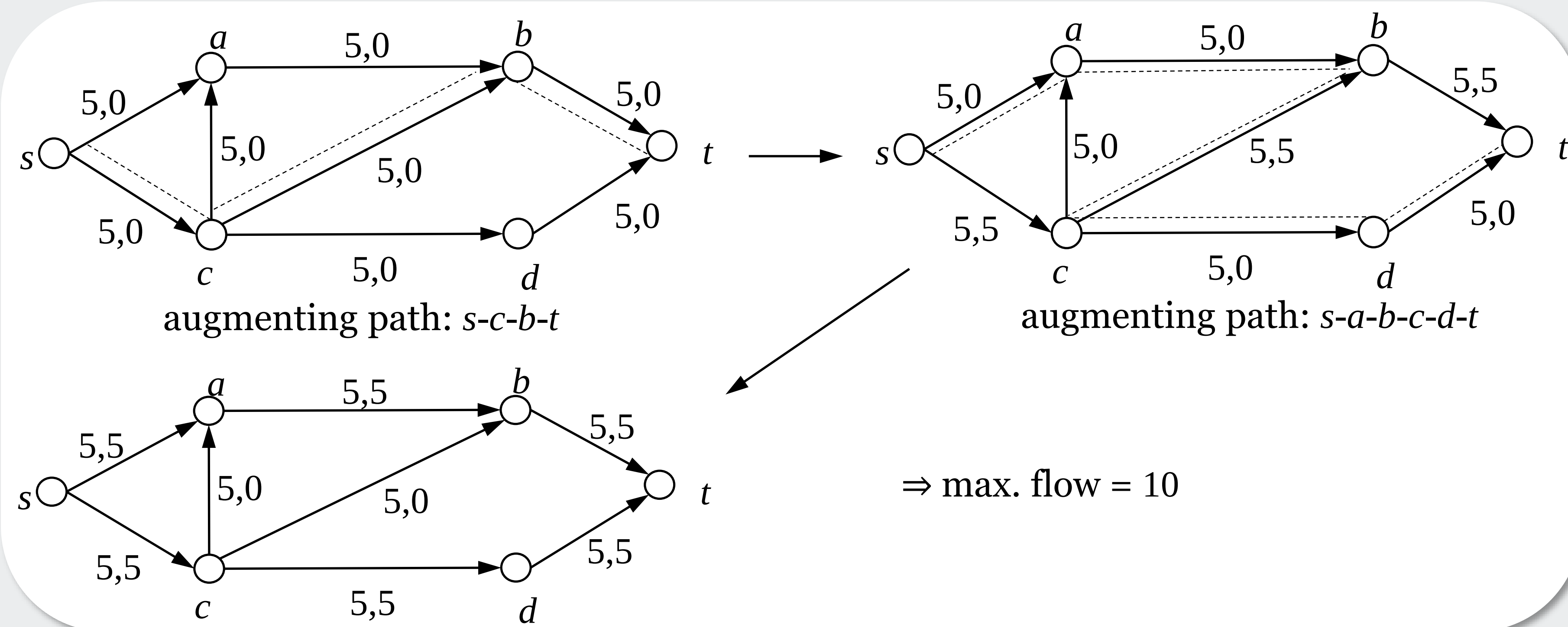
Algorithm Procedure

$\forall i,j \in E$, let $x_{ij} = 0$
repeat
    set all labels to 0; LIST = $\{s\}$
    while LIST $\neq \emptyset$ do
        pick any node $i \in$ LIST and remove it
        scan $i \Rightarrow$ add to list all nodes on augmenting path
        if t is labeled
            augment flow x_{ij}
            goto repeat
        end if
    end do

- What does scan i mean?
- Procedure scan i
    - Label forward to all unlabeled nodes adjacent to i by arcs that are unsaturated, putting newly labeled nodes on LIST
    - Label backward to all unlabeled nodes from which i is adjacent by arcs that have positive flows, putting newly labeled nodes on LIST

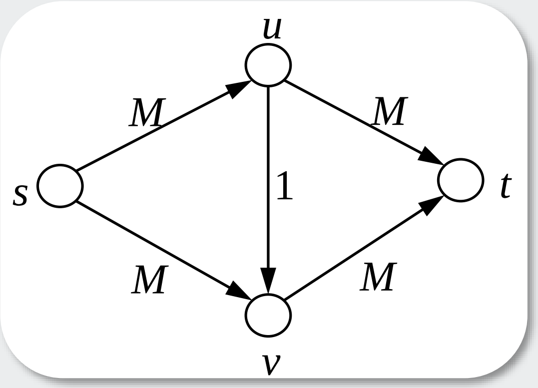
Example

- Example



Cost analysis

- When c_{ij} are integers ⇒ Ford-Fulkerson takes at most f augmentations



$\langle s\ u\ v\ t\rangle \to \langle s\ v\ u\ t\rangle \to \langle s\ u\ v\ t\rangle \to \cdots \to 2M$ iterations

- When c_{ij} are rational
  - Write as ratio of integers with a common denominator D
  - Scale each cost by D ⇒ takes at most Df iterations
- When c_{ij} are irrational (of infinite precision), Ford-Fulkerson may not terminate
  - In fact, may converge to a non-optimal value
  - **If use shortest augmenting path, all these problems go away** . . . In fact, Edmonds & Karp showed that the # of augmenting paths $\leq \frac{n(n^2-1)}{4}$ with this strategy (∃ even better algorithms)

Pathological Example (Ford and Fulkerson, 1962)

$< x_i, y_i >=$ arcs A_i

$A_1 = a_0 = 1$

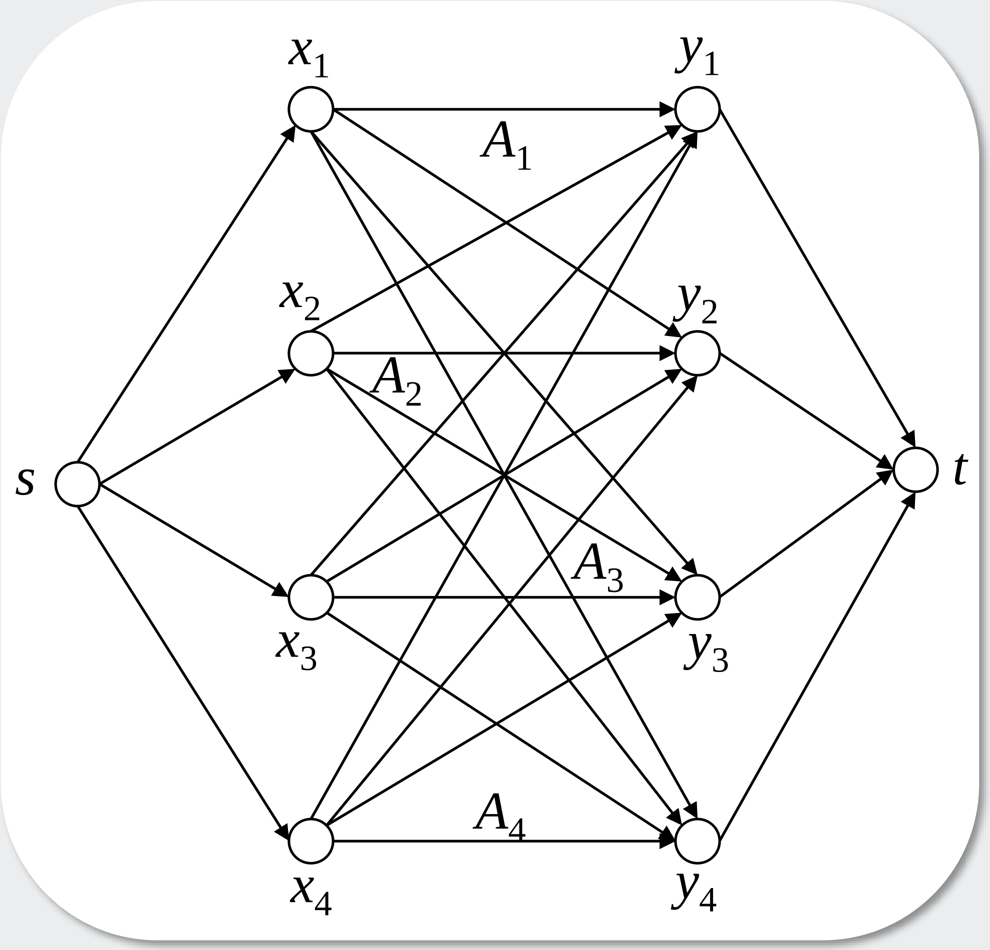
$A_2 = a_1 = \frac{\sqrt{5}-1}{2} = 0.618\ldots = \sigma$

$A_3 = a_2 = a_0 - a_1 = \sigma^2$

$A_4 = a_2 = a_0 - a_1 = \sigma^2$

All other arcs have capacity $s = \frac{1}{1-\sigma}$

In general, for this network, at the n^{th} Step, flow augmentation will be a_{n+1} and a_{n+2} such that $a_{n+2} = a_n - a_{n+1}$



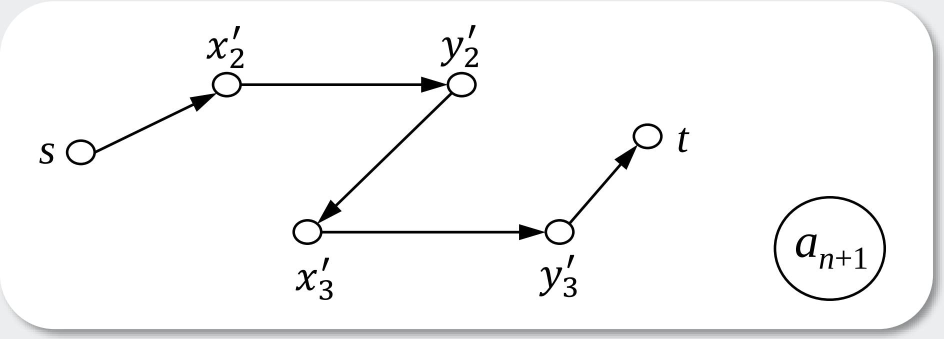
Pathological Example (Ford and Fulkerson, 1962)

- At step n … add a_{n+1} & a_{n+2}

$$\Rightarrow a_0 + (a_1 + a_2) + \quad \cdots \quad + (a_{n+1} + a_{n+2}) = \frac{1}{1-\sigma} = s$$

- Start with $\langle s \;\; x_1 \;\; y_1 \;\; t \rangle \Rightarrow \langle A_1 \;\; A_2 \;\; A_3 \;\; A_4 \rangle = \langle 0 \;\; a_1 \;\; a_2 \;\; a_2 \rangle \Rightarrow$ flow a_0
- At step $n (n \geq 1)$:
  - Suppose at step n, we order arcs A_1', A_2', A_3', A_4' ∍ residual capacities are: 0, a_n , a_{n+1} , a_{n+1} , respectively
  - Order $<x_i', y_i'>$ accordingly
  - Flow so far: $a_0 + a_1 + \ldots + a_{n-1}$
- Step: n (a):
  - Choose flow augmenting path

    $\Rightarrow$ Residual cap: 0, a_{n+2} , 0, a_{n+1} , respectively



Pathological Example (Ford and Fulkerson, 1962)

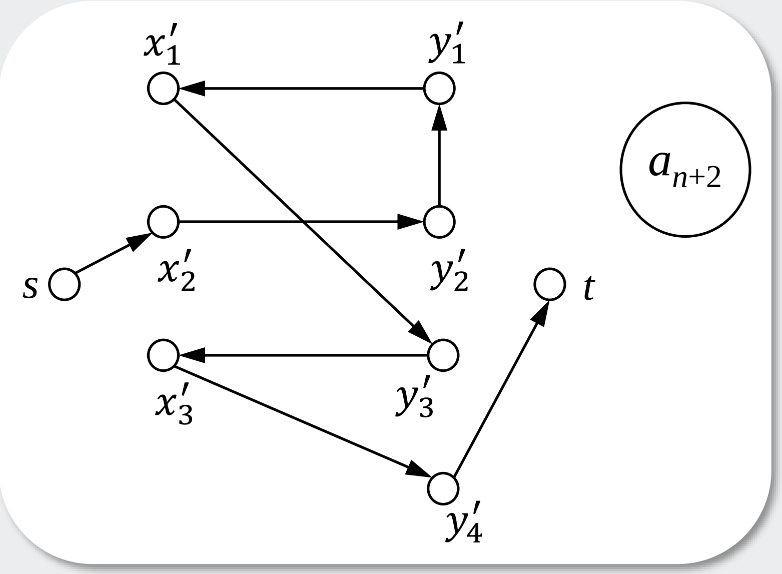
- Step: $n - b$:
  - Choose flow augmenting path

    $\Rightarrow a_{n+2}$, 0, a_{n+2} , a_{n+1}

$\Rightarrow$ Flow so far: $a_0 + a_1 + \cdots + a_n$

$\Rightarrow$ Step n ends with appropriate residual capacities for step (n+1)

As $n \rightarrow \infty$, flow converges to $s = \frac{1}{1-a_1} = \frac{1}{1-\sigma} = s$



- However, max. flow = 4s
- Ford-Fulkerson terminates with non-optimal flows !!

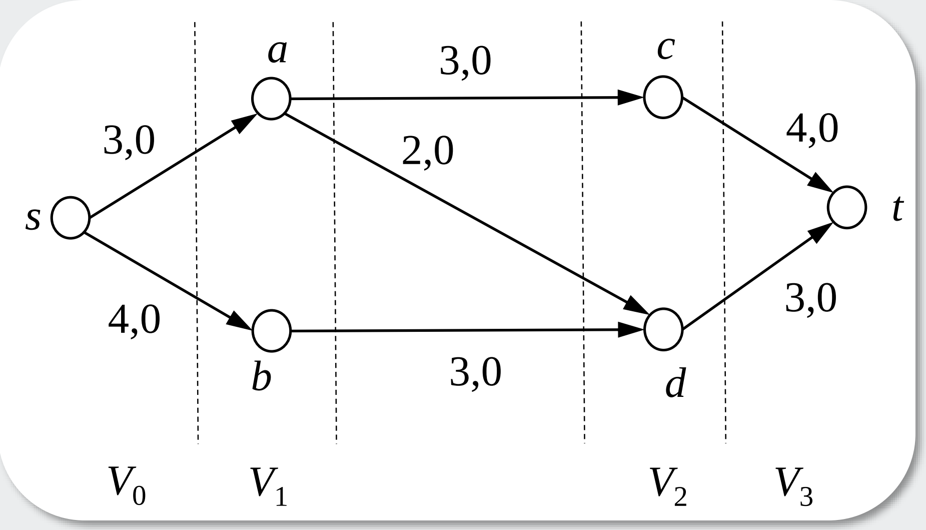
DMKM Algorithm

- Two phase algorithm executed iteratively
- Phase 1
  - Obtain an auxiliary layered network (i.e., an acyclic graph) from the original network G with a feasible flow pattern
- Phase 2
  - Find *saturating flow* in a layered network . . . also called *blocking flows*
  - Phase 2 takes $O(n^2)$ or $O(m \log n)$ steps depending on implementation
- We will show that phase 1 need be executed at most n times

  $\Rightarrow O(n^3)$ or $O(mn \log n)$ steps for the algorithm

DMKM Algorithm (Phase 2)

- Consider phase 2 first
  - Want to find saturation flows in a layered network
  - What is a layered network?
    - An acyclic graph $G_L = < V_L, E_L > \ni V_L$ is partitioned into layers $V_0, V_1, \cdots, V_L$
    - $V_0 = \{s\}$, V_1 = set of nodes adjacent to s
    - V_k = set of nodes adjacent to all nodes of $V_{k-1}, k \geq 1$
    - Finally, $V_L = \{t\}$



How to find saturating flows?

DMKM Algorithm (Phase 2)

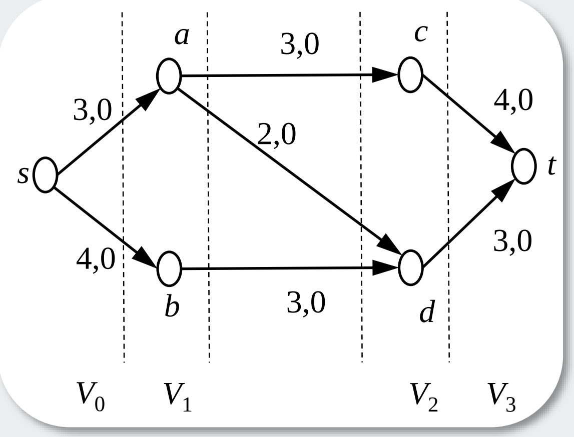
- Repeat until *s* and *t* are disconnected
  - Saturate some of the edges
  - Remove edges (& nodes if either all incoming or outgoing edges are saturated)
- The process is called “finding saturating flows” or “finding blocking flows”
- Two algorithms for finding blocking flows
  - “Push-pull" algorithm
  - Wave method

DMKM Algorithm (Phase 2)

- "Push-pull method"
  - Define throughput of a node i, $i \neq s, t$ as: $$TP_i = \min\left\{ \sum_{(k,i)\in E} (c_{ki} - x_{ki}), \sum_{(i,j)\in E} (c_{ij} - x_{ij}) \right\}$$

    = min{potential input to i, potential output from i}
  - Similarly $$TP_s = \sum_{(s,i)\in E} (c_{si} - x_{si}); TP_t = \sum_{(k,t)\in E} (c_{kt} - x_{kt})$$
  - Suppose $$TP_r = \min_i TP_i \;\&\; r = \arg\min_i TP_i$$
  - r is called the reference node
- For the example problem

$$TP_s = 7,\ TP_a = 3,\ TP_b = 3,\ TP_c = 3,\ TP_d = 3,\ TP_t = 7$$
$r = a$ or b or c or d

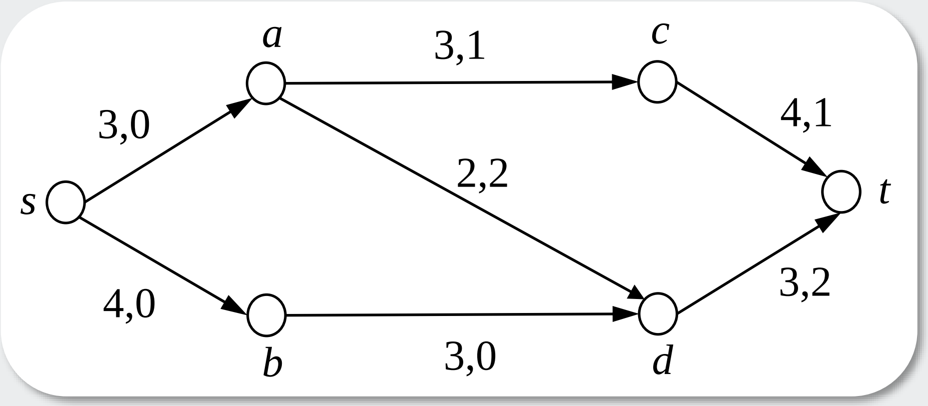


- **Key**: guaranteed at least TP_r units of flow from s to t
- **Q**: How to "pull" TP_r units of flow from s to t & how to "push" TP_r units from r to t?

DMKM Algorithm (Phase 2)

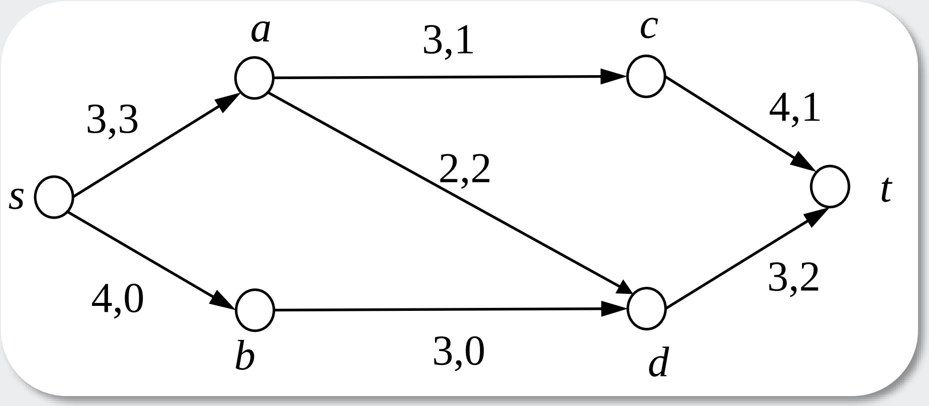
- "Push" TP_r units from r to t
  - Distribute TP_r units to the outgoing edges from r
    - Take these edges one by one & saturate them until all TP_r units are exhausted
    - Flow reaching the next layer is distributed among its outgoing edges & pushed to the next layer

- Example:
  - Pick $r = a$



DMKM Algorithm (Phase 2)

- "Pull" TP_r units from s to r
  - Pull TP_r from immediate predecessors of r
  - Then from their immediate predecessors & so on
- Example:

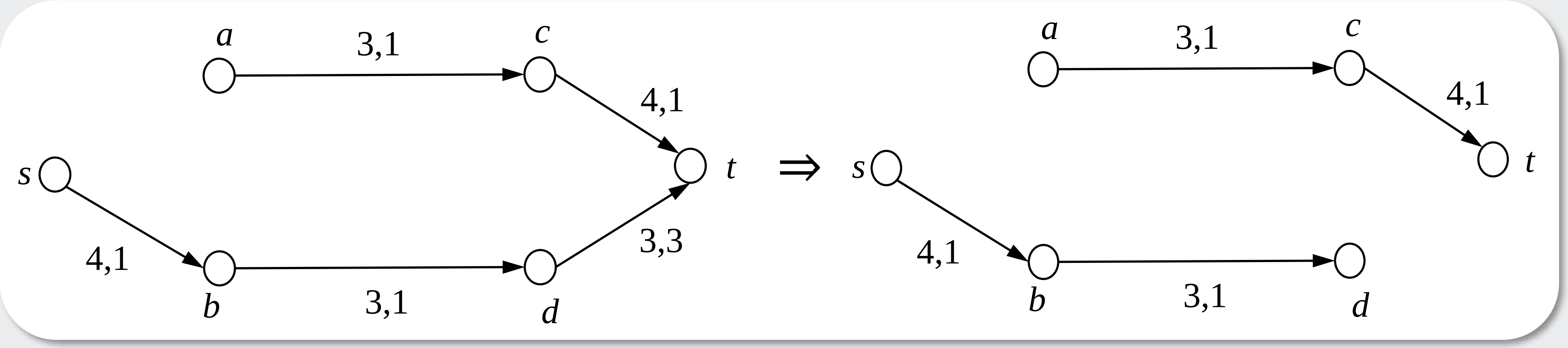


- Delete all saturated edges & nodes that have all their incoming or outgoing edges saturated
  - Deletion of a node ⇒ deletion of all its incoming or outgoing edges

DMKM Algorithm (Phase 2)

- Result

$$TP_s = 4$$
$$TP_b = 3$$
$$TP_d = 1$$
$$\downarrow$$

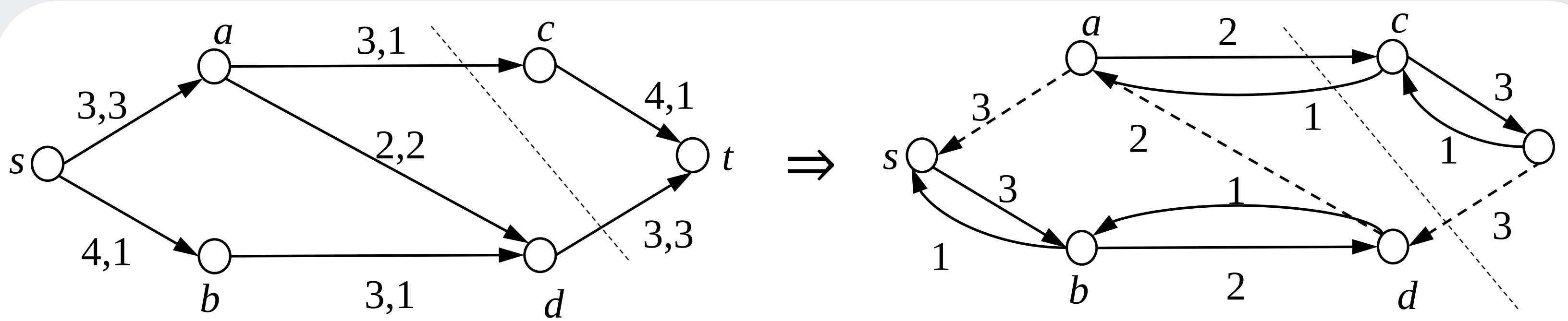


⇒ Saturating flow = 4, since *s* and *t* are disconnected

- Note: saturating flow ≠ maximum flow

DMKM Algorithm (Phase 1)

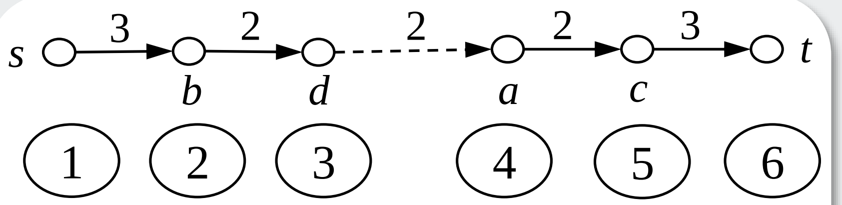
- Phase 1 … construct a layered network from a graph with a feasible flow pattern
  - We do it in two steps
    - Construct a network G_x with a feasible flow pattern $\langle x_{ij} \rangle$ from G
    - Then, construct a layered network from G_x
  - How to construct G_x?
    - If $<i, j> \in E$ and $x_{ij} < c_{ij}$, then $<i, j> \in G_x$ and $d_{ij} = c_{ij} - x_{ij}$, where d_{ij} = capacity of edge $<i, j> \in G_x \Rightarrow x_{ij} \uparrow$
    - If $<i, j> \in E$ and $x_{ij} > 0$, then $<j, i> \in G_x$ and $d_{ji} = x_{ji} \Rightarrow x_{ji} \downarrow$
  - Network G_x is called the "residual graph" (residual network)
- Layered network example



If f^* is max. flow on G $\Rightarrow$ $f^* - f$ is max. flow on G_x

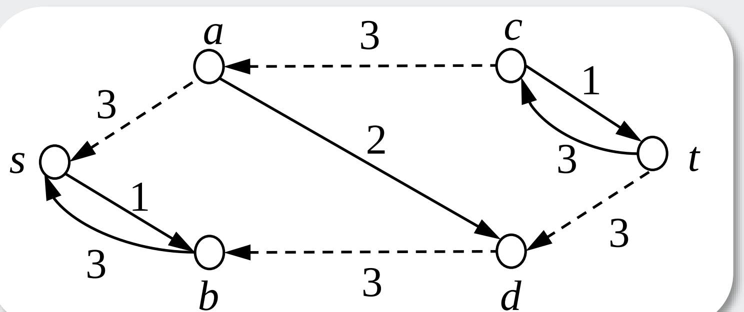
DMKM Algorithm (Phase 1)

- Construction of a layered network from G_x
  - Use breadth-first search



⇒ saturating flow = 2
total saturating flow so far = 4 + 2 = 6

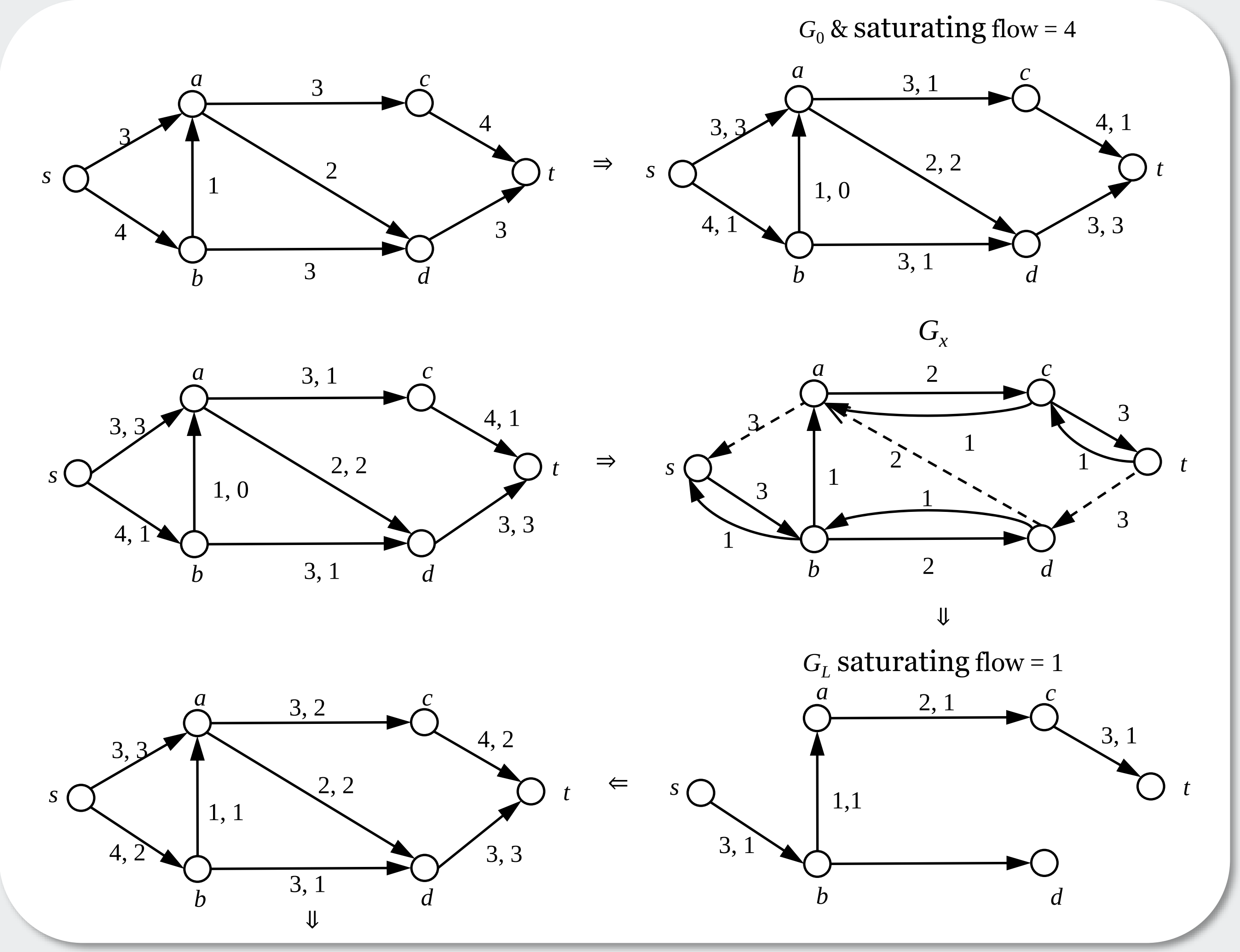
- Rules
  - If any node is in a higher layer than t, then discard the node & all edges incident on it
  - Discard all nodes other than t that are in the same layer as t
  - Discard all edges that go from a higher layer to a lower layer
  - Discard any edge that joins two nodes of the same layer
- Example: next G_x for our layered network example



s & t disconnected ⇒ max. flow = 6

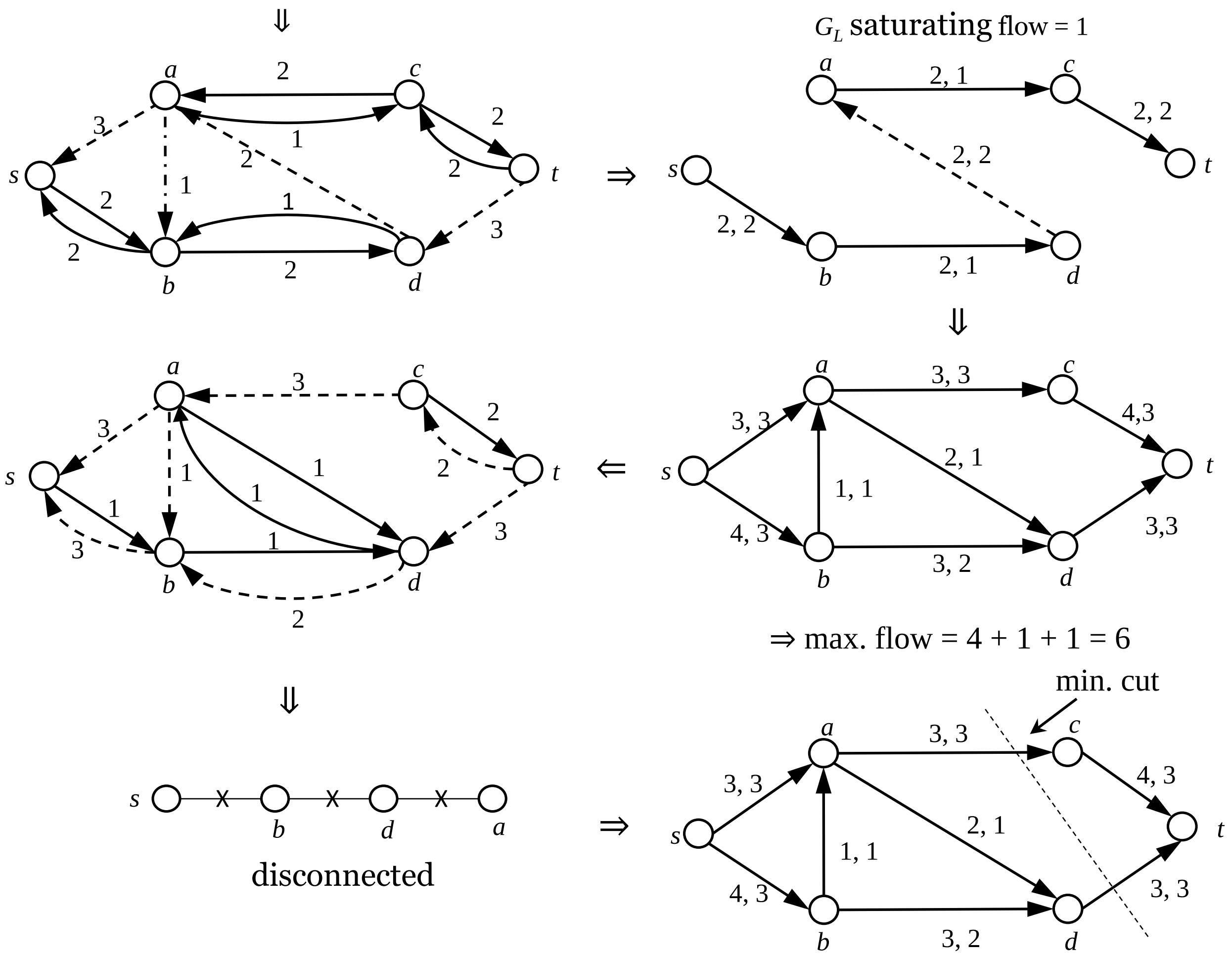
DMKM Algorithm (Phase 1)

- Example 2:



DMKM Algorithm (Phase 1)

- Example 2 continued:



DMKM algorithm

- Initialize flows $x_{ij} = 0$, *done* = "*false*", $f = 0$
- While not (*done*) do
  - Construct $G_x = <V_x, E_x>$ with capacity matrix D
  - If t is not reachable from $s \in G_x$
    - *done* = "*true*"
  - Else
    - Construct a layered network G_L from G_x
    - Find saturating flow g of G_L
    - $f = f + g$
  - End if
- End do

Time complexity

- Finding saturating flows in a layered network (phase 2)
  - At least one node is deleted at each iteration

    $\Rightarrow$ At most n iterations
  - In the i^{th} iteration
    - Work involved is related to the # of times different edges are processed

      $$T = T_s + T_p$$

      where T_s ...saturated to capacity and T_p ...partial
    - If an arc is saturated, delete it

      $$\Rightarrow T_s = O(m)$$
    - # of partial steps $\leq n$ (1 for each node)

      $$\Rightarrow T_p = O(n^2)$$

    $\Rightarrow$ Total work = $O(m) + O(n^2) = O(n^2)$
- Phase 1
  - There are at most $(n - 1)$ steps since the layers increase by at least one & $s - t$ path length $\leq n - 1$
  - Constructing layered network ... $O(m)$

    $\Rightarrow$ Total work: $O(nm) + O(n^3) = O(n^3)$

Blocking flow computation via "wave method"

- To present the method, we need the concept of *preflow*
  - A preflow (x_{ij}) satisfies skew symmetry ($x_{ij} = -x_{ji}$) and capacity constraints
  - The conservation constraints are not satisfied
    - Flow (x_{ij}) is such that inflow ≥ outflow for every node ≠ s

      ⇒ Total inflow into any node $i \neq s$ must be at least as great as the total outflow from i

$$\Delta_i = \sum_j x_{ji} - \sum_k x_{ik} \geq 0$$

    - Since $x_{ik} = -x_{ki}$, we can also write this as:

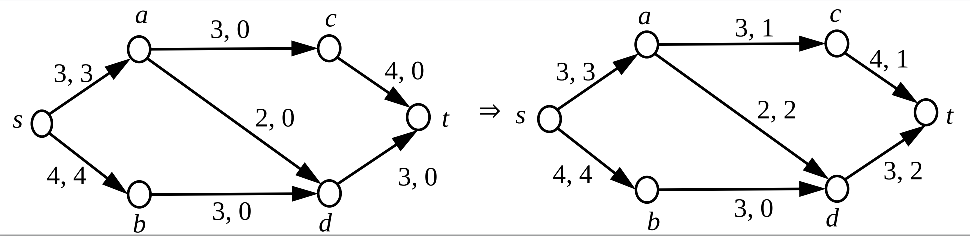
$$\Delta_i = \sum_j x_{ji} \geq 0$$

      where j is over all edges incident to i (both incoming and outgoing edges)
  - Balanced node $\Delta_i = 0, (i \neq s, t)$
  - Unbalanced node $\Delta_i \geq 0, (i \neq s, t)$
  - A preflow is blocking if it saturates every path
  - An edge on each path is at its capacity
- **Key idea of wave method**
  - Start with a blocking preflow
  - Iteratively convert it into a balanced blocking flow

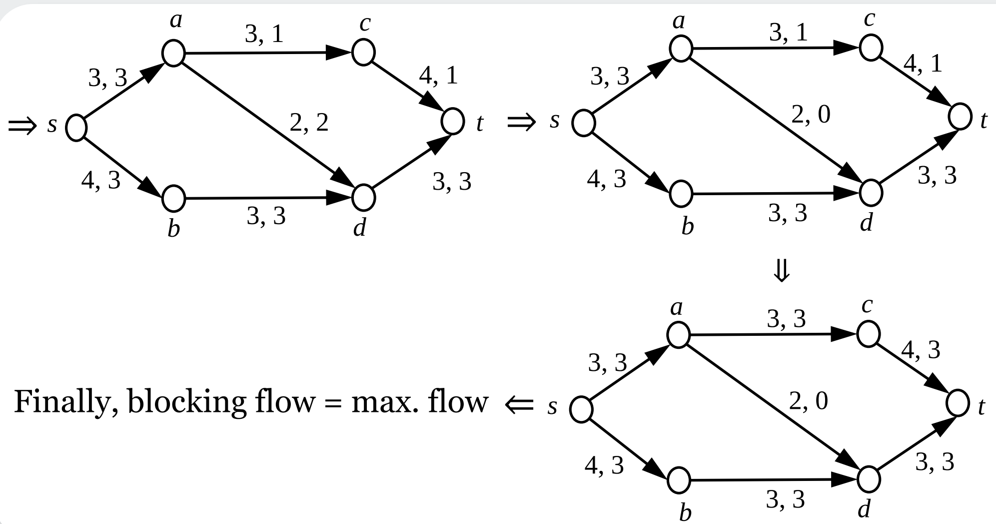
    ⇒ A flow that satisfies conservation constraints
- How?
  - **Increase the outgoing flow of an unblocked & unbalanced node** (or)
  - **Decrease the incoming flow of a blocked node**

Illustrative Example

- Start with a preflow that saturates every edge out of *s* & zero flow on all other edges

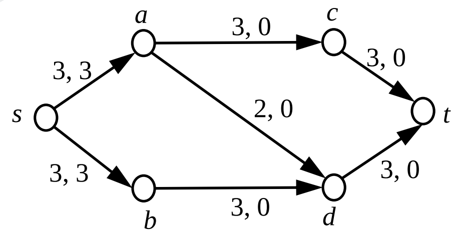


- Blocked node ⇒ decrease incoming flow; unblocked node ⇒ increase outgoing flow
- Increase step:
  - If (i, j) is an unsaturated edge such that j is unblocked, increase x_{ij} via: $x_{ij} \leftarrow x_{ij} + \min\{c_{ij} - x_{ij}, \Delta_i\}$
- Decrease step:
  - If node i is blocked and ∃ a positive flow x_{ji}, then: $x_{ji} \leftarrow x_{ji} - \min\{x_{ji}, \Delta_i\}$

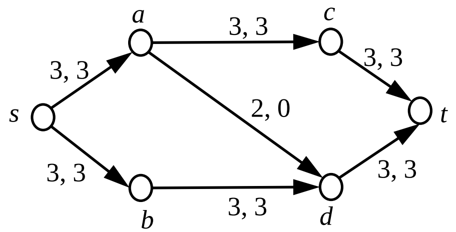


Mechanization of the wave method

- Start with a preflow ∋ every edge out of *s* is saturated & has zero flow on all other edges
- Repeat increase flow & decrease flow until all nodes are balanced
- Increase flow
  - Scan nodes other than *s* and *t* in topological order (reverse post-order visit)
  - Balance each node *i* that is unbalanced & unblocked when it is scanned
  - If balancing fails, label node *i* blocked (permanently)
- Decrease flow
  - Scan vertices other than *s* and *t* in reverse topological order (i.e., post-order visit)
  - Balance each vertex that is unbalanced & blocked when it is scanned
- Example:



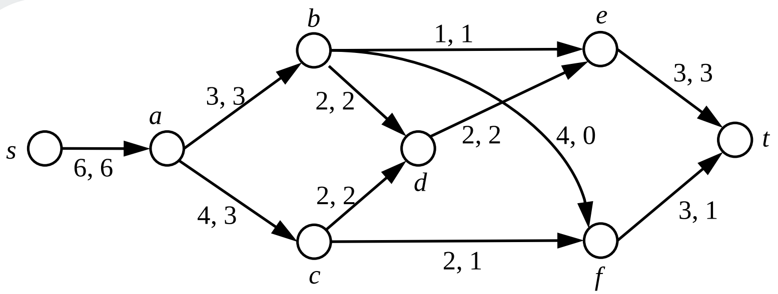
dfs scanning: *s b d t a c*
Post order: *t d b c a s* (reverse topological order)
Topological order: *s a c b d t*



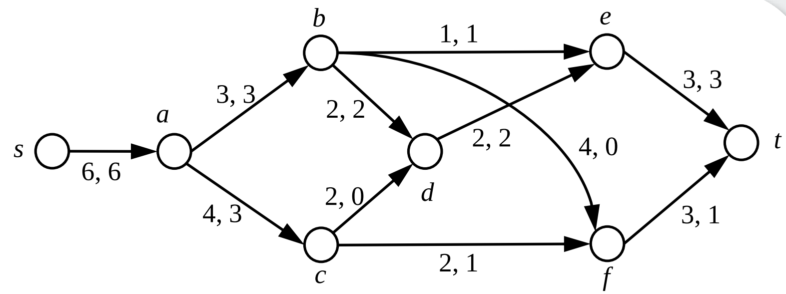
Easy problem!

Mechanization of the wave method

Example:

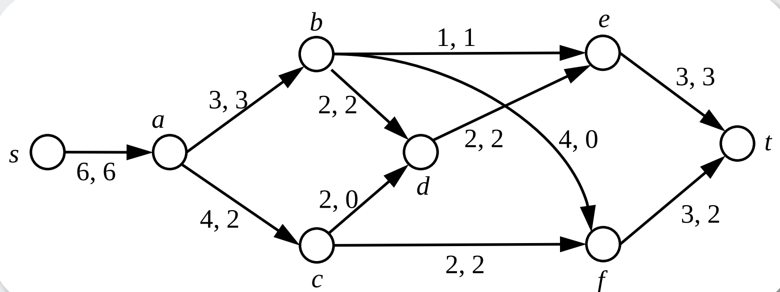


dfs scanning: *s a c f t d e b*
Post order: *t f e d c b a s*
Topological order: *s a b c d e f t*



d blocked ⇒ initiate decrease flow and result of <u>iteration 1:</u> make flow in (*c*,*d*) = 0

- Second flow increase (*c* is blocked. Balance)



- Third flow increase
  - *a* is blocked ⇒ make flow <*s*, *a*> = 5
  - We are done since every path from *s* to *t* is blocked
  - Blocking flow = 5 units

Complexity result

- Wave method computes blocking flow of an acyclic graph in $O(n^2)$ time (& blocking flow of a general graph in $O(n^3)$ time)
- Proof:
  - If a node i is blocked, every path from i to t is blocked
  - Initially s is blocked
  - After increase flow step, if the balancing is a success, ∃ no unblocked, unbalanced nodes
  - If balancing fails, ∃ a blocked, unbalanced node
  - This blocked node is balanced during decrease flow step & remains balanced during subsequent increase flow steps

    ⇒ We block at least one node in each step

    ⇒ At most $(n - 1)$ steps

    ⇒ At each step of increase flow, either an edge is saturated or terminates in a balance

    ⇒ Similarly at each step of decrease flow either an edge flow is set to zero or terminates in a balance

    ⇒ $O(2m) + (n-1)(n-2)$ operations ⇒ $O(n^2)$
  - $O(n^3)$ complexity for max. flow follows from our earlier discussion w.r.t. DMKM algorithm

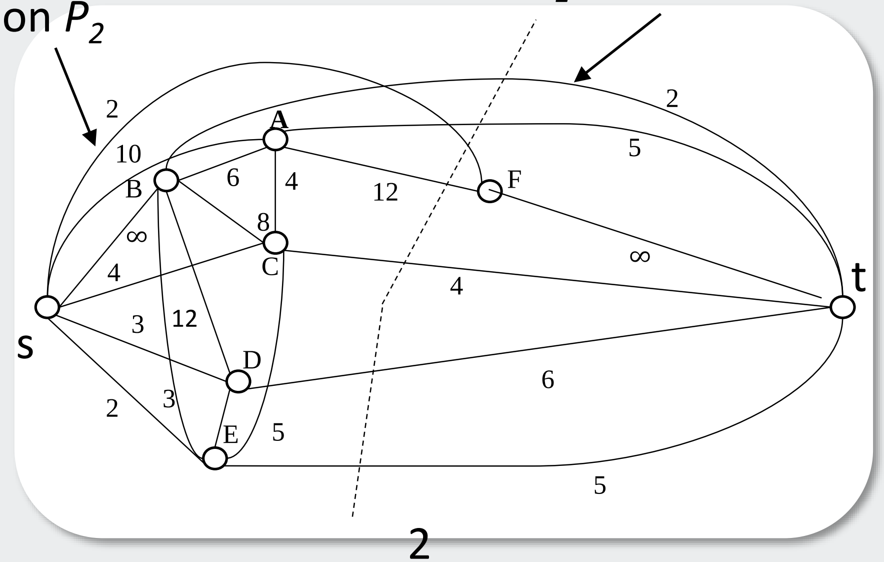
More Recent Algorithms

- D. D. Sleator and R. Tarjan, “A data structure for dynamic trees,” J. of Comput. Sys. Sci., vol. 26, pp. 362-91, 1983
- Y. Shiloach and U. Vishkin, “An $O(n^2 \log n)$ parallel max-flow algorithm”, J. of Algorithms, vol. 3, pp. 128-46, 1982
- N. Gabow, “Scaling algorithms for network problems”, J. of Comput. Sys. Sci., pp.260-270, 1981
- R. E. Tarjan, “A simple version of Karzanov's blocking flow algorithm,” OR letters, vol. 2, pp 265-268, 1984
- Goldberg, A. V., “Efficient graph algorithms for sequential & parallel computers,” Ph.D. thesis, LCS, MIT, 1987
- Bertsekas, D. P., Linear network optimization, MIT press, 1991

Mapping Problem

Processing times on P_2

Processing times on P_1



- Set of tasks A,B,…, F with a graph structure
- Arcs ⇒ communication time
- Processing times on two processors: t_{i1}, t_{i2}
- Problem: minimize (processing time + communication time)

$$\text{Tasks for } P_2 = \{F\}$$
$$\text{Tasks for } P_1 = \{A, B, C, D, E\}$$

- Total cost: 36 = cap. min. cut
- Makes sense since for an arbitrary partition of tasks: $(W, \overline{W})$

$$\text{total cost: } \sum_{i\in W} t_{i1} + \sum_{i\in \overline{W}} t_{i2} + \sum_{\substack{<i,j> \\ i\in W \\ j\in \overline{W}}} c_{ij}$$

- Establishing formal equivalence:

$$\text{let } x_i = \begin{cases} 1 & \text{if task } i \text{ is allocated to } P_1 \\ 0 & \text{otherwise} \end{cases}$$
$$y_i = \begin{cases} 1 & \text{if task } i \text{ is allocated to } P_2 \\ 0 & \text{otherwise} \end{cases}$$

⇒ Need: $x_i + y_i = 1, \forall i$

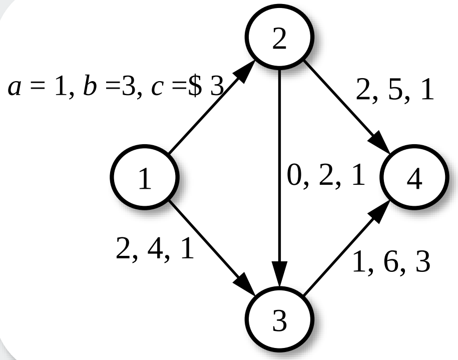
Mapping Problem

- Cost function: $\sum_{i=1}^{n} t_{i1} x_i + \sum_{j=1}^{n} t_{i2} y_j + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_i y_j$
- Define $x_i y_j = \mu_{ij}$
- Then $x_i + y_j - \mu_{ij} \geq 0$
- The problem is:

$$\left.\begin{aligned} \min\ & \sum_i t_{i1} x_i + \sum_j t_{i2} y_j + \sum_i \sum_{\substack{j=1 \\ j \neq i}} c_{ij} \mu_{ij} \\ \text{s.t.}\ & x_i + y_j \geq 1 \\ & x_i + y_j - \mu_{ij} \geq 0 \\ & \mu_{ij} \geq 0 \end{aligned}\right\} \text{Similar to dual of max. flow}$$

- <u>Note</u>: can't extend to more than two processors

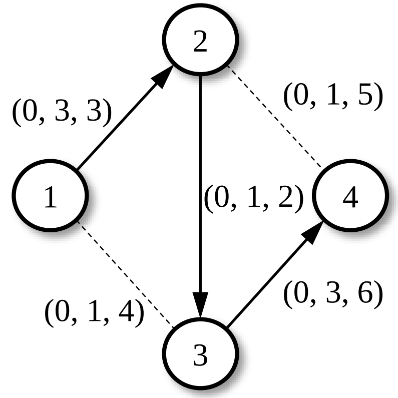
PERT networks



a = Min. time to perform a task
b = Normal completion time
c = \$ to be spent to reduce completion time by one unit

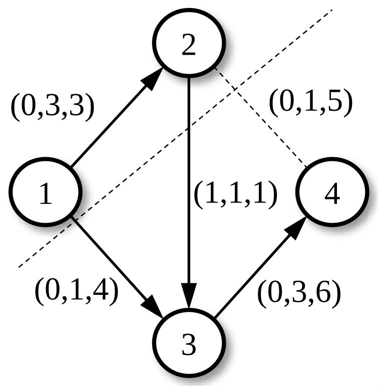
- If spend \$0; project completes in 3 + 2 + 6 = 11 days
  - Critical path 1 – 2 – 3 – 4
- If want to reduce the time, must spend \$'s on tasks 1 – 2, 2 – 3, 3 – 4, since they are on the critical path
- Also, must spend on tasks with lowest cost per unit time ⇒ task 2 – 3
- Q: How far should we reduce?
- Answer
  - Till the arc is reduced to the minimum time a_{ij}
    - If this occurs, pick arc with the next lower cost per unit time
  - (or) path is no longer the critical path

How to decide where to invest?



(a, c, b)
a = amount spent on arc
c = \$/unit time
b = current processing time

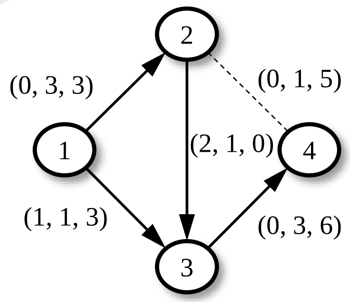
- Reduce <2, 3> by one unit
  - ⇒ Two critical paths 1 – 2 – 3 – 4 and 1 – 3 – 4



- To shorten longest paths, have three choices:
  - 1 – 2 & 1 – 3 with $c_{12} + c_{13} = 3 + 1 = 4$
  - 2 – 3 & 1 – 3 with $c_{23} + c_{13} = 1 + 1 = 2$
  - 3 – 4 with cost $c_{34} = 3$

Where to invest?

- Looks like a min. cut of a graph of active arcs
  - 2 – 3 & 1 – 3
- Note: Can't reduce 2 – 3 any further

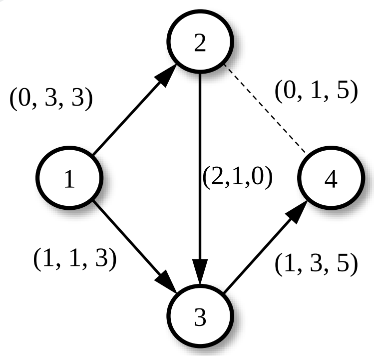


two choices

1 – 2 & 1 – 3 with $c_{12} + c_{13} = 4$

3 – 4 with cost $c_{34} = 3$

- Reduce c_{34} by one unit, since then 1 – 2 – 4 is also a critical path

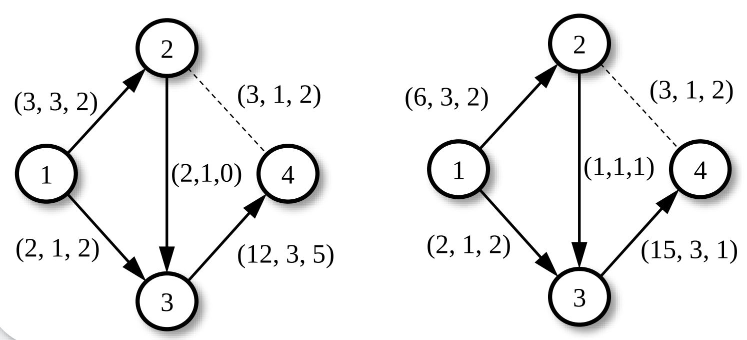


two choices

$c_{12} + c_{13} = 4$

$c_{24} + c_{34} = 4$

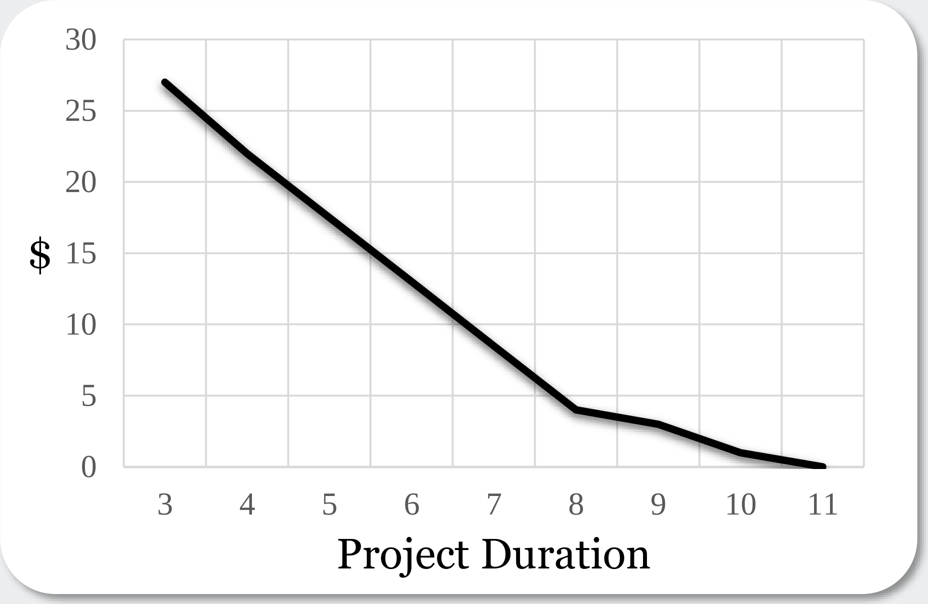
$\Rightarrow$



- Now 1 – 2, 2 – 4, & 2 – 3 are rigid

Trade-off curve

- If we reduce 1 – 3 & 3 – 4 to their value & increase 2 – 3 w/o affecting the longest path
  - \$0 ⇒ 11 days; \$1⇒10 days; \$3 ⇒ 9 days; \$4 ⇒ 8 days; \$22 ⇒ 4 days; \$27 for 3 days



Summary

- Max. flow ≡ Min. cut
- Ford-Fulkerson labeling algorithm
  - Exponential and can converge to non-optimal solutions
  - Can fix the problem by computing shortest augmenting paths rather than any augmenting path
- DMKM algorithm
  - Push-pull version
  - Wave method
- Applications of maximum flow (mapping, PERT)