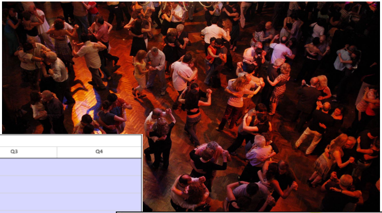
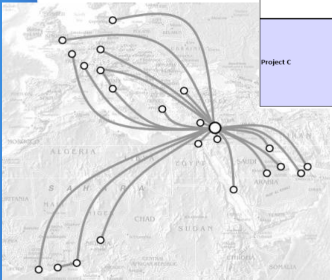
Modeling, Analysis and Optimization of Networks Flows

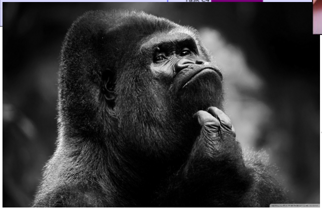
Alberto Ceselli
Dipartimento di Informatica, Università degli Studi di Milano

A.Y. 2022/2023

Trivia: how many networks do you see?





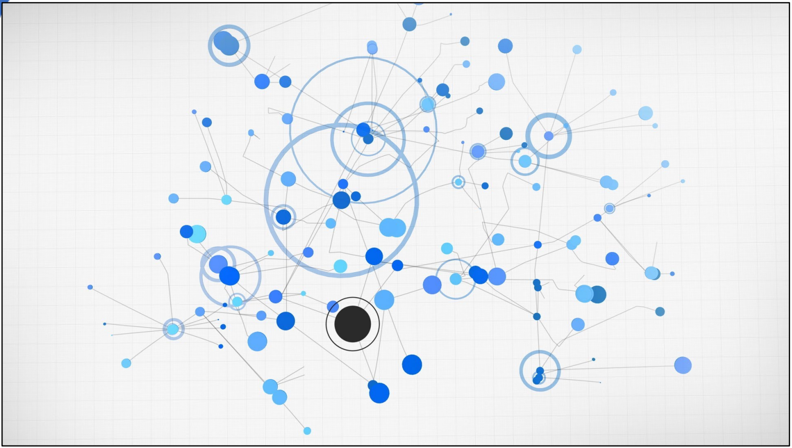


$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}.$$

What's a Network ?

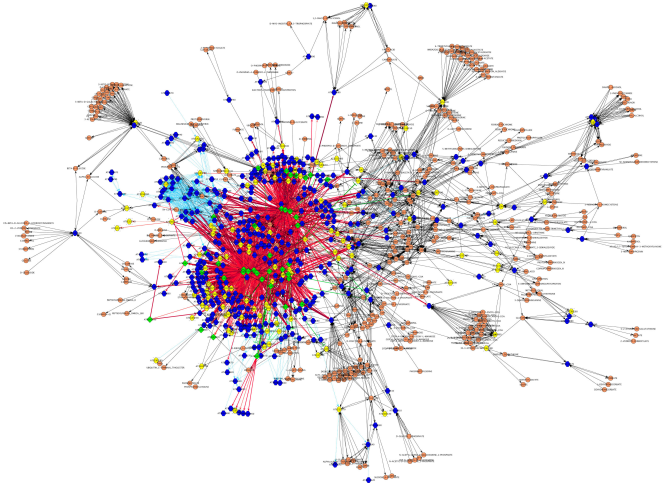
- Dictionary.com:
  any netlike combination of filaments, lines, veins, passages, or the like
- Wikipedia:
  disambiguation page with 30 entries in 4 categories
- Let's try images !

A few images after: artistic ones



A few images after: artistic ones

… and very different domains



… only rank 26!

To remember:

1 - Networks are pervasive !

2 - by « network » we mean far more than computers connected by cables;

3 - network problems moved from technologies to applications, and now to services;

4 - networks are in general too complex to be managed by humans without decision support systems.

Modeling, Analysis and Optimization of Networks

- A « transverse » course offered by D.I.
- (Far) More on modeling and structural properties than on specific techniques and technologies
- Different editions cover different sub-topics

- This year: **network flows**

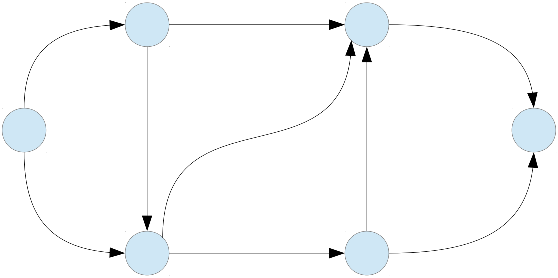
Course objectives

- finding network (flow) structures in applications
- modeling as flow problems on networks
- main theoretical results on flows and flow algorithms
- overview of tools for solving network flow problems

PART I: Modeling *with* network flows

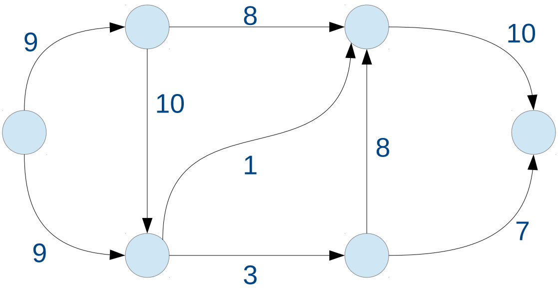
Notation

- A directed graph G(V,A)



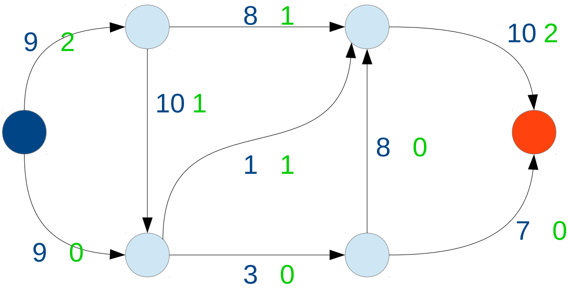
Notation

- A directed graph G(V,A)
- A function c: A → ℝ (arc capacities)



Notation

- A directed graph G(V,A)
- A function c: A → ℝ (arc capacities)
- Special nodes s (source) and t (sink)
- A flow is a function f: A → $\mathbb{R}_+$



Notation

- Let $\partial^-(i)$ be the set of incoming arcs in i
- Let $\partial^+(i)$ be the set of outgoing arcs from i
- A flow is **feasible** iff
  - $f(i,j) \leq c(i,j)$
    for each (i,j) in A (capacity constr.)
  - $f(i,j) \geq 0$
    for each (i,j) in A (non-negativity constr.)
  - $\sum_{j\in\partial-(i)} f(j,i) = \sum_{j\in\partial+(i)} f(i,j)$
    for each i in V \ {s,t} (flow conservation constr.)
- A flow is **maximum** if
  $\sum_{j\in\partial+(s)} f(s,j) = \sum_{j\in\partial-(t)} f(j,t)$
  is maximum

How to *model* with flows ?

- Given an application
- Design V, s and t
- Design A
- Design c()
- Find f()
- Give to f() an interpretation in the original application

How to *learn* modeling with flows ?

- As for math modeling in general, no specific recipe
  - A bit of theory
  - A few modeling tricks and gadgets
  - Training on a few examples
  - + Intuition :)

How to learn modeling with flows ?

- As for math modeling in general, no specific recipe
  - A bit of theory
  - A few modeling tricks and gadgets
  - Training on a few examples
  - + Intuition :)

References

These lectures are taken from:

- J. Kleinberg, É. Tardos "Algorithm Design", Person (2005)
- (R. Ahuja, T. Magnanti, J. Orlin "Network Flows: Theory, Algorithms, and Applications", Pearson (1993))

Slides are (partially) taken from:

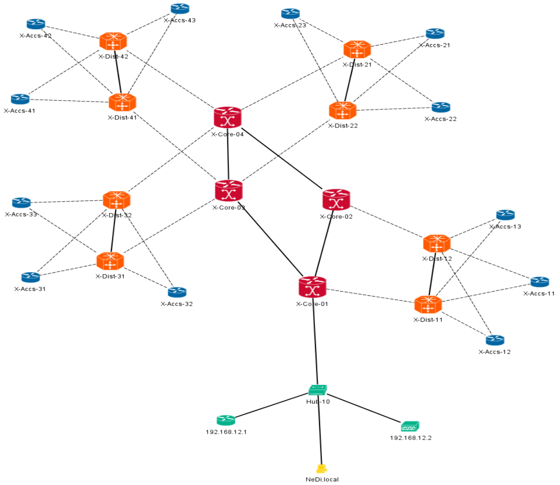
- Kevin Wayne (Algorithm Design)
- James Orlin (Network Flows)

Example 1: routing in communication networks

Given a communication network composed by nodes and links, and a special pair of nodes (source and destination) that need to transfer data

- A Path Problem
  - Find a path from source to destination in the network
- A Backup Path Problem
  - Find **two** paths from source to destination in the network having **no common arc**
- A network robustness Problem
  - find **how many** paths are there from source to destination, having **no common arc**

- From X-Accs-33 to X-Accs-41
- From X-Core-01 to X-Dist-21



Example 2: Tango Dancers problem

Taken from J. Kleinberg, E. Tardos, « Algorithm Design»

- Given :
  - a set G of gentlemen and a set L of ladies
  - a set of compatibilities
  - find how many couples can be on the dance floor at the same time, at most
- i.e. a max matching problem on a bipartite graph
- **Observation:** the number of couples equals the flow, that in turn equals the number of arcs whose capacity constraints are active!

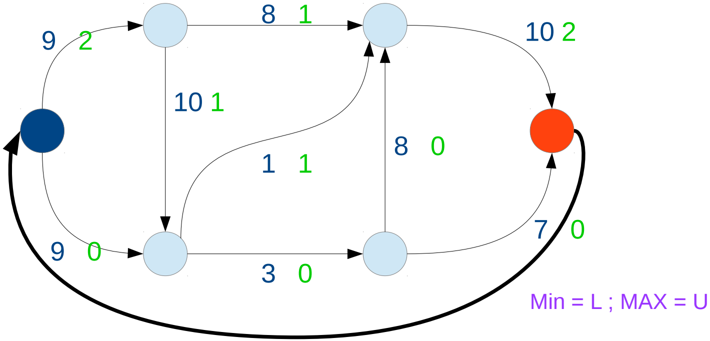
A few extensions (1)

- Capacity on nodes
- Multiple sources and multiple sinks
- Integral flows

(blackboard discussion)

A few extensions (2)

- Flows and Circulations



- Circulations with demands
- Lower bound on flow on each arc

(blackboard discussion)

Example 3: consistent rounding problem

- Did you ever compile tax forms?
- Given a pxq matrix D containing real values d_{ij} with row sums a_i and column sums b_j.
- You want to round **both** matrix coefficients **and** row/col sums in a **consistent** way
- The decision to round up or down is up to you

Example 3: consistent rounding problem

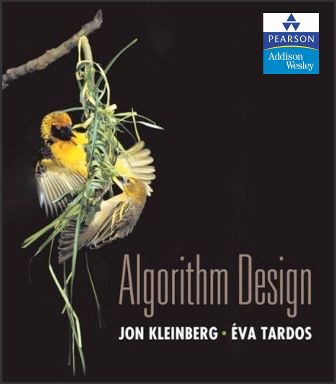
- From N.F., Application 6.3

3.1	6.8	7.3	17.2
9.6	2.4	0.7	12.7
3.6	1.2	6.5	11.3
16.3	10.4	14.5	

Roadmap

Outline:

- To review a few modeling frameworks (e.g. graphs and mathematical programming)
- To learn how to *model with flows and cuts* complex decision problems
- To understand theory and algorithms underlying network flows



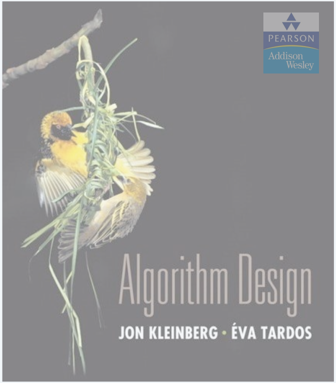
3. GRAPHS

- *basic definitions and applications*
- *graph connectivity and graph traversal*
- *testing bipartiteness*
- *connectivity in directed graphs*
- *DAGs and topological ordering*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on 7/29/17 6:09 PM



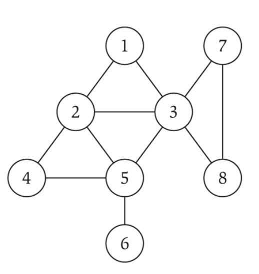
3. Graphs

- *basic definitions and applications*
- *graph connectivity and graph traversal*
- *testing bipartiteness*
- *connectivity in directed graphs*
- *DAGs and topological ordering*

Undirected graphs

Notation. $G = (V, E)$

- V = nodes (or vertices).
- E = edges (or arcs) between pairs of nodes.
- Captures pairwise relationship between objects.
- Graph size parameters: $n = |V|, m = |E|$.

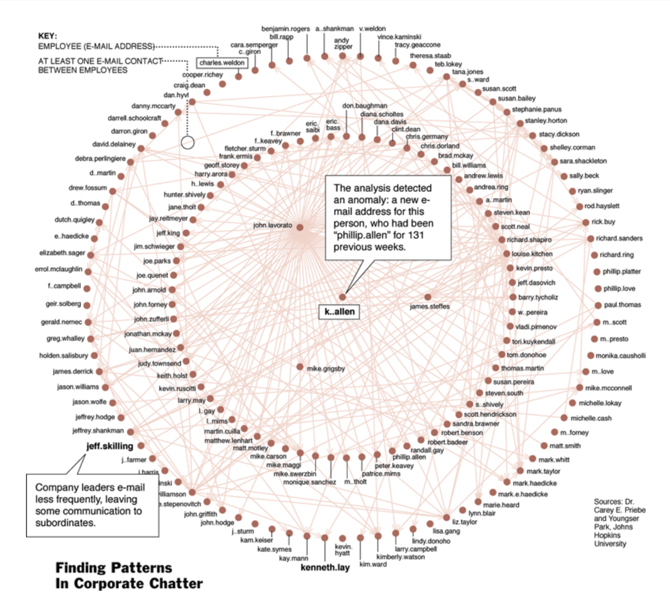


$V = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$

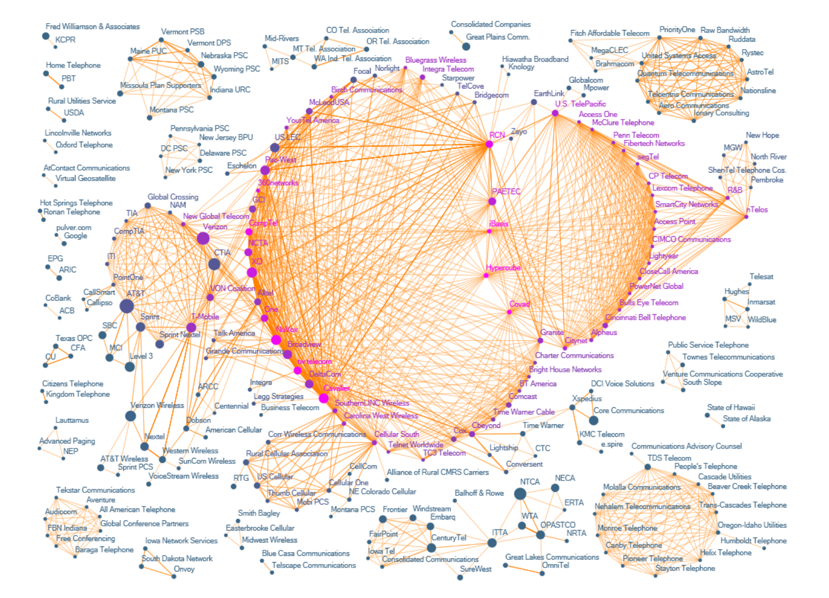
$E = \{ 1–2, 1–3, 2–3, 2–4, 2–5, 3–5, 3–7, 3–8, 4–5, 5–6, 7–8 \}$

$m = 11, n = 8$

One week of Enron emails



The evolution of FCC lobbying coalitions



"The Evolution of FCC Lobbying Coalitions" by Pierre de Vries in JoSS Visualization Symposium 2010

Framingham heart study

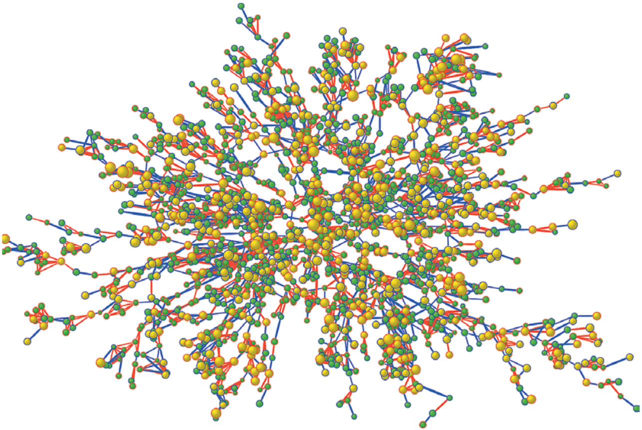


Figure 1. Largest Connected Subcomponent of the Social Network in the Framingham Heart Study in the Year 2000.
Each circle (node) represents one person in the data set. There are 2200 persons in this subcomponent of the social network. Circles with red borders denote women, and circles with blue borders denote men. The size of each circle is proportional to the person's body-mass index. The interior color of the circles indicates the person's obesity status: yellow denotes an obese person (body-mass index, ≥30) and green denotes a nonobese person. The colors of the ties between the nodes indicate the relationship between them: purple denotes a friendship or marital tie and orange denotes a familial tie.

"The Spread of Obesity in a Large Social Network over 32 Years" by Christakis and Fowler in New England Journal of Medicine, 2007

Some graph applications

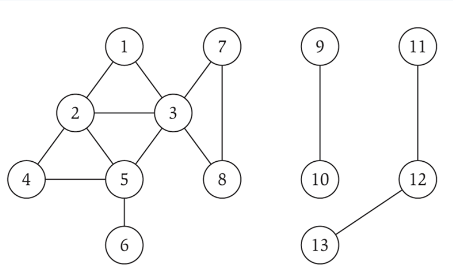
graph	node	edge
communication	telephone, computer	fiber optic cable
circuit	gate, register, processor	wire
mechanical	joint	rod, beam, spring
financial	stock, currency	transactions
transportation	street intersection, airport	highway, airway route
internet	class C network	connection
game	board position	legal move
social relationship	person, actor	friendship, movie cast
neural network	neuron	synapse
protein network	protein	protein-protein interaction
molecule	atom	bond

Paths and connectivity

Def. A path in an undirected graph $G = (V, E)$ is a sequence of nodes $v_1, v_2, \ldots, v_k$ with the property that each consecutive pair v_{i-1}, v_i is joined by an edge in E.

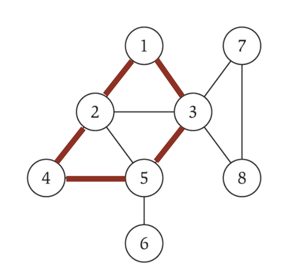
Def. A path is simple if all nodes are distinct.

Def. An undirected graph is connected if for every pair of nodes u and v, there is a path between u and v.



Cycles

Def. A cycle is a path $v_1, v_2, \ldots, v_k$ in which $v_1 = v_k$, $k > 2$, and the first $k - 1$ nodes are all distinct.



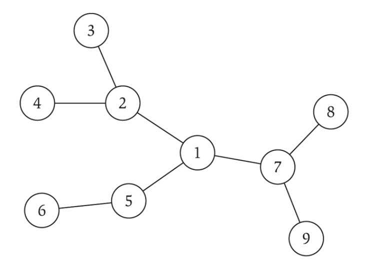
cycle C = 1–2–4–5–3–1

Trees

Def. An undirected graph is a tree if it is connected and does not contain a cycle.

Theorem. Let G be an undirected graph on n nodes. Any two of the following statements imply the third:

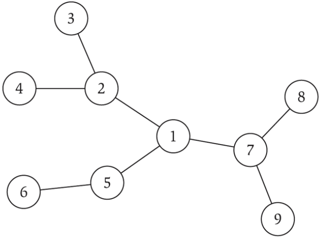
- G is connected.
- G does not contain a cycle.
- G has $n-1$ edges.



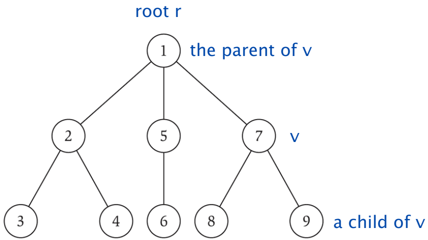
Rooted trees

Given a tree T, choose a root node r and orient each edge away from r.

Importance. Models hierarchical structure.



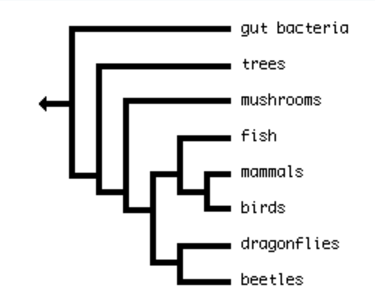
a tree



the same tree, rooted at 1

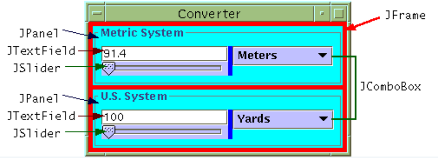
Phylogeny trees

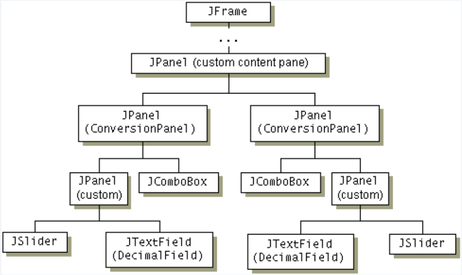
Describe evolutionary history of species.



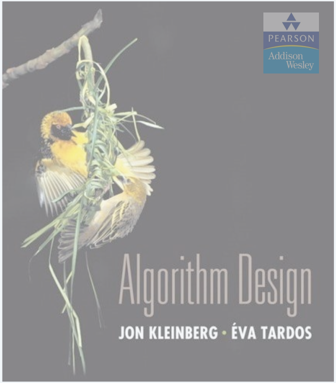
GUI containment hierarchy

Describe organization of GUI widgets.





http://java.sun.com/docs/books/tutorial/uiswing/overview/anatomy.html



3. Graphs

- *basic definitions and applications*
- *graph connectivity and graph traversal*
- *testing bipartiteness*
- *connectivity in directed graphs*
- *DAGs and topological ordering*

Connectivity

s-t connectivity problem. Given two nodes s and t, is there a path between s and t?

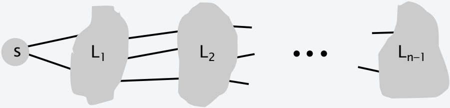
s-t shortest path problem. Given two nodes s and t, what is the length of a shortest path between s and t?

Applications.

- Friendster.
- Maze traversal.
- Kevin Bacon number.
- Fewest hops in a communication network.

Breadth-first search

BFS intuition. Explore outward from s in all possible directions, adding nodes one "layer" at a time.



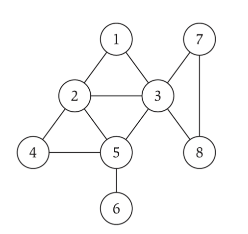
BFS algorithm.

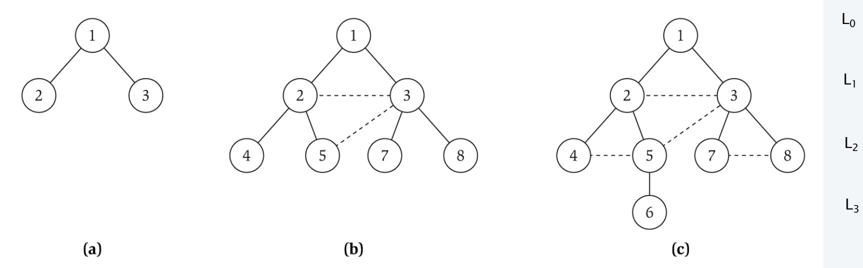
- $L_0 = \{ s \}$.
- $L_1 =$ all neighbors of L_0.
- $L_2 =$ all nodes that do not belong to L_0 or L_1, and that have an edge to a node in L_1.
- $L_{i+1} =$ all nodes that do not belong to an earlier layer, and that have an edge to a node in L_i.

Theorem. For each i, L_i consists of all nodes at distance exactly i from s. There is a path from s to t iff t appears in some layer.

Breadth-first search

Property. Let T be a BFS tree of $G = (V, E)$, and let (x, y) be an edge of G. Then, the levels of x and y differ by at most 1.





Breadth-first search: analysis

Theorem. The above implementation of BFS runs in $O(m + n)$ time if the graph is given by its adjacency representation.

Pf.

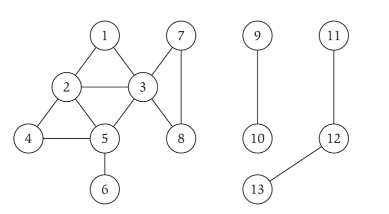
- Easy to prove $O(n^2)$ running time:
  - at most n lists $L[i]$
  - each node occurs on at most one list; for loop runs $\leq n$ times
  - when we consider node u, there are $\leq n$ incident edges (u, v), and we spend $O(1)$ processing each edge

- Actually runs in $O(m + n)$ time:
  - when we consider node u, there are $degree(u)$ incident edges (u, v)
  - total time processing edges is $\Sigma_{u \in V}\, degree(u) \;=\; 2m$. ▪

each edge (u, v) is counted exactly twice
in sum: once in degree(u) and once in degree(v)

Connected component

Connected component. Find all nodes reachable from s.



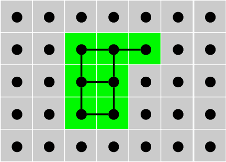
Connected component containing node $1 = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$.

Flood fill

Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.

- Node: pixel.
- Edge: two neighboring lime pixels.
- Blob: connected component of lime pixels.

recolor lime green blob to blue

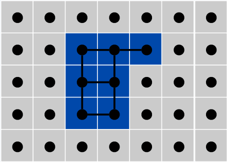


Flood fill

Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.

- Node: pixel.
- Edge: two neighboring lime pixels.
- Blob: connected component of lime pixels.

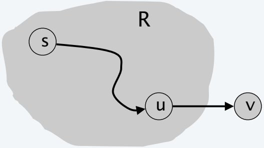
recolor lime green blob to blue



Connected component

Connected component. Find all nodes reachable from s.

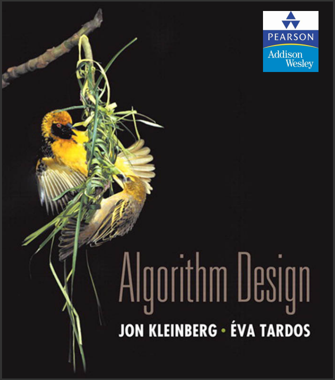
```
R will consist of nodes to which s has a path
Initially R = {s}
While there is an edge (u, v) where u ∈ R and v ∉ R
  Add v to R
Endwhile
```



it's safe to add v

Theorem. Upon termination, R is the connected component containing s.

- BFS = explore in order of distance from s.
- DFS = explore in a different way.

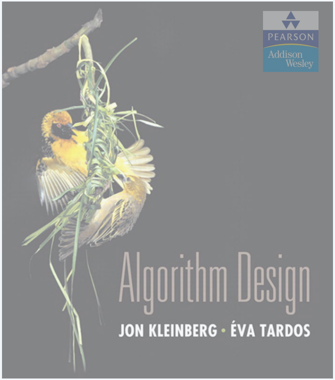


Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

7. Network Flow I

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *capacity-scaling algorithm*
- *shortest augmenting paths*
- *Dinitz' algorithm*
- *simple unit-capacity networks*



SECTION 7.1

7. NETWORK FLOW I

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *capacity-scaling algorithm*
- *shortest augmenting paths*
- *Dinitz' algorithm*
- *simple unit-capacity networks*

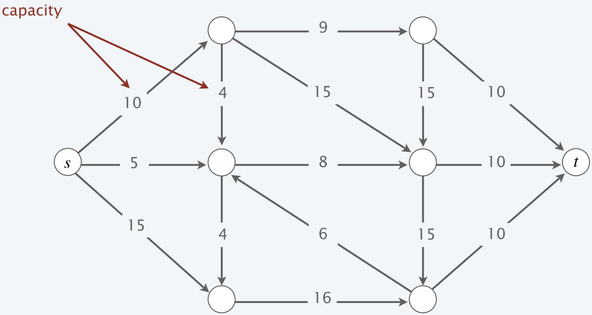
Flow network

A flow network is a tuple $G = (V, E, s, t, c)$.

- Digraph (V, E) with source $s \in V$ and sink $t \in V$.
- Capacity $c(e) > 0$ for each $e \in E$.

assume all nodes are reachable from s

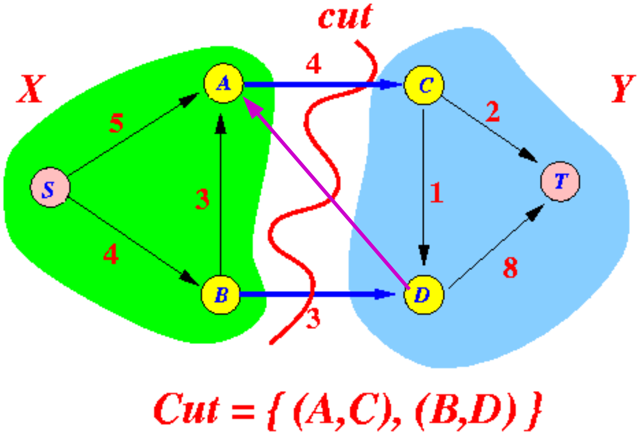
Intuition. Material flowing through a transportation network; material originates at source and is sent to sink.



Flows and Cuts

- Intuitively (s-t) **Cut**: a set of arcs whose removal makes sink unreachable from source
  (no more path exists going from s to t)
- Formally, a cut is a **partition** [S,V \ S] of the nodes of the graph (s-t cut if s is in S and t in V \ S)
- Arcs of the cut are those having one endpoint in S and the other in V \ S;
  - forward arc: (i,j) with i in S and j in V \ S
  - backward arc: (i,j) with i in V \ S and j in S
- The **capacity** of a cut is the sum of capacities of its **forward** arcs
- A cut is **minimum** if its capacity is minimum

Flows and cuts

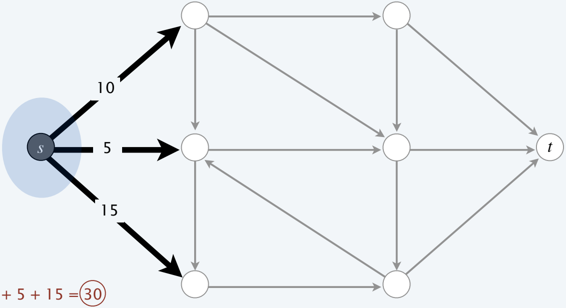


Minimum-cut problem

Def. An *st*-cut (cut) is a partition (A, B) of the nodes with $s \in A$ and $t \in B$.

Def. Its capacity is the sum of the capacities of the edges from A to B.

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



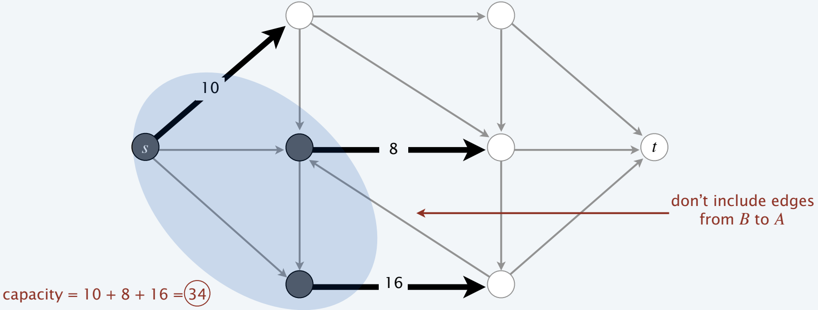
capacity = 10 + 5 + 15 = 30

Minimum-cut problem

Def. An *st*-cut (cut) is a partition (A, B) of the nodes with $s \in A$ and $t \in B$.

Def. Its capacity is the sum of the capacities of the edges from A to B.

$$cap(A,B) = \sum_{e \text{ out of } A} c(e)$$



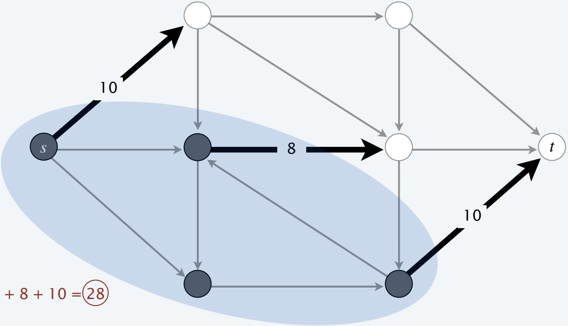
Minimum-cut problem

Def. An *st-cut (cut)* is a partition (A, B) of the nodes with $s \in A$ and $t \in B$.

Def. Its capacity is the sum of the capacities of the edges from A to B.

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$

Min-cut problem. Find a cut of minimum capacity.



capacity = 10 + 8 + 10 = 28

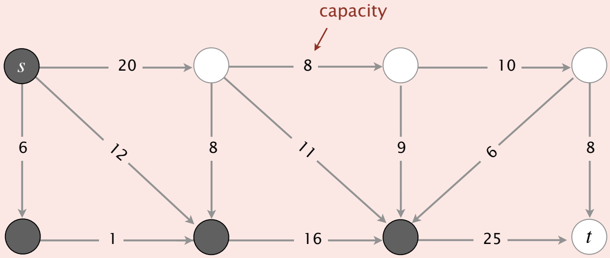
Which is the capacity of the given st-cut?

A. 11 (20 + 25 – 8 – 11 – 9 – 6)

B. 34 (8 + 11 + 9 + 6)

C. 45 (20 + 25)

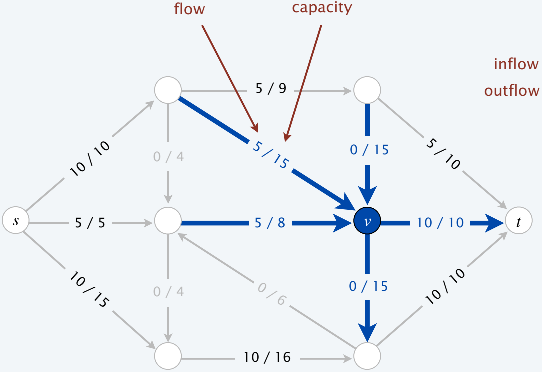
D. 79 (20 + 25 + 8 + 11 + 9 + 6)



Maximum-flow problem

Def. An *st*-flow (flow) f is a function that satisfies:

- For each $e \in E$: $0 \le f(e) \le c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]



inflow at v = $5 + 5 + 0$ = 10

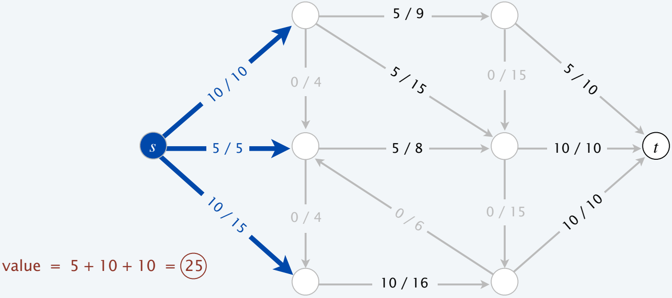
outflow at v = $10 + 0$ = 10

Maximum-flow problem

Def. An *st*-flow (flow) f is a function that satisfies:

- For each $e \in E$: $\quad 0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\quad \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

Def. The value of a flow f is: $val(f) = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e)$



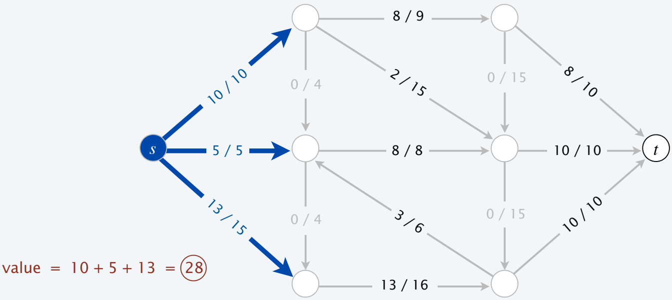
Maximum-flow problem

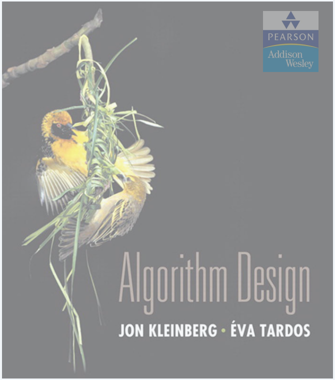
Def. An *st*-flow (flow) f is a function that satisfies:

- For each $e \in E$: $\quad 0 \le f(e) \le c(e)$ $\quad$ [capacity]
- For each $v \in V - \{s, t\}$: $\quad \displaystyle\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ $\quad$ [flow conservation]

Def. The value of a flow f is: $\quad val(f) = \displaystyle\sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e)$

Max-flow problem. Find a flow of maximum value.





SECTION 7.1

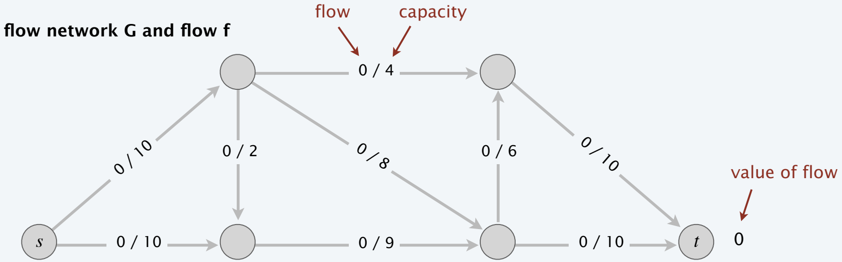
7. NETWORK FLOW I

- max-flow and min-cut problems
- Ford–Fulkerson algorithm
- max-flow min-cut theorem
- capacity-scaling algorithm
- shortest augmenting paths
- Dinitz' algorithm
- simple unit-capacity networks

Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

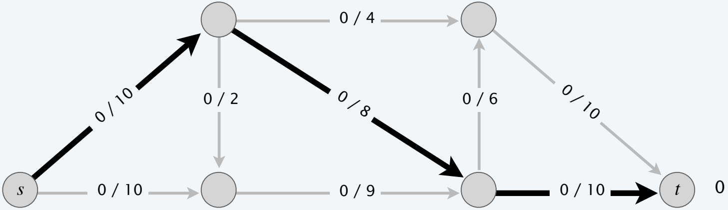


Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

flow network G and flow f

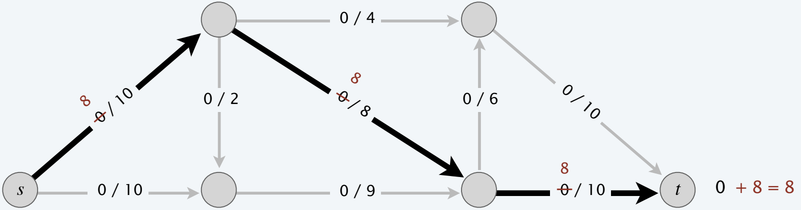


Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

flow network G and flow f

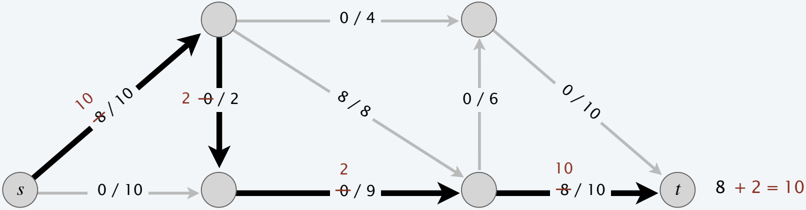


Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

flow network G and flow f

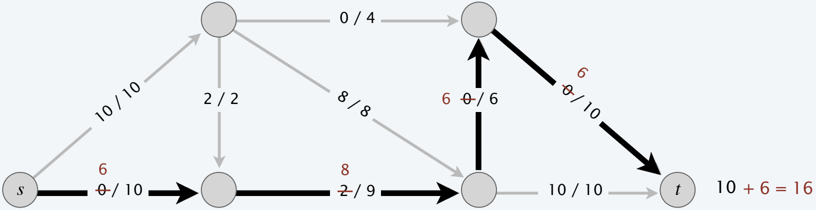


Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

flow network G and flow f



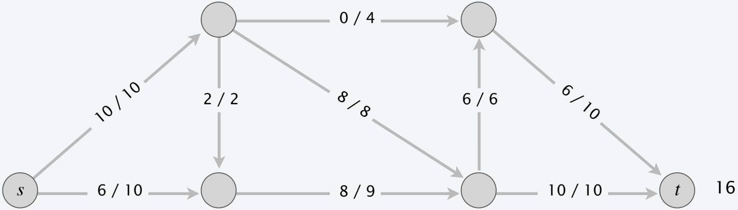
Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

ending flow value = 16

flow network G and flow f



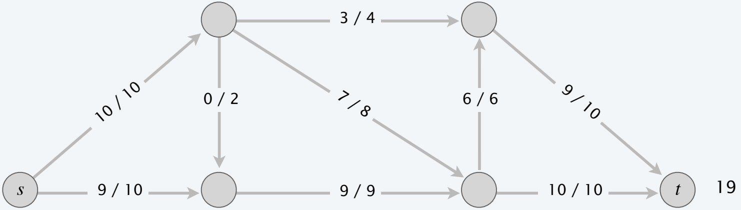
Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

but max-flow value = 19

flow network G and flow f



Why the greedy algorithm fails

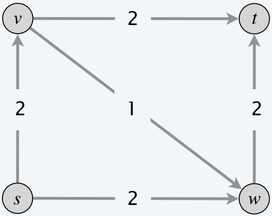
Q. Why does the greedy algorithm fail?

A. Once greedy algorithm increases flow on an edge, it never decreases it.

Ex. Consider flow network G.

- The unique max flow has $f^*(v, w) = 0$.
- Greedy algorithm could choose $s \rightarrow v \rightarrow w \rightarrow t$ as first augmenting path.

flow network G



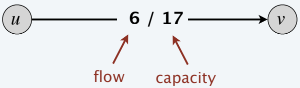
Bottom line. Need some mechanism to "undo" a bad decision.

Residual network

Original edge. $e = (u, v) \in E$.

- Flow $f(e)$.
- Capacity $c(e)$.

original flow network G



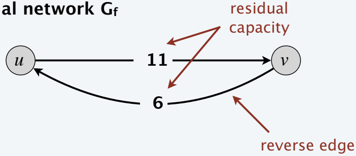
Reverse edge. $e^{\text{reverse}} = (v, u)$.

- "Undo" flow sent.

Residual capacity.

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^{\text{reverse}} \in E \end{cases}$$

residual network G_f



Residual network. $G_f = (V, E_f, s, t, c_f)$.

edges with positive residual capacity

where flow on a reverse edge negates flow on corresponding forward edge

- $E_f = \{e : f(e) < c(e)\} \cup \{e^{\text{reverse}} : f(e) > 0\}$.
- Key property: f' is a flow in G_f iff $f + f'$ is a flow in G.

Augmenting path

Def. An augmenting path is a simple $s \rightsquigarrow t$ path in the residual network G_f.

Def. The bottleneck capacity of an augmenting path P is the minimum residual capacity of any edge in P.

Key property. Let f be a flow and let P be an augmenting path in G_f. Then, after calling $f' \leftarrow$ AUGMENT(f, c, P), the resulting f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

AUGMENT(f, c, P)

$\delta \leftarrow$ bottleneck capacity of augmenting path P.

FOREACH edge $e \in P$:

  IF $(e \in E)$ $f(e) \leftarrow f(e) + \delta$.

  ELSE $f(e^{\text{reverse}}) \leftarrow f(e^{\text{reverse}}) - \delta$.

RETURN f.

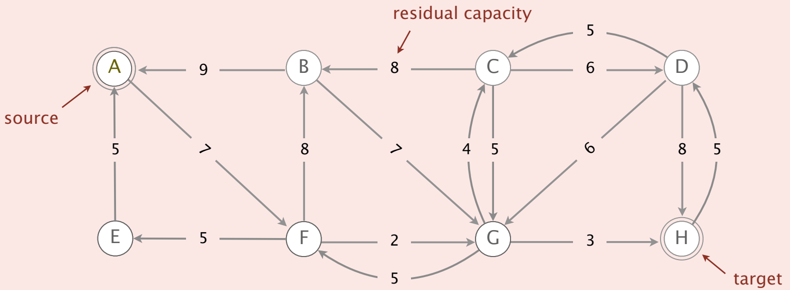
Which is the augmenting path of highest bottleneck capacity?

A. $A \to F \to G \to H$

B. $A \to B \to C \to D \to H$

C. $A \to F \to B \to G \to H$

D. $A \to F \to B \to G \to C \to D \to H$



Ford–Fulkerson algorithm

Ford–Fulkerson augmenting path algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P in the residual network G_f.
- Augment flow along path P.
- Repeat until you get stuck.

FORD–FULKERSON(G)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f.

WHILE (there exists an s↝t path P in G_f)

  $f \leftarrow$ AUGMENT(f, c, P).

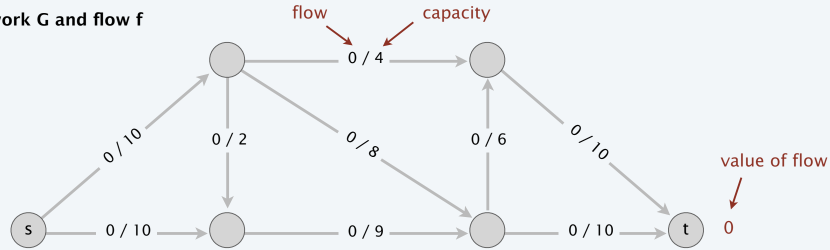
  Update G_f.

RETURN f.

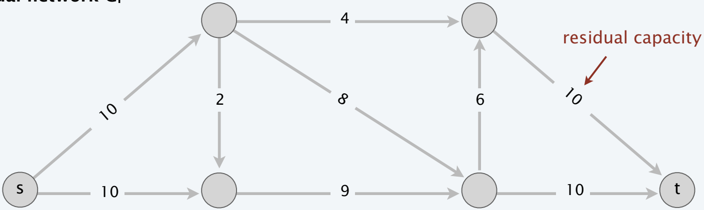
augmenting path

Ford–Fulkerson algorithm demo

network G and flow f

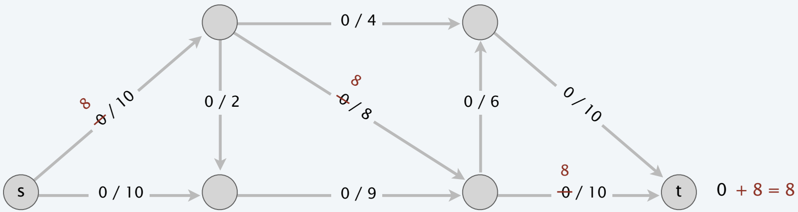


residual network G_f

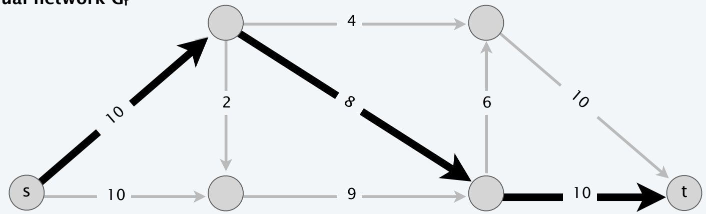


Ford–Fulkerson algorithm demo

network G and flow f

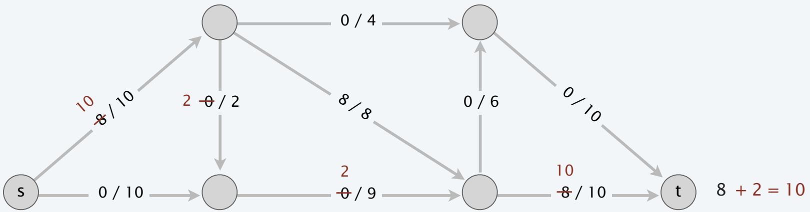


residual network G_f

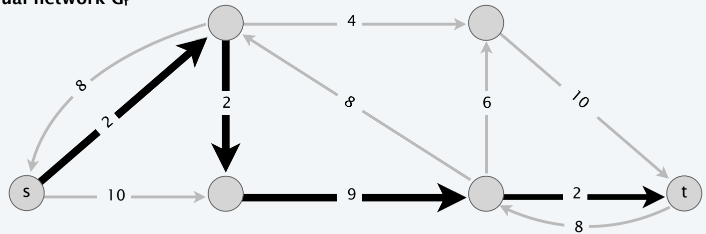


Ford–Fulkerson algorithm demo

network G and flow f

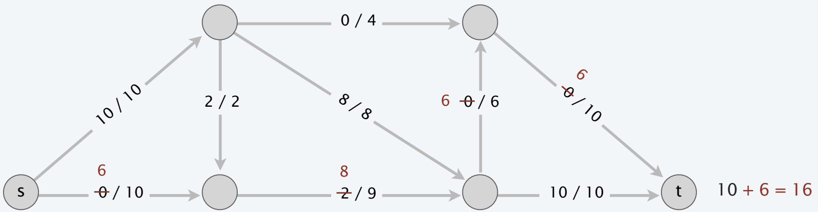


residual network G_f

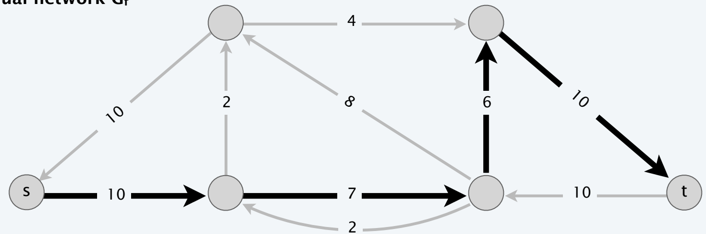


Ford–Fulkerson algorithm demo

network G and flow f

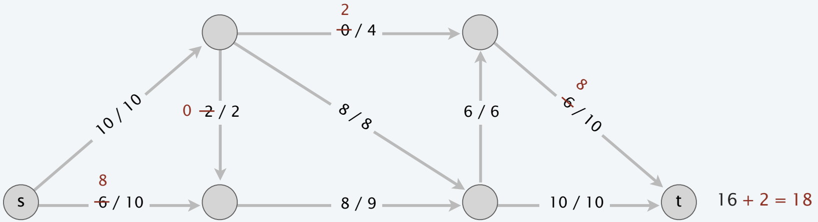


residual network G_f

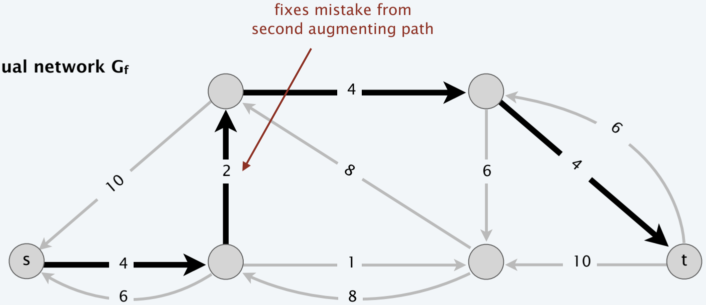


Ford–Fulkerson algorithm demo

network G and flow f

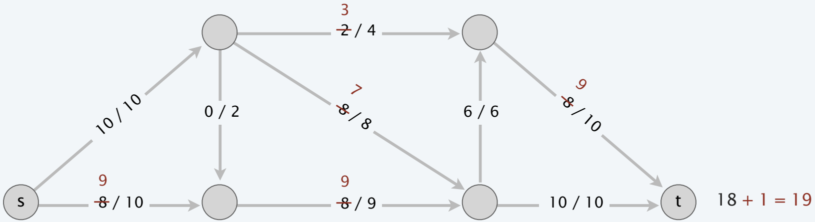


residual network G_f

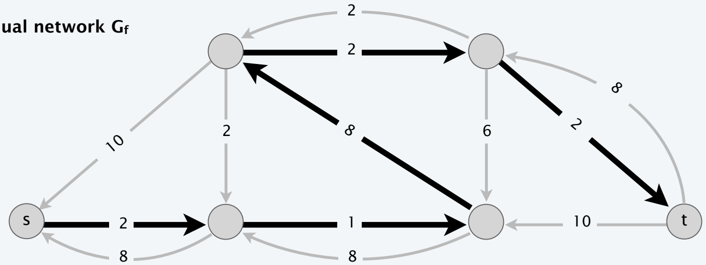


Ford–Fulkerson algorithm demo

network G and flow f

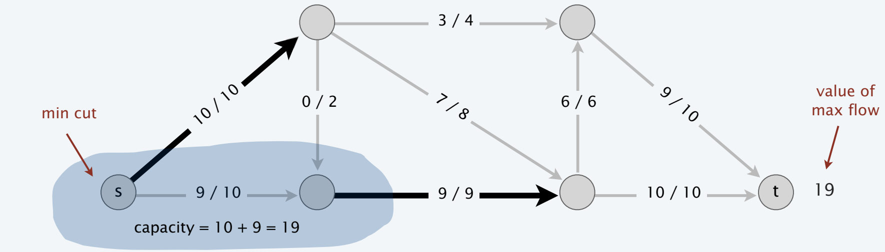


residual network G_f

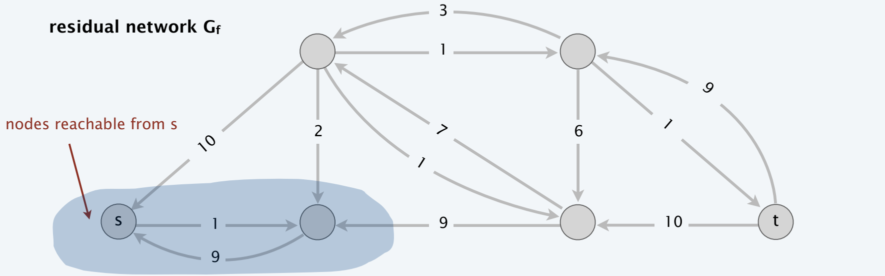


Ford–Fulkerson algorithm demo

network G and flow f



residual network G_f



Modeling, Analysis and Optimization of Networks Flows

Alberto Ceselli
Dipartimento di Informatica, Università degli Studi di Milano

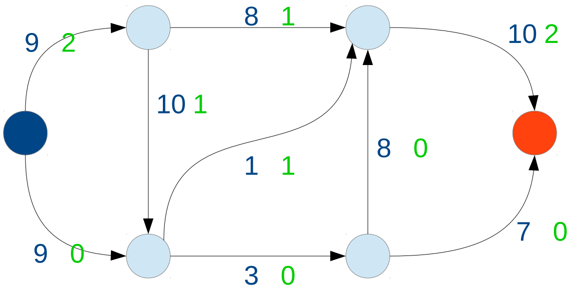
A.Y. 2022/2023

Lecture 2

Lecture 2

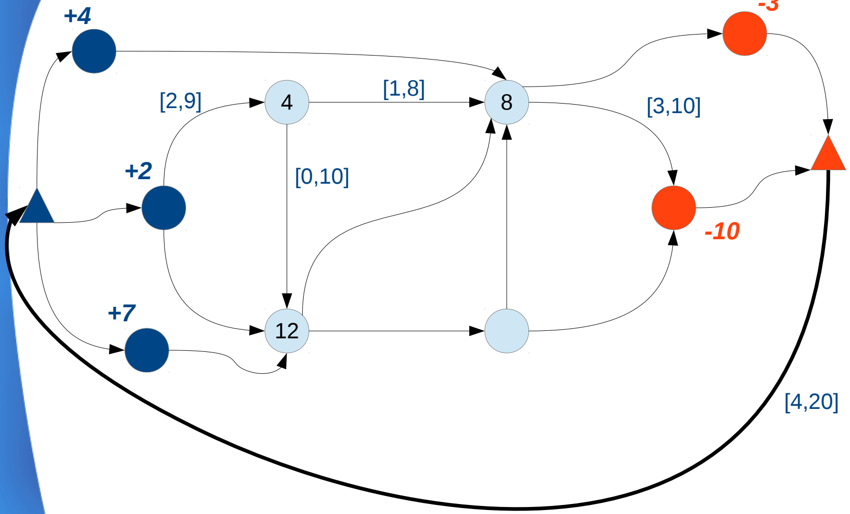
Summary of Lecture 1

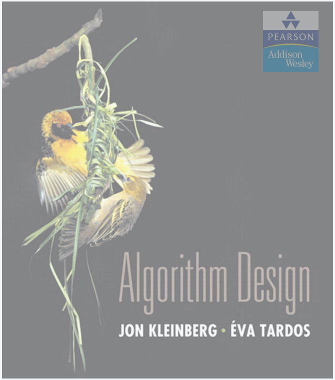
- Given :
  - a graph G(V,A), with two special nodes s and t
  - capacity on each arc
- Find: an optimal flow from s to t



Summary of Lecture 1

Multiple «production» and «demand» points





SECTION 7.2

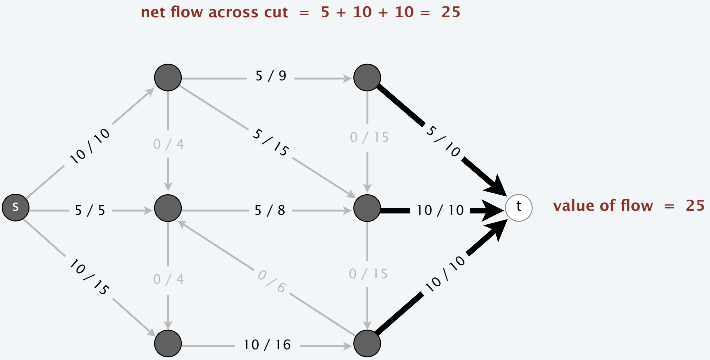
7. NETWORK FLOW I

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *capacity-scaling algorithm*
- *shortest augmenting paths*
- *Dinitz' algorithm*
- *simple unit-capacity networks*

Relationship between flows and cuts

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the value of the flow f equals the net flow across the cut (A, B).

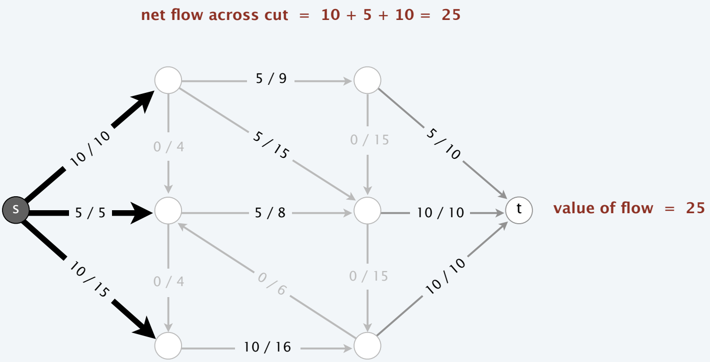
$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$



Relationship between flows and cuts

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the value of the flow f equals the net flow across the cut (A, B).

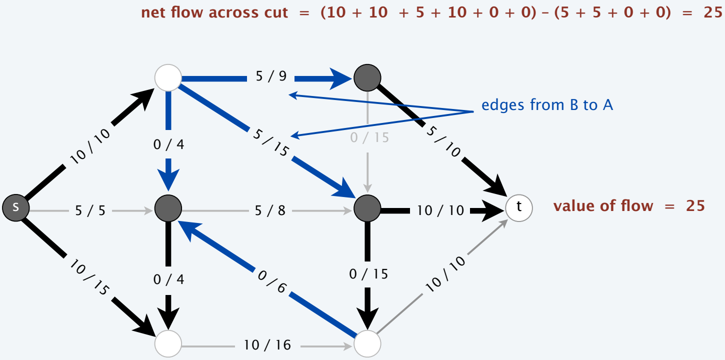
$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$



Relationship between flows and cuts

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the value of the flow f equals the net flow across the cut (A, B).

$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$



Network flow: quiz 3

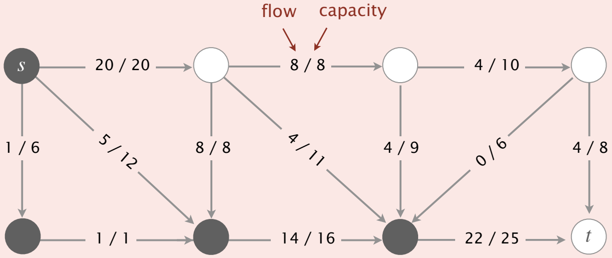
Which is the net flow across the given cut?

A. 11 (20 + 25 – 8 – 11 – 9 – 6)

B. 26 (20 + 22 – 8 – 4 – 4)

C. 42 (20 + 22)

D. 45 (20 + 25)



Relationship between flows and cuts

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the value of the flow f equals the net flow across the cut (A, B).

$$val(f) = \sum_{e \text{ out of } A} f(e) \;-\; \sum_{e \text{ in to } A} f(e)$$

Pf.

$$val(f) = \sum_{e \text{ out of } s} f(e) \;-\; \sum_{e \text{ in to } s} f(e)$$

by flow conservation, all terms except for $v = s$ are 0 $\longrightarrow$

$$= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) \;-\; \sum_{e \text{ in to } v} f(e) \right)$$

$$= \sum_{e \text{ out of } A} f(e) \;-\; \sum_{e \text{ in to } A} f(e) \;\blacksquare$$

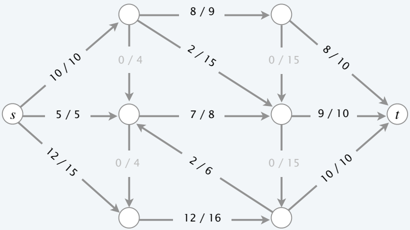
Relationship between flows and cuts

Weak duality. Let f be any flow and (A, B) be any cut. Then, $val(f) \leq cap(A, B)$.

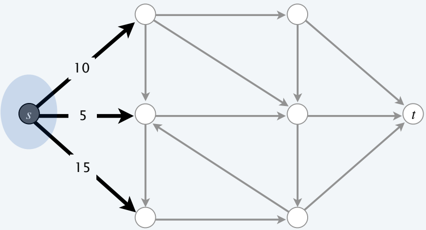
Pf.

$$
\begin{aligned}
val(f) &= \sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to } A} f(e) \\
&\leq \sum_{e \text{ out of } A} f(e) \\
&\leq \sum_{e \text{ out of } A} c(e) \\
&= cap(A, B) \quad \blacksquare
\end{aligned}
$$

flow value lemma



value of flow = 27 $\leq$



capacity of cut = 30

Certificate of optimality

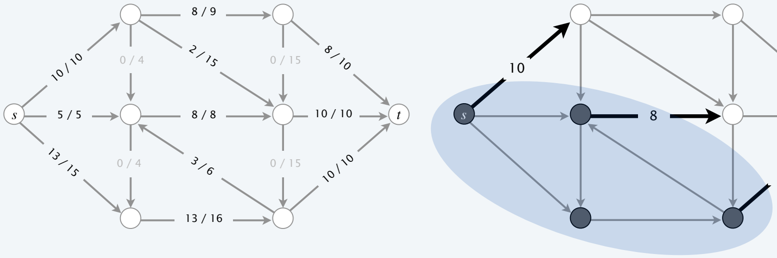
Corollary. Let f be a flow and let (A, B) be any cut.
If $val(f) = cap(A, B)$, then f is a max flow and (A, B) is a min cut.

Pf.

weak duality

- For any flow f': $val(f') \leq cap(A, B) = val(f)$.
- For any cut (A', B'): $cap(A', B') \geq val(f) = cap(A, B)$. ▪

weak duality



value of flow = 28 = capacity of cut = 28

Max-flow min-cut theorem

Max-flow min-cut theorem. Value of a max flow = capacity of a min cut.

strong duality

MAXIMAL FLOW THROUGH A NETWORK

L. R. FORD, JR. AND D. R. FULKERSON

Introduction. The problem discussed in this paper was formulated by T. Harris as follows:

"Consider a rail network connecting two cities by way of a number of intermediate cities, where each link of the network has a number assigned to it representing its capacity. Assuming a steady state condition, find a maximal flow from one given city to the other."

ON THE MAX FLOW MIN CUT THEOREM OF NETWORKS

G. B. Dantzig
D. R. Fulkerson

P-826

April 15, 1955

A Note on the Maximum Flow Through a Network*

P. ELIAS†, A. FEINSTEIN‡, AND C. E. SHANNON§

Summary—This note discusses the problem of maximizing the rate of flow from one terminal to another, through a network which consists of a number of branches, each of which has a limited capacity. The main result is a theorem: The maximum possible flow from left to right through a network is equal to the minimum value among all simple cut-sets. This theorem is applied to solve a more general problem, in which a number of input nodes and a number of output nodes are used.

from one terminal to the other in the original network passes through at least one branch in the cut-set. In the network above, some examples of cut-sets are (d, e, f), and (b, c, e, g, h), (d, g, h, i). By a *simple cut-set* we will mean a cut-set such that if any branch is omitted it is no longer a cut-set. Thus (d, e, f) and (b, c, e, g, h) are simple

Max-flow min-cut theorem

Max-flow min-cut theorem. Value of a max flow = capacity of a min cut.
Augmenting path theorem. A flow f is a max flow iff no augmenting paths.

Pf. The following three conditions are equivalent for any flow f:

i. There exists a cut (A, B) such that $cap(A, B) = val(f)$.
ii. f is a max flow.
iii. There is no augmenting path with respect to f. ← if Ford–Fulkerson terminates, then f is max flow

[i ⇒ ii]

- This is the weak duality corollary. ▪

Max-flow min-cut theorem

Max-flow min-cut theorem. Value of a max flow = capacity of a min cut.

Augmenting path theorem. A flow f is a max flow iff no augmenting paths.

Pf. The following three conditions are equivalent for any flow f:

i. There exists a cut (A, B) such that $cap(A, B) = val(f)$.

ii. f is a max flow.

iii. There is no augmenting path with respect to f.

[ii $\Rightarrow$ iii] We prove contrapositive: $\neg$ iii $\Rightarrow$ $\neg$ ii.

- Suppose that there is an augmenting path with respect to f.
- Can improve flow f by sending flow along this path.
- Thus, f is not a max flow. ▪

Max-flow min-cut theorem

[iii ⇒ i]

- Let f be a flow with no augmenting paths.
- Let A be set of nodes reachable from s in residual network G_f.
- By definition of A: $s \in A$.
- By definition of flow f: $t \notin A$.

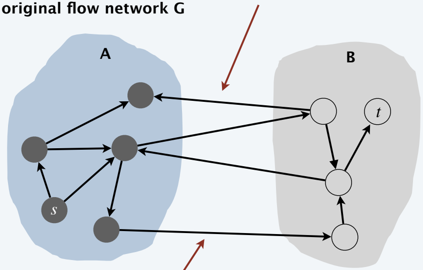
$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

flow value lemma

$$= \sum_{e \text{ out of } A} c(e) - 0$$

$$= cap(A, B) \quad \blacksquare$$

edge $e = (v, w)$ with $v \in B, w \in A$
must have $f(e) = 0$



edge $e = (v, w)$ with $v \in A, w \in B$
must have $f(e) = c(e)$

Airline Rostering

Recap example: consider the following set of flights

- Boston (6 AM) Washington DC (7AM)
- Urbana (7 AM) Champaign (8 AM)
- Washington (8 AM) Los Angeles (11 AM)
- Urbana (11 AM) San Francisco (2 PM)
- San Francisco (2:15 PM) Seattle (3:15 PM)
- Las Vegas (5 PM) Seattle (6 PM)

How many crews do you need to operate all of them?

Preemptive Scheduling

Recap example: consider the following combinatorial problem. Given

- a set of jobs J, each having a release date r_j, a processing time p_j and a due date d_j
- a set of M identical machines

decide if a scheduling exists, such that

- each job is completed within C
- no jobs overlap on the same machine
- no job j is started before (resp. completed after) its release date r_j (resp. due date d_j)

Preemption is allowed.

Preemptive Scheduling

Recap example: instance

Job	1	2	3	4
Proc. time	1.50	1.25	2.10	3.60
Release date	3	1	3	5
Due date	5	4	7	9

Max Flow - Min Cut duality

Recap: max flow and min cut form a pair of *strongly dual* problems

Max-flow min-cut theorem

Max-flow min-cut theorem. Value of a max flow = capacity of a min cut.

Augmenting path theorem. A flow f is a max flow iff no augmenting paths.

Pf. The following three conditions are equivalent for any flow f:

i. There exists a cut (A, B) such that $cap(A, B) = val(f)$.

ii. f is a max flow.

iii. There is no augmenting path with respect to f.

[ii ⇒ iii] We prove contrapositive: ¬ iii ⇒ ¬ ii.

- Suppose that there is an augmenting path with respect to f.
- Can improve flow f by sending flow along this path.
- Thus, f is not a max flow. ▪

Max-flow min-cut theorem

[iii ⇒ i]

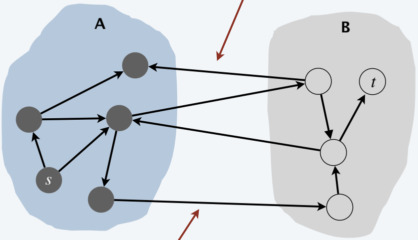
- Let f be a flow with no augmenting paths.
- Let A be set of nodes reachable from s in residual network G_f.
- By definition of A: $s \in A$.
- By definition of flow f: $t \notin A$.

$$
\begin{aligned}
val(f) &= \sum_{e \text{ out of } A} f(e) \;-\; \sum_{e \text{ in to } A} f(e) \\
&= \sum_{e \text{ out of } A} c(e) \;-\; 0 \\
&= cap(A, B) \quad \blacksquare
\end{aligned}
$$

flow value lemma

edge $e = (v, w)$ with $v \in B$, $w \in A$ must have $f(e) = 0$

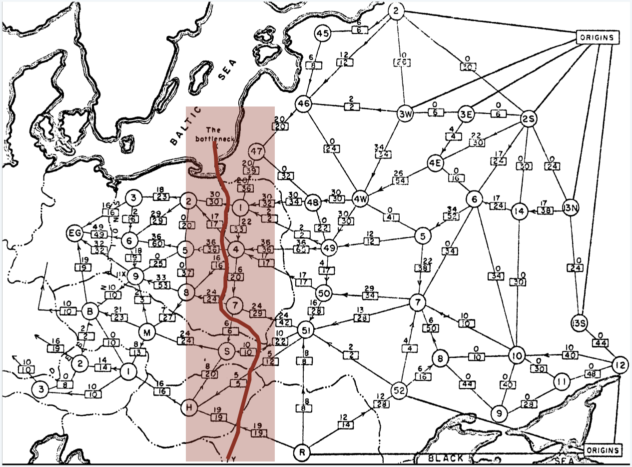
original flow network G



edge $e = (v, w)$ with $v \in A$, $w \in B$ must have $f(e) = c(e)$

Minimum cut application (RAND 1950s)

"Free world" goal. Cut supplies (if Cold War turns into real war).



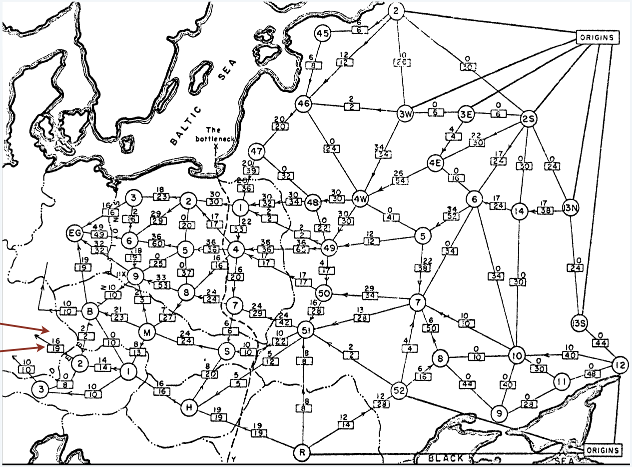
rail network connecting Soviet Union with Eastern European countries
(map declassified by Pentagon in 1999)

Maximum flow application (Tolstoĭ 1930s)

Soviet Union goal. Maximize flow of supplies to Eastern Europe.

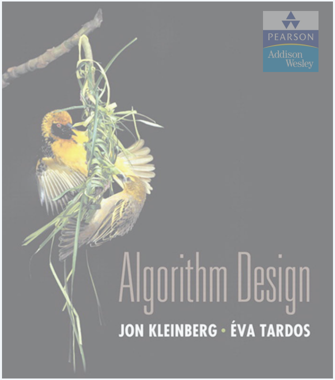
flow

capacity



rail network connecting Soviet Union with Eastern European countries

(map declassified by Pentagon in 1999)



SECTION 7.10

7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Image segmentation

Image segmentation.

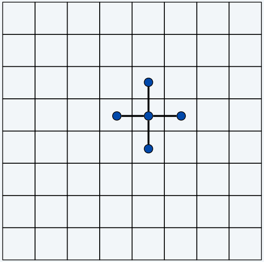
- Divide image into coherent regions.
- Central problem in image processing.

Ex. Separate human and robot from background scene.

Image segmentation

Foreground / background segmentation.

- Label each pixel in picture as belonging to foreground or background.
- V = set of pixels, E = pairs of neighboring pixels.
- $a_i \geq 0$ is likelihood pixel i in foreground.
- $b_i \geq 0$ is likelihood pixel i in background.
- $p_{ij} \geq 0$ is separation penalty for labeling one of i and j as foreground, and the other as background.



Goals.

- Accuracy: if $a_i > b_i$ in isolation, prefer to label i in foreground.
- Smoothness: if many neighbors of i are labeled foreground, we should be inclined to label i as foreground.
- Find partition (A, B) that maximizes: $\displaystyle \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$

foreground background

Image segmentation

Formulate as min-cut problem.

- Maximization.
- No source or sink.
- Undirected graph.

Turn into minimization problem.

- Maximizing $$\sum_{i \in A} a_i \;+\; \sum_{j \in B} b_j \;-\; \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

- is equivalent to minimizing

$$\underbrace{\left(\sum_{i \in V} a_i \;+\; \sum_{j \in V} b_j \right)}_{\text{a constant}} \;-\; \sum_{i \in A} a_i \;-\; \sum_{j \in B} b_j \;+\; \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

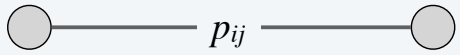
- or alternatively $$\sum_{j \in B} a_j \;+\; \sum_{i \in A} b_i \;+\; \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

Image segmentation

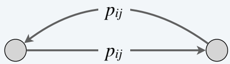
Formulate as min-cut problem $G' = (V', E')$.

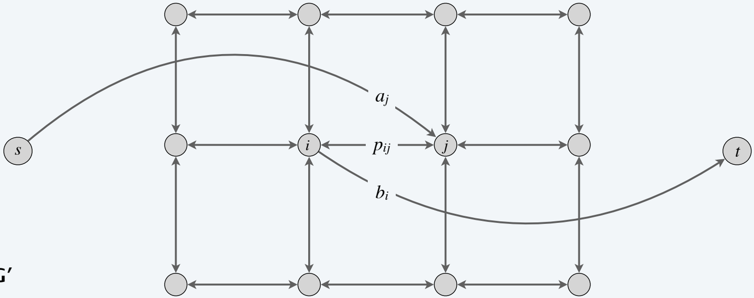
- Include node for each pixel.
- Use two antiparallel edges instead of undirected edge.
- Add source s to correspond to foreground.
- Add sink t to correspond to background.

edge in G



two antiparallel edges in G′





G′

Image segmentation

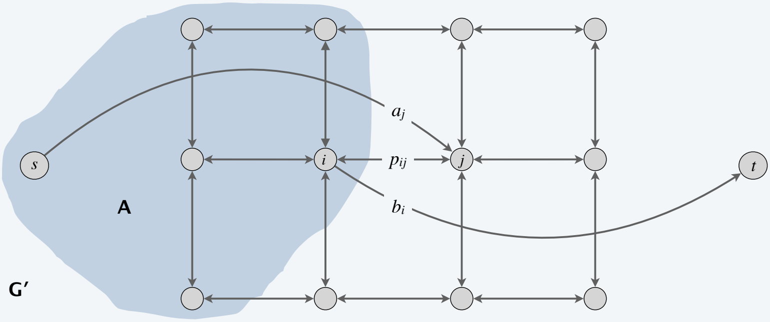
Consider min cut (A, B) in G'.

- A = foreground.

$$cap(A,B) \;=\; \sum_{j \in B} a_j \;+\; \sum_{i \in A} b_i \;+\; \sum_{\substack{(i,j) \in E \\ i \in A,\; j \in B}} p_{ij}$$

if i and j on different sides, p_{ij} counted exactly once

- Precisely the quantity we want to minimize.



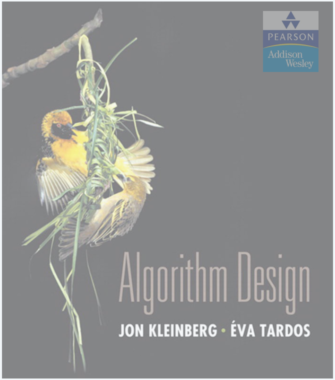
Grabcut image segmentation

Grabcut. [Rother–Kolmogorov–Blake 2004]

"GrabCut" — Interactive Foreground Extraction using Iterated Graph Cuts

Carsten Rother* Vladimir Kolmogorov† Andrew Blake‡
Microsoft Research Cambridge, UK

Figure 1: **Three examples of GrabCut .** The user drags a rectangle loosely around an object. The object is then extracted automatically.



SECTION 7.11

7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Project selection (maximum weight closure problem)

Projects with prerequisites.

can be positive or negative

- Set of possible projects P: project v has associated revenue p_v.
- Set of prerequisites E: $(v, w) \in E$ means w is a prerequisite for v.
- A subset of projects $A \subseteq P$ is feasible if the prerequisite of every project in A also belongs to A.

Project selection problem. Given a set of projects P and prerequisites E, choose a feasible subset of projects to maximize revenue.

MANAGEMENT SCIENCE
Vol. 22, No. 11, July, 1976
Printed in U.S.A.

MAXIMAL CLOSURE OF A GRAPH AND APPLICATIONS TO COMBINATORIAL PROBLEMS*†

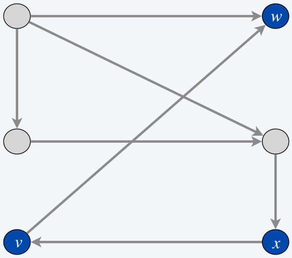
JEAN-CLAUDE PICARD

Ecole Polytechnique, Montreal

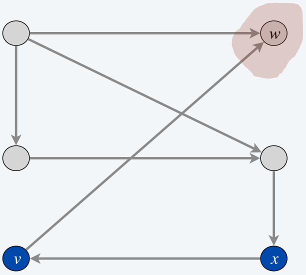
This paper generalizes the selection problem discussed by J. M. Rhys [12], J. D. Murchland [9], M. L. Balinski [1] and P. Hansen [4]. Given a directed graph G, a closure of G is defined as a subset of nodes such that if a node belongs to the closure all its successors also belong to the set. If a real number is associated to each node of G a maximal closure is defined as a closure of maximal value.

Project selection: prerequisite graph

Prerequisite graph. Add edge (v, w) if w is a prerequisite for v.



{ v, w, x } is feasible

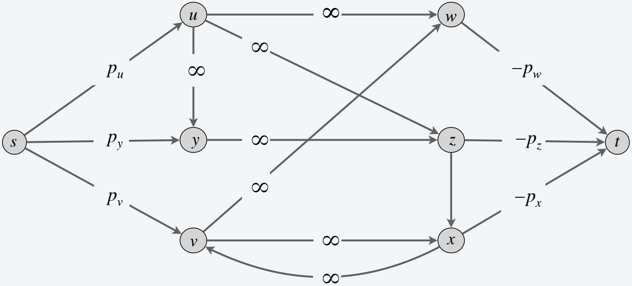


{ v, x } is infeasible

Project selection: min-cut formulation

Min-cut formulation.

- Assign a capacity of ∞ to each prerequisite edge.
- Add edge (s, v) with capacity p_v if $p_v > 0$.
- Add edge (v, t) with capacity $-p_v$ if $p_v < 0$.
- For notational convenience, define $p_s = p_t = 0$.



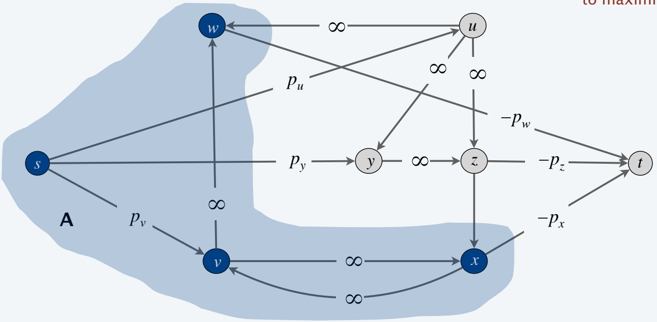
Project selection: min-cut formulation

Claim. (A, B) is min cut iff $A - \{ s \}$ is an optimal set of projects.

- Infinite capacity edges ensure $A - \{ s \}$ is feasible.
- Max revenue because: $$cap(A, B) = \sum_{v \in B:\ p_v > 0} p_v + \sum_{v \in A:\ p_v < 0} (-p_v)$$ $$= \sum_{v:\ p_v > 0} p_v - \sum_{v \in A} p_v$$

a constant

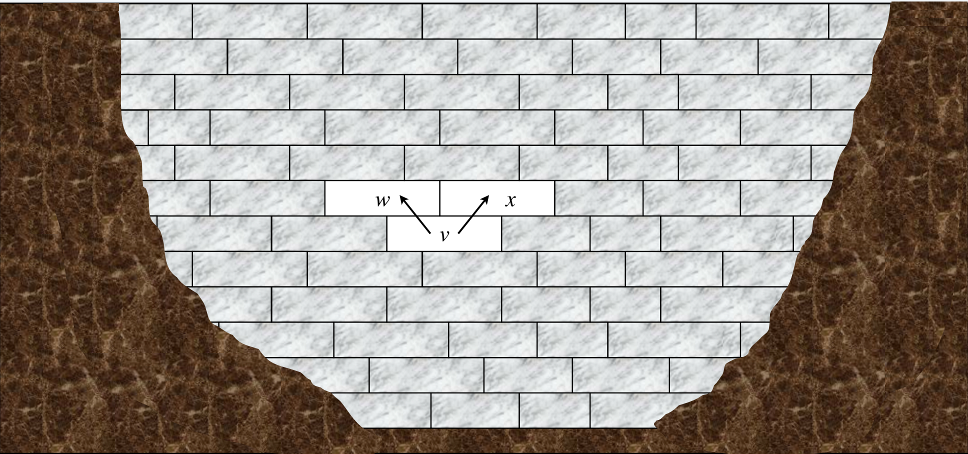
minimizing this is equivalent to maximizing revenue



Open-pit mining

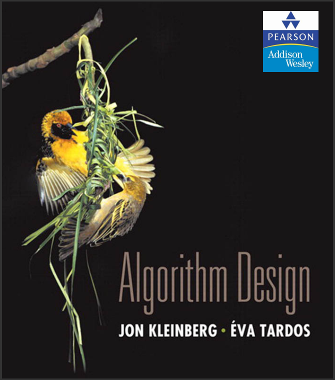
Open-pit mining. [studied since early 1960s]

- Blocks of earth are extracted from surface to retrieve ore.
- Each block v has net value p_v = value of ore − processing cost.
- Can't remove block v until both blocks w and x are removed.



Integrality Theorem

Claim: if all arc capacities are integer, an integral maximum flow always exists.
Proof: consider Ford Fulkerson and proceed by induction (blackboard).



Lecture slides by Kevin Wayne

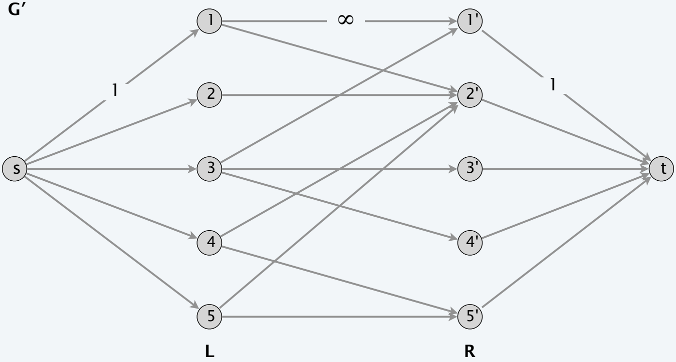
http://www.cs.princeton.edu/~wayne/kleinberg-tardos

7. Network Flow II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Bipartite matching: max-flow formulation

- Create digraph $G' = (L \cup R \cup \{s, t\},\ E')$.
- Direct all edges from L to R, and assign infinite (or unit) capacity.
- Add unit-capacity edges from s to each node in L.
- Add unit-capacity edges from each node in R to t.

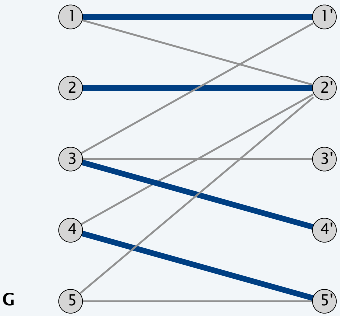


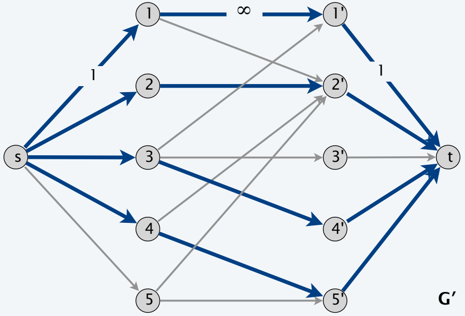
Max-flow formulation: proof of correctness

Theorem. Max cardinality of a matching in G = value of max flow in G'.

Pf. ≤

- Given a max matching M of cardinality k.
- Consider flow f that sends 1 unit on each of the k corresponding paths.
- f is a flow of value k. ▪



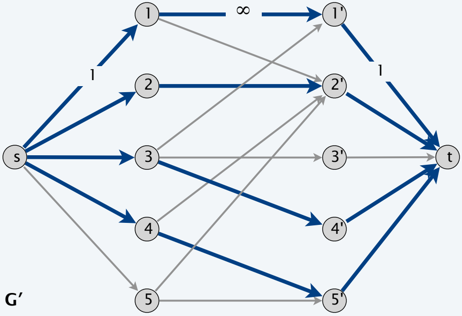


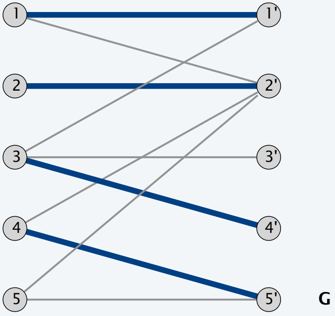
Max-flow formulation: proof of correctness

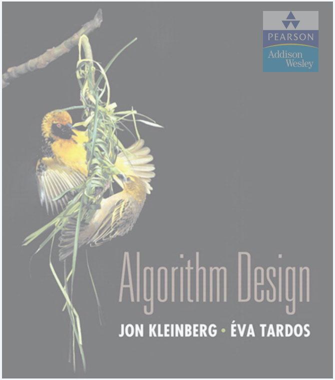
Theorem. Max cardinality of a matching in G = value of max flow in G'.

Pf. $\geq$

- Let f be a max flow in G' and let k denote its value.
- Integrality theorem $\Rightarrow$ k is integral and can assume f is 0–1.
- Consider M = set of edges from L to R with $f(e) = 1$.
  - each node in L and R participates in at most one edge in M
  - $|M| = k$: apply flow-value lemma to cut $(L \cup \{s\}, R \cup \{t\})$ ▪







SECTION 7.6

7. NETWORK FLOW II

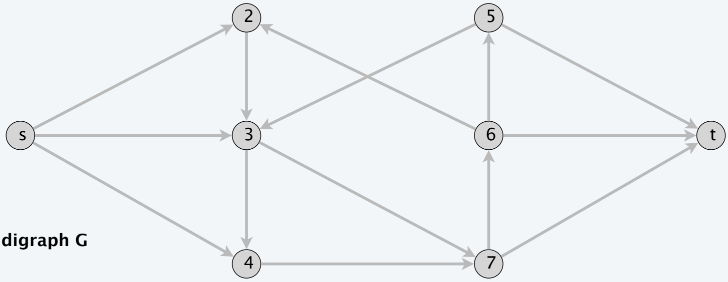
- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Edge-disjoint paths

Def. Two paths are edge-disjoint if they have no edge in common.

Edge-disjoint paths problem. Given a digraph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint $s \rightarrow t$ paths.

Ex. Communication networks.

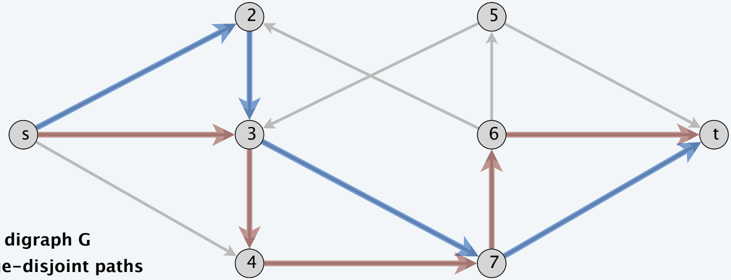


Edge-disjoint paths

Def. Two paths are edge-disjoint if they have no edge in common.

Edge-disjoint paths problem. Given a digraph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint $s \rightarrow t$ paths.

Ex. Communication networks.



digraph G
2 edge-disjoint paths

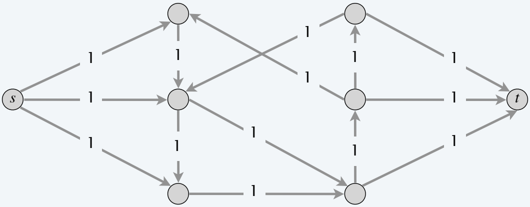
Edge-disjoint paths

Max-flow formulation. Assign unit capacity to every edge.

Theorem. Max number of edge-disjoint $s \rightsquigarrow t$ paths = value of max flow.

Pf. $\le$

- Suppose there are k edge-disjoint $s \rightsquigarrow t$ paths $P_1, \ldots, P_k$.
- Set $f(e) = 1$ if e participates in some path P_j ; else set $f(e) = 0$.
- Since paths are edge-disjoint, f is a flow of value k. ▪



Edge-disjoint paths

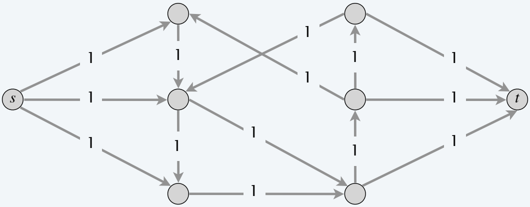
Max-flow formulation. Assign unit capacity to every edge.

Theorem. Max number of edge-disjoint $s \rightsquigarrow t$ paths = value of max flow.

Pf. $\geq$

- Suppose max flow value is k.
- Integrality theorem $\Rightarrow$ there exists $0\text{–}1$ flow f of value k.
- Consider edge (s, u) with $f(s, u) = 1$.
  - by flow conservation, there exists an edge (u, v) with $f(u, v) = 1$
  - continue until reach t, always choosing a new edge
- Produces k (not necessarily simple) edge-disjoint paths. ▪

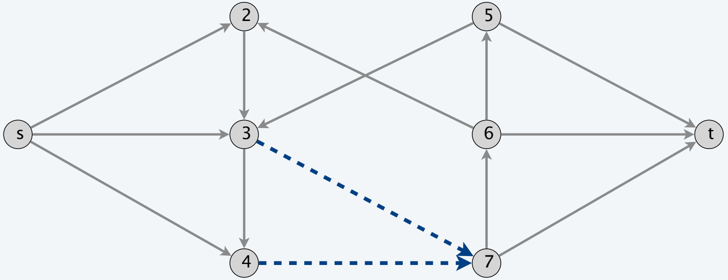
can eliminate cycles to get simple paths in $O(mn)$ time if desired (flow decomposition)



Network connectivity

Def. A set of edges $F \subseteq E$ disconnects t from s if every $s \to t$ path uses at least one edge in F.

Network connectivity. Given a digraph $G = (V, E)$ and two nodes s and t, find min number of edges whose removal disconnects t from s.

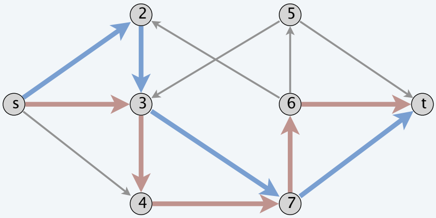


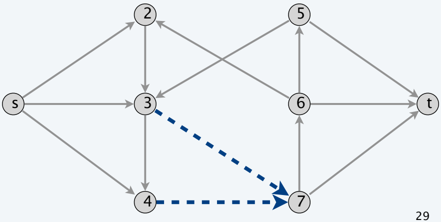
Menger's theorem

Theorem. [Menger 1927] The max number of edge-disjoint $s \rightarrow t$ paths equals the min number of edges whose removal disconnects t from s.

Pf. $\leq$

- Suppose the removal of $F \subseteq E$ disconnects t from s, and $|F| = k$.
- Every $s \rightarrow t$ path uses at least one edge in F.
- Hence, the number of edge-disjoint paths is $\leq$ k. ▪



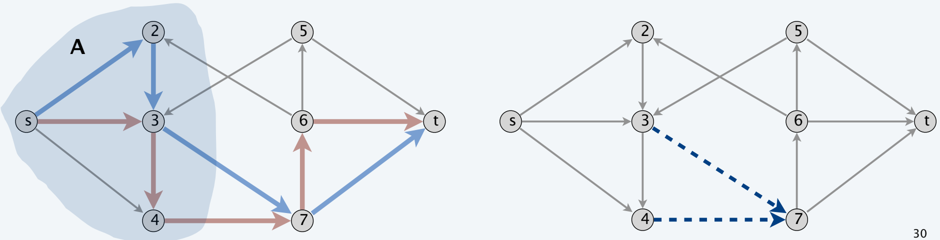


Menger's theorem

Theorem. [Menger 1927] The max number of edge-disjoint $s \to t$ paths equals the min number of edges whose removal disconnects t from s.

Pf. $\geq$

- Suppose max number of edge-disjoint paths is k.
- Then value of max flow $= k$.
- Max-flow min-cut theorem $\Rightarrow$ there exists a cut (A, B) of capacity k.
- Let F be set of edges going from A to B.
- $|F| = k$ and disconnects t from s. ▪

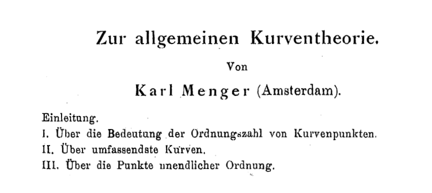


More Menger theorems

Theorem. Given an undirected graph and two nodes s and t, the max number of edge-disjoint s–t paths equals the min number of edges whose removal disconnects s and t.

Theorem. Given an undirected graph and two nonadjacent nodes s and t, the max number of internally node-disjoint s–t paths equals the min number of internal nodes whose removal disconnects s and t.

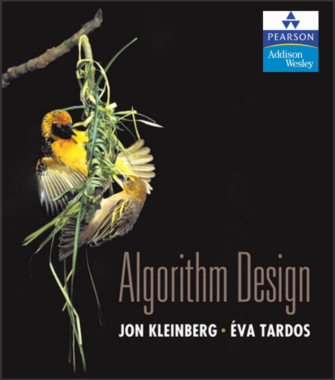
Theorem. Given a directed graph with two nonadjacent nodes s and t, the max number of internally node-disjoint $s \to t$ paths equals the min number of internal nodes whose removal disconnects t from s.


Zur allgemeinen Kurventheorie.

Von

Karl Menger (Amsterdam).

Einleitung.
I. Über die Bedeutung der Ordnungszahl von Kurvenpunkten.
II. Über umfassendste Kurven.
III. Über die Punkte unendlicher Ordnung.



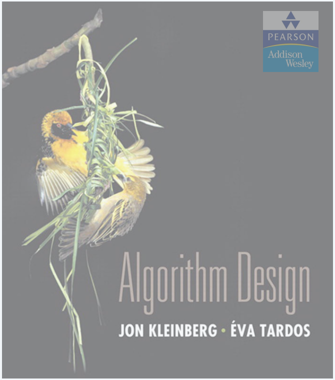
7. Network Flow I

- *Ford–Fulkerson demo*
- *exponential-time example*
- *pathological example*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on 4/17/18 9:07 AM



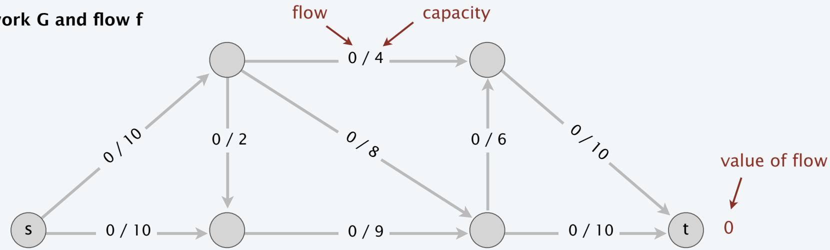
SECTION 7.1

7. NETWORK FLOW I

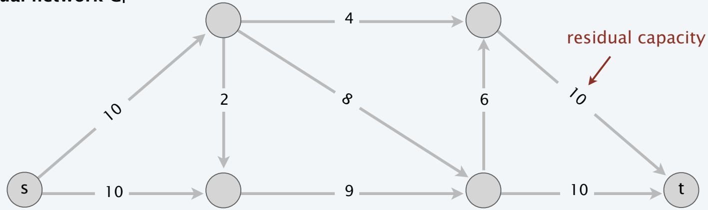
- *Ford–Fulkerson demo*
- *exponential-time example*
- *pathological example*

Ford–Fulkerson algorithm demo

network G and flow f

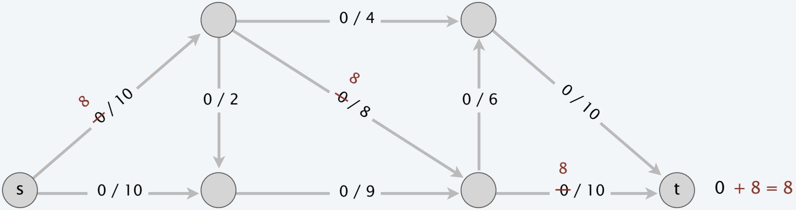


residual network G_f

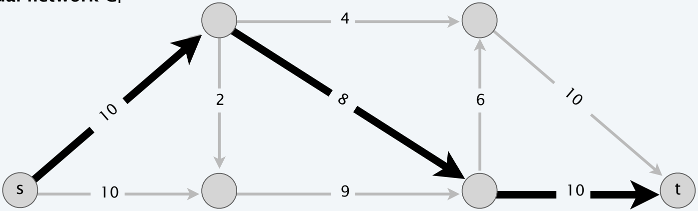


Ford–Fulkerson algorithm demo

network G and flow f

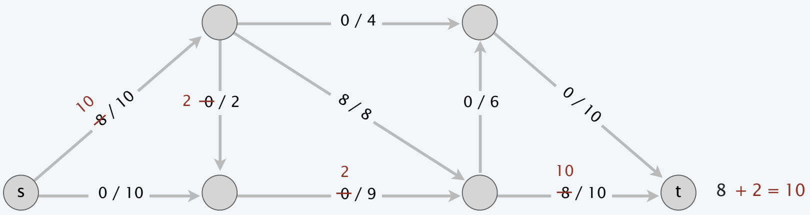


residual network G_f

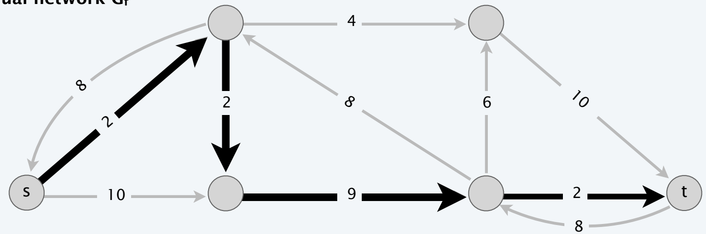


Ford–Fulkerson algorithm demo

network G and flow f

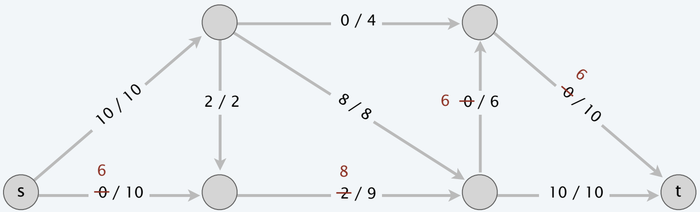


residual network G_f



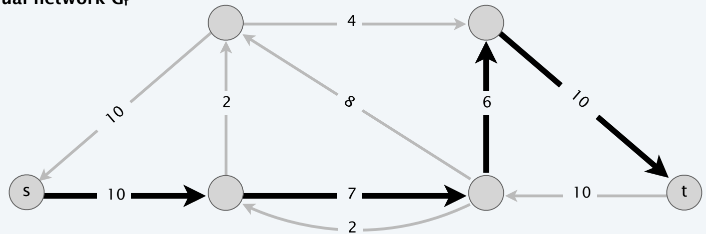
Ford–Fulkerson algorithm demo

network G and flow f



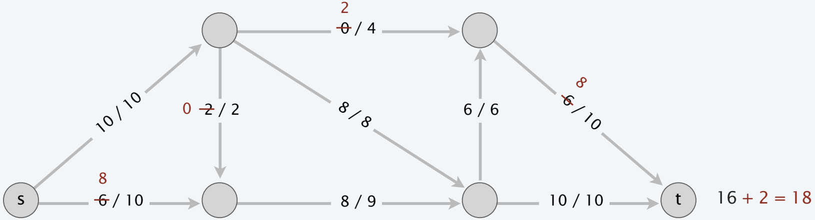
10 + 6 = 16

residual network G_f



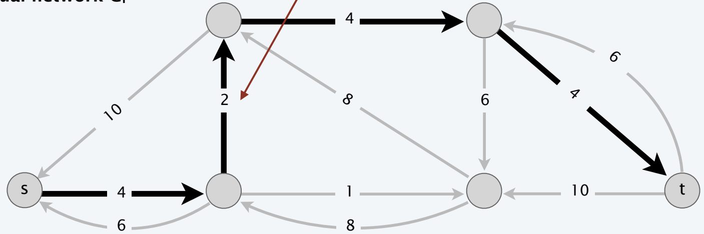
Ford–Fulkerson algorithm demo

network G and flow f



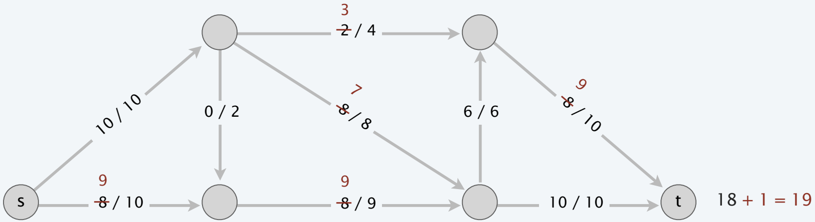
fixes mistake from second augmenting path

residual network G_f

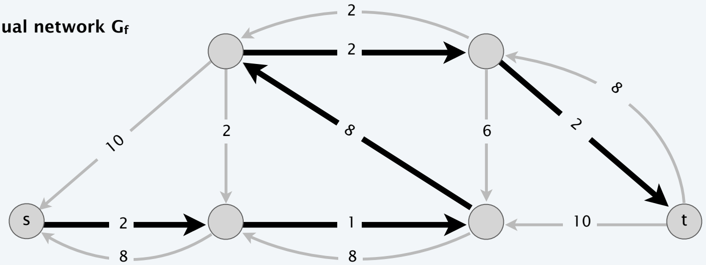


Ford–Fulkerson algorithm demo

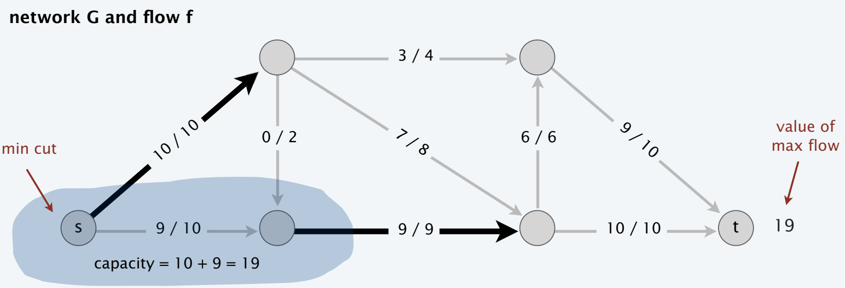
network G and flow f

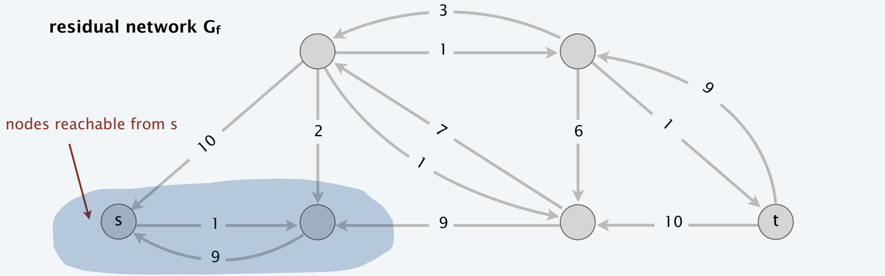


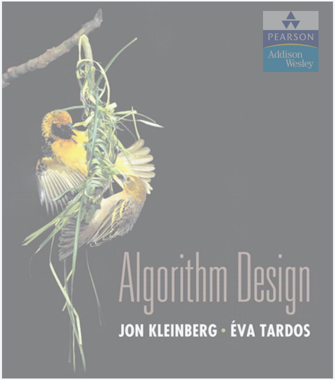
residual network G_f



Ford–Fulkerson algorithm demo







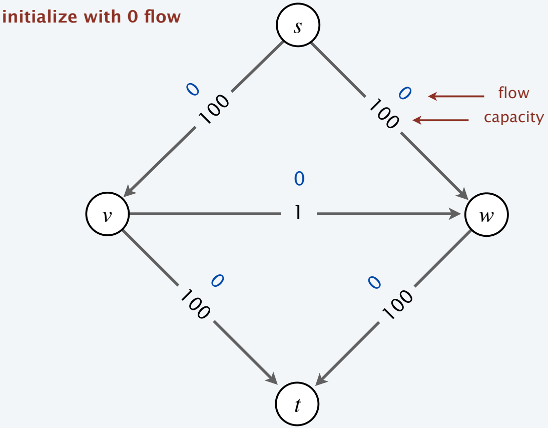
SECTION 7.1

7. NETWORK FLOW I

- Ford–Fulkerson demo
- *exponential-time example*
- pathological example

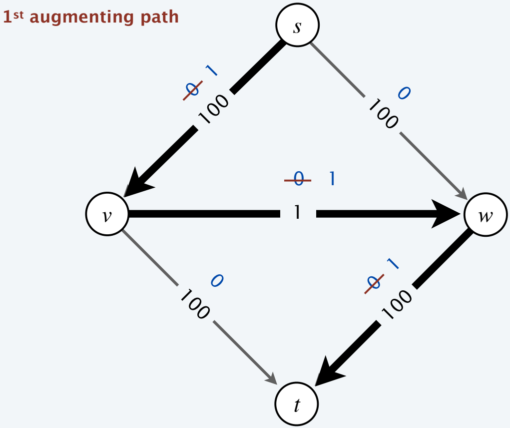
Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.



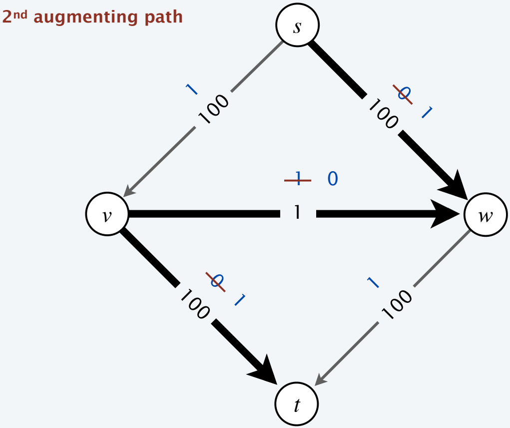
Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.



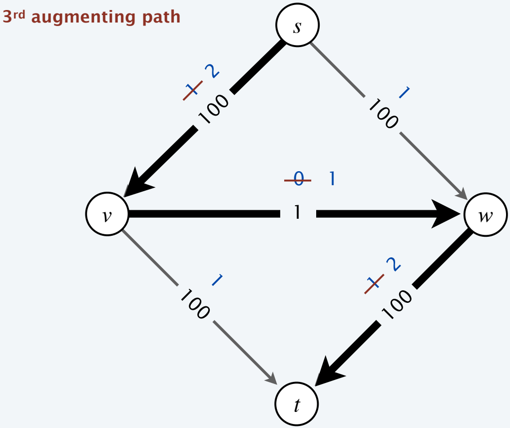
Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.



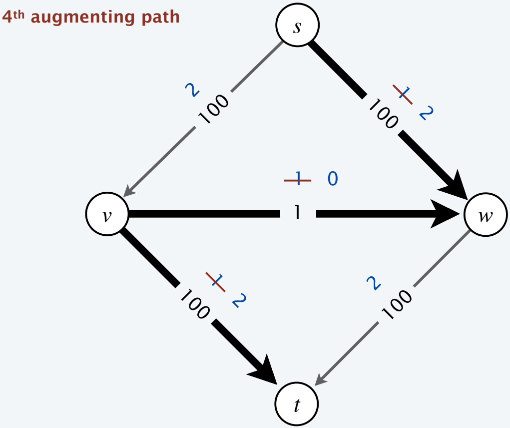
Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.



Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.

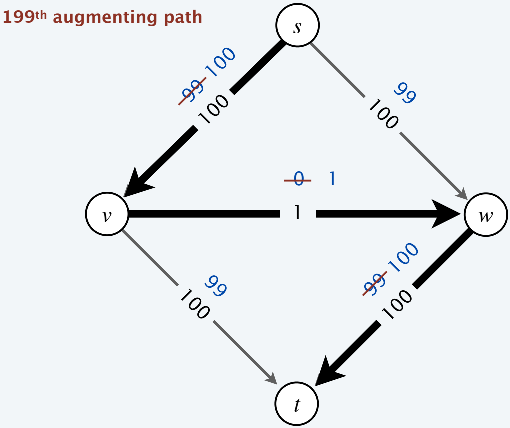


Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.

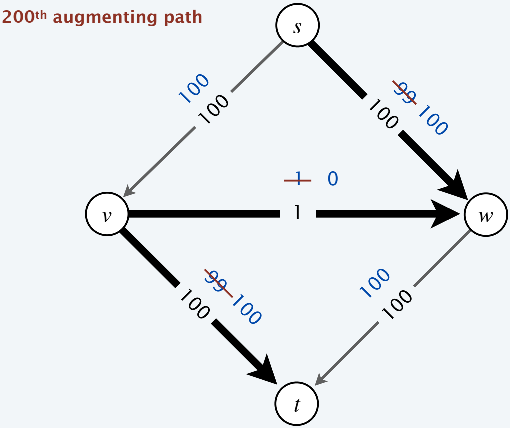
Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.



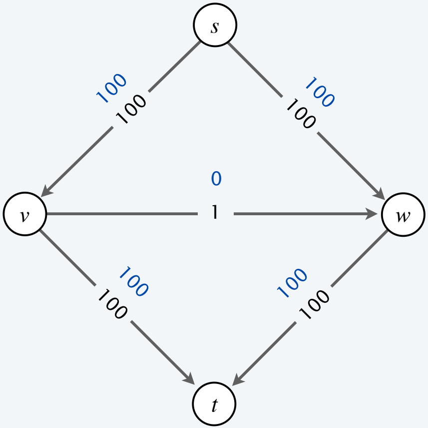
Ford–Fulkerson algorithm: exponential-time example

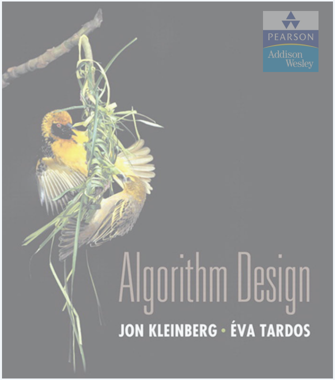
Bad news. Number of augmenting paths can be exponential in input size.



Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.





Section 7.1

7. Network Flow I

- *Ford–Fulkerson demo*
- *exponential-time example*
- *pathological example*

Ford–Fulkerson algorithm: pathological example

Intuition. Let $r > 0$ satisfy $r^2 = 1 - r$.

- Initially, some residual capacities are 1 and r.
- After two augmenting paths, some residual capacities are r and r^2.
- After two more augmenting paths, some residual capacities are r^2 and r^3.
- After two more, some residual capacities are r^3 and r^4.
- By carefully choreographing the augmenting paths, infinitely many residual capacities arise!

$1 - r$

$r - r^2$

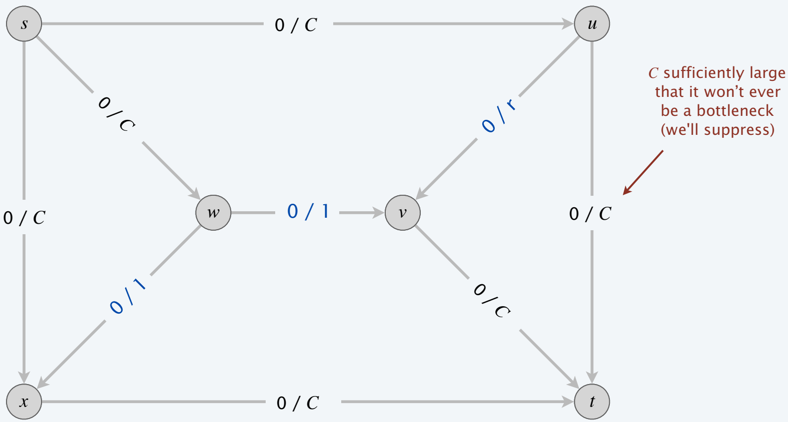
$r^2 - r^3$

$$r = \frac{\sqrt{5} - 1}{2} \implies r^2 = 1 - r$$

$$r \approx 0.618 \implies r^4 < r^3 < r^2 < r < 1$$

Ford–Fulkerson algorithm: pathological example

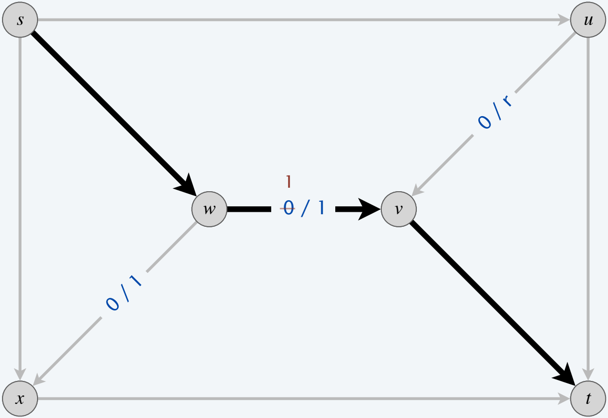
flow network G



$$r^2 = 1 - r$$

Ford–Fulkerson algorithm: pathological example

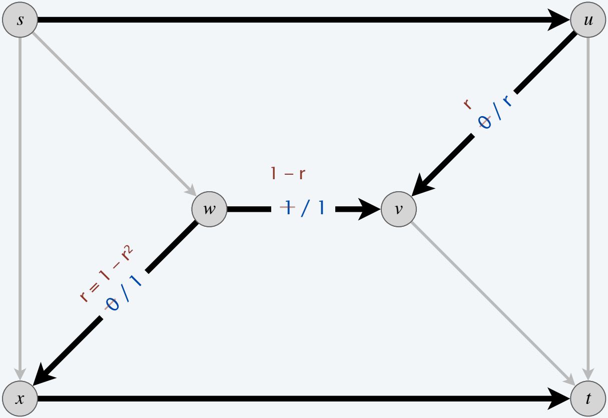
augmenting path 1: s→w→v→t (bottleneck capacity = 1)



$$r^2 = 1 - r$$

Ford–Fulkerson algorithm: pathological example

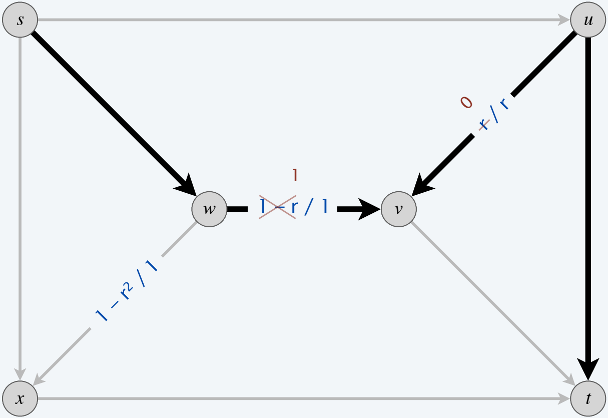
augmenting path 2: s→u→v→w→x→t (bottleneck capacity = r)



$$r^2 = 1 - r$$

Ford–Fulkerson algorithm: pathological example

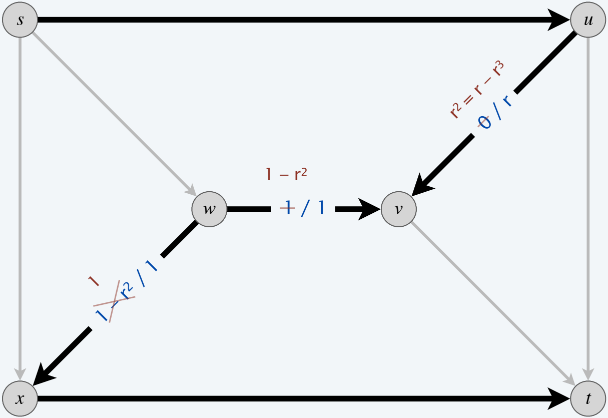
augmenting path 3: s→w→v→u→t (bottleneck capacity = r)



$$r^2 = 1 - r$$

Ford–Fulkerson algorithm: pathological example

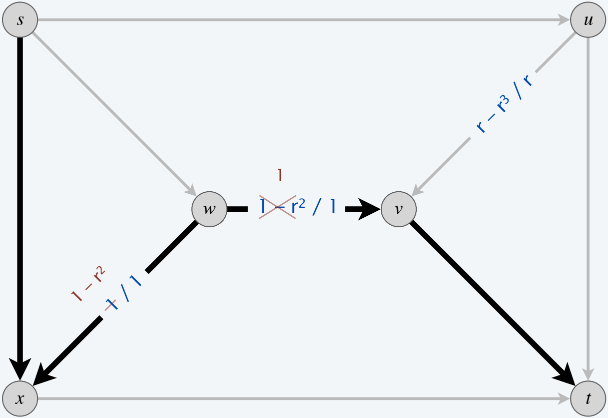
augmenting path 4: s→u→v→w→x→t (bottleneck capacity = r^2)



$$r^2 = 1 - r$$

Ford–Fulkerson algorithm: pathological example

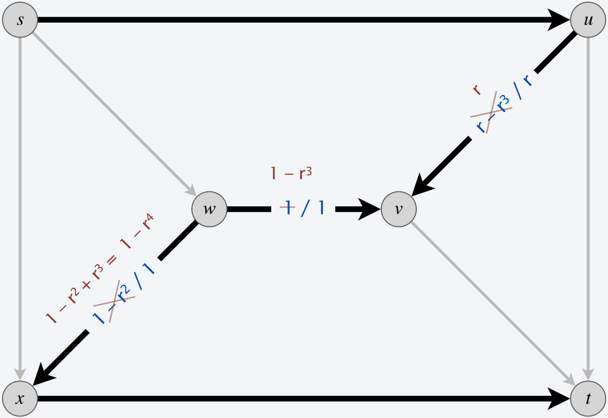
augmenting path 5: s→x→w→v→t (bottleneck capacity = r^2)



$$r^2 = 1 - r$$

Ford–Fulkerson algorithm: pathological example

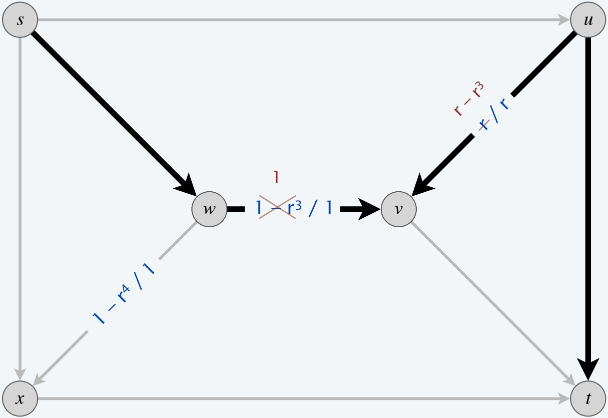
augmenting path 6: s→u→v→w→x→t (bottleneck capacity = r^3)



$$r^2 = 1 - r$$

Ford–Fulkerson algorithm: pathological example

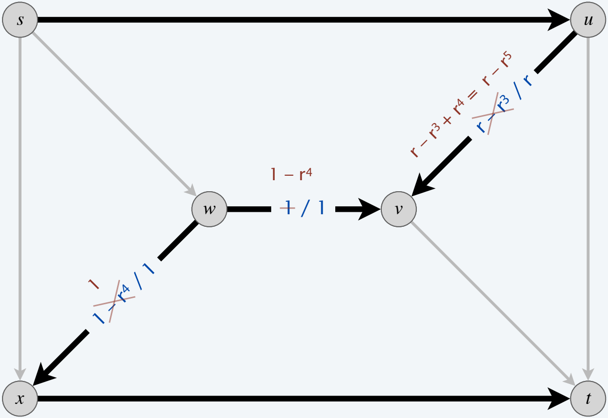
augmenting path 7: s→w→v→u→t (bottleneck capacity = r^3)



$$r^2 = 1 - r$$

Ford–Fulkerson algorithm: pathological example

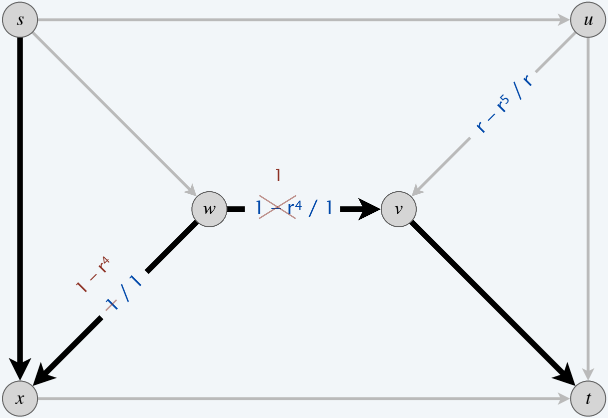
augmenting path 8: s→u→v→w→x→t (bottleneck capacity = r^4)



$$r^2 = 1 - r$$

Ford–Fulkerson algorithm: pathological example

augmenting path 9: s→x→w→v→t (bottleneck capacity = r^4)



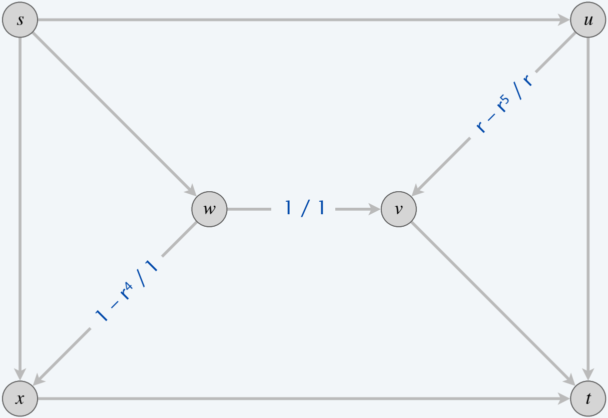
$$r^2 = 1 - r$$

Ford–Fulkerson algorithm: pathological example

flow after augmenting path 1: { r – r^1, 1, 1 – r^0 } (value of flow = 1)

flow after augmenting path 5: { r – r^3, 1, 1 – r^2 } (value of flow = 1 + 2r + $2r^2$)

flow after augmenting path 9: { r – r^5, 1, 1 – r^4 } (value of flow = 1 + 2r + $2r^2$ + $2r^3$ + $2r^4$)



$$r^2 = 1 - r$$

Ford–Fulkerson algorithm: pathological example

Theorem. The Ford–Fulkerson algorithm may not terminate; moreover, it may converge to a value not equal to the value of the maximum flow.

Pf.

- After $(1 + 4k)$ augmenting paths of the form just described, the value of the flow

$$\begin{aligned} &= 1 + 2\sum_{i=1}^{2k} r^i \\ &\leq 1 + 2\sum_{i=1}^{\infty} r^i \\ &= 1 + \frac{2r}{1-r} \\ &< 5 \end{aligned}$$

$$\boxed{r = \frac{\sqrt{5}-1}{2}}$$

- Value of maximum flow $= 2C + 1$. ▪

Reference

ELSEVIER Theoretical Computer Science 148 (1995) 165–170

Theoretical Computer Science

Note

The smallest networks on which the Ford–Fulkerson maximum flow procedure may fail to terminate

Uri Zwick *

Department of Computer Science, Tel Aviv University, Ramat Aviv, 69978 Tel Aviv, Israel

Received November 1993
Communicated by M. Nivat

Abstract

It is widely known that the Ford–Fulkerson procedure for finding the maximum flow in a network need not terminate if some of the capacities of the network are irrational. Ford and Fulkerson gave as an example a network with 10 vertices and 48 edges on which their procedure may fail to halt. We construct much smaller and simpler networks on which the same may happen. Our smallest network has only 6 vertices and 8 edges. We show that it is the smallest example possible.

Review of the Ford-Fulkerson Algorithm

```
Begin
    x := 0;
    create the residual network G(x);
    while there is some directed path from s to t in
      G(x) do
    begin
        let P be a path from s to t in G(x);
        δ* := δ(P);
        send δ* units of flow along P;
        update the r's;
    end
end {the flow x is now maximum}.
```

Improved Algorithms

The largest augmenting path algorithm:

- **Let P be a path from s to t in G(x) such that δ*` is maximum.**

The shortest augmenting path algorithm:

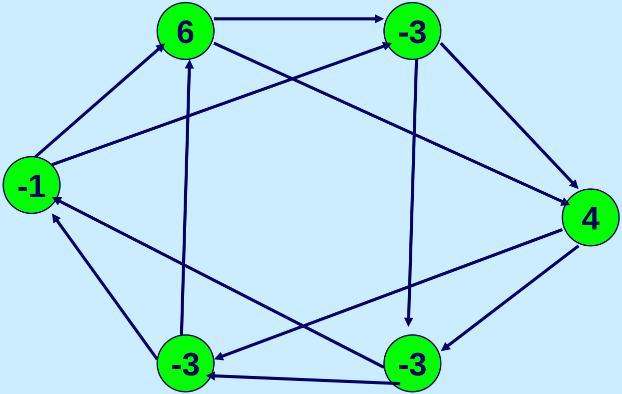
- **Let P be a path from s to t in G(x) with the fewest number of arcs.**

The Capacity Scaling Algorithm

- **For any fixed value Δ, let** G(x,Δ) **be the arcs in G(x) with capacity at least Δ.**
- **A flow x is called** *Δ-maximum* **if there is no augmenting path of size Δ or more.**
- **Subroutine** ImproveApprox(x,Δ)**. It takes a flow that is Δ-maximum and outputs a flow that is Δ/2-maximum.**

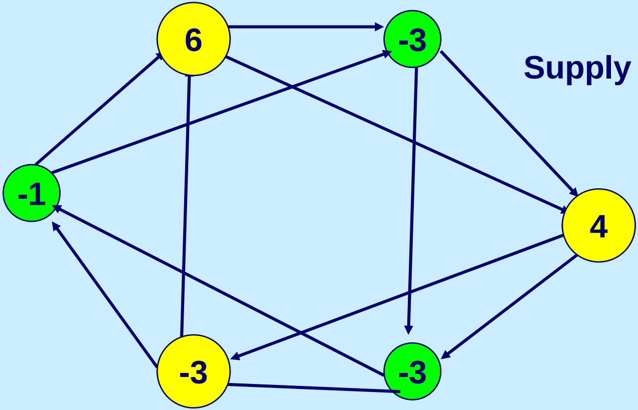
ImproveApprox(x,Δ)
begin
    $\Delta := \Delta/2$;
    while **there is a path from s to t in G(x,Δ)** do
    begin
        find a path P from s to t in G(x,Δ);
        augment flow along P;
        update data structures;
    end
end

When is there a feasible flow?



Suppose all arcs have a capacity of 2, and that node numbers are supplies/demands. Then there is no feasible flow.

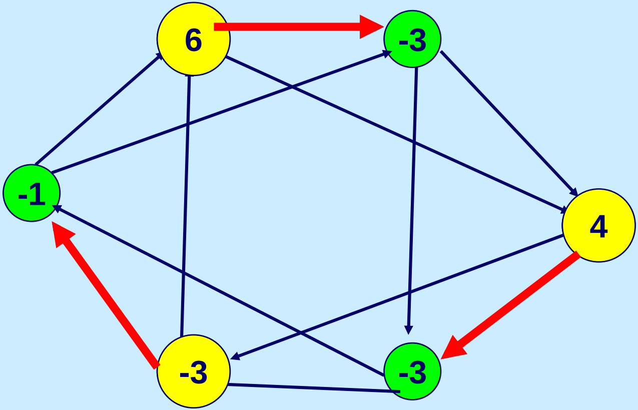
Infeasibility Theorem



Supply of S is 7

***Infeasibility Theorem.* Either there is a feasible flow, or there is a cut (S, T) such that:**
the capacity of (S, T) < supply of S

Infeasibility Theorem

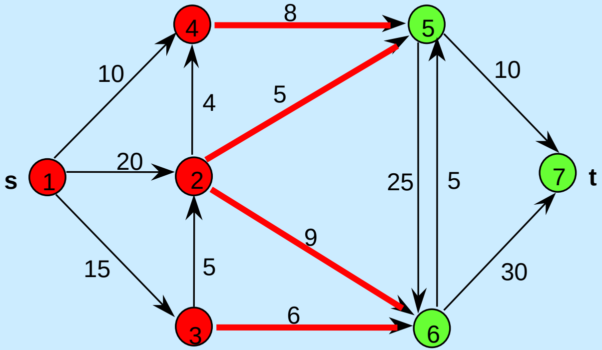


Supply of S is 7.

The capacity of (S, N\S) = 6.

More on Capacity Scaling

This is the residual network at the end of the scaling phase when $\Delta = 10$.



How many augmentations can there be from s to t when Δ is reduced from 10 to 5?

Analysis of Capacity Scaling

There are O(log U) scaling phases.

- **Initially Δ is at most 2U.**
- **Δ is halved at each scaling phase**
- **We can stop when Δ is 1.**

The running time per scaling phase is $O(m^2)$.

- **Each scaling phase has O(m) augmentations.**
- **The time per augmentation is O(m).**

The total running time is $O(m^2 \log U)$

The Shortest Augmenting Path Algorithm

Overview:

- **We will establish the following:**
  - **We can determine each augmentation in O(n) time if we maintain "distance labels" and can carry out the augmentation in O(n) time.**
  - **The total time to maintain and update all distance labels is O(nm).**
  - **The total number of augmentations is O(nm).**

Conclusion. The total running time is $O(n^2m)$.

Distance Labels

A ***distance label*** is a function $d: N \rightarrow Z^+$. A distance label is said to be ***valid*** if it satisfies the following:

$d(t) = 0$.

$d(i) \leq d(j) + 1$ for each $(i,j) \in G(x)$.

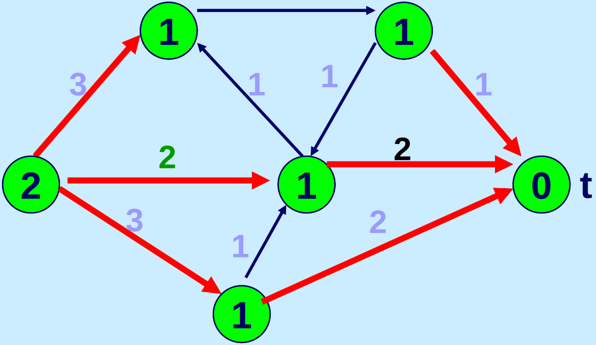
An arc $(i,j) \in G(x)$ is *admissible* if $d(i) = d(j) + 1$.

An example of valid distance labels

The distance labels are on the nodes.

All arcs are in the residual network.

The admissible arcs are thick and red.

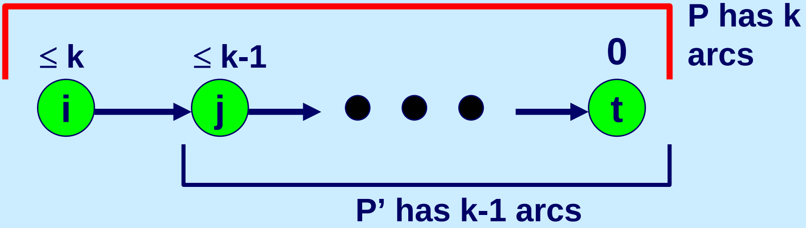


The labels would not be valid if there were an arc from “2” to “0”.

More on valid distance labels

Lemma. ***Let d( ) be a valid distance label. Then d(i) is a lower bound on the distance from i to t in the residual network. (The distance is measured in terms of the number of arcs.)***

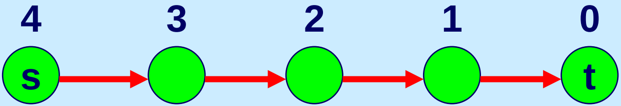
Proof. Let P be any path from i to t in G(x) with k arcs. We claim to show that d(i) ≤ k. Assume the claim is true for paths of k-1 or fewer arcs.



On Finding Paths shortest s-t paths

Lemma. ***If there is an admissible path P from s to t, then it is a shortest path.***

Proof. The length of the path is d(s) which is at most the length of the shortest path.



The shortest augmenting path algorithm

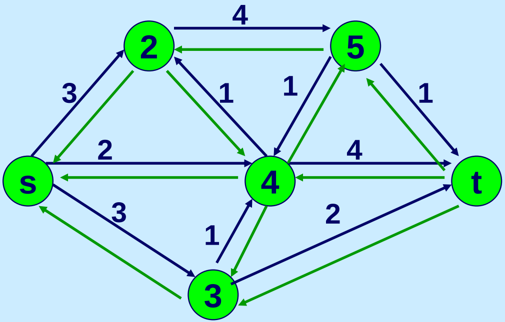
```
begin
    while d(s) < n do
    begin
        if there is a node  with d(i) ≤ d(s) and no admissible
           arcs from j then Relabel(i)
        else find an admissible path from s to t and augment
           flow along the path
    end
end

Procedure Relabel(i)
begin
    if there are no admissible arcs coming out of node i, then
       d(i) := 1 + min ( d(j) : r_ij > 0};
    if d(s) > n-1, then quit;
end
```

Shortest augmenting path animation

15.082 and 6.855J

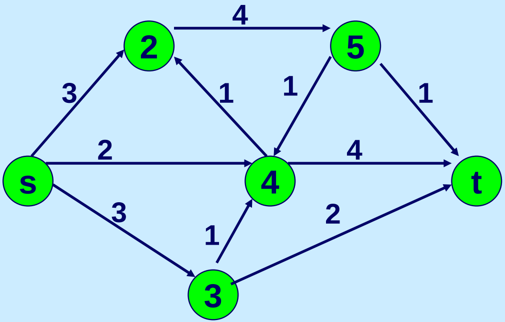
The Shortest Augmenting Path Algorithm for the Maximum Flow Problem

Shortest Augmenting Path



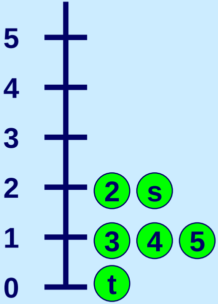
This is the original network, plus reversals of the arcs.

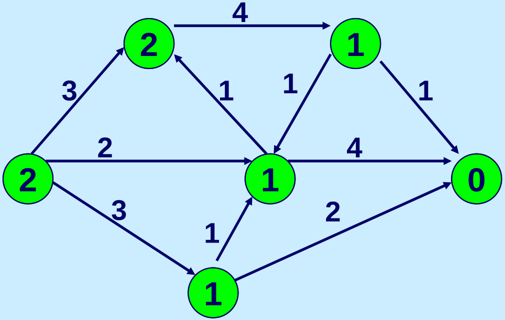
Shortest Augmenting Path



This is the original network, and the original residual network.

Initialize Distances

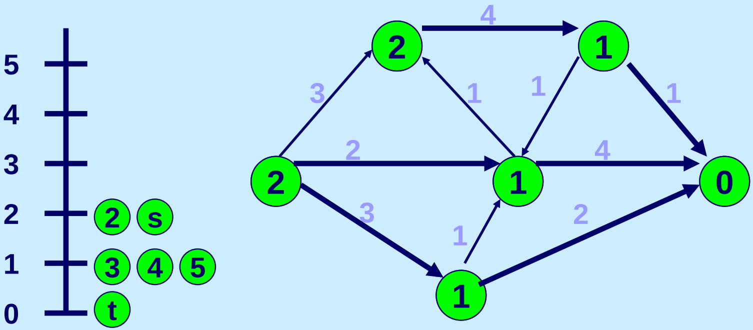




The node label henceforth will be the distance label.

d(j) is at most the distance of j to t in G(x)

Representation of admissible arcs

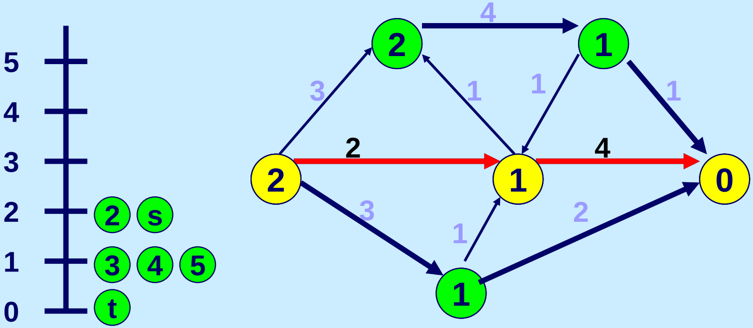


An arc (i,j) is ***admissible*** if d(i) = d(j) + 1.

An s-t path of admissible arcs is a shortest path

Admissible arcs will be represented with thick lines

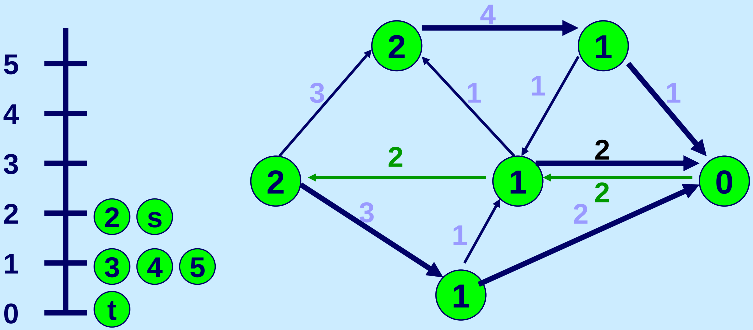
Look for a shortest s-t path



Start with s and do a depth first search using admissible arcs.

Next. Send flow, and update the residual capacities.

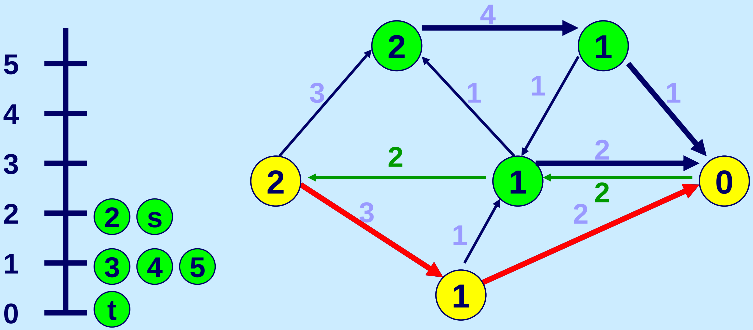
Update residual capacities



Here are the updated residual capacities.

We will update distance labels later, as needed.

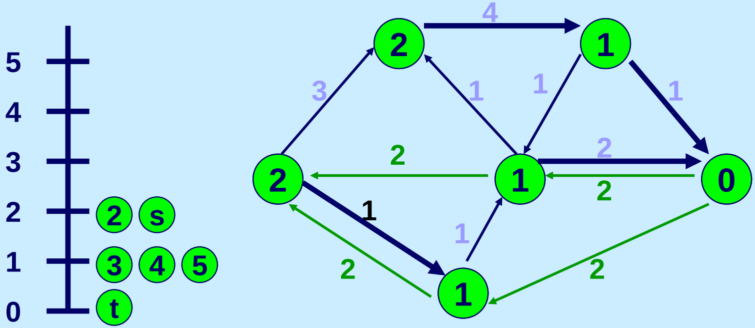
Look for a shortest s-t path



Start with s and do a depth first search using admissible arcs.

Next. Send flow, and update the residual capacities.

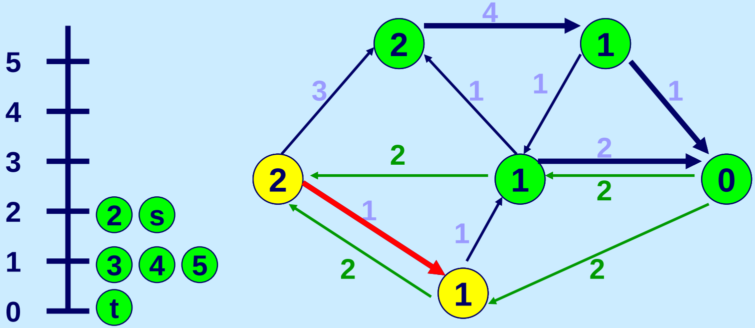
Update residual capacities



Here are the updated residual capacities.

We will update distance labels later, as needed.

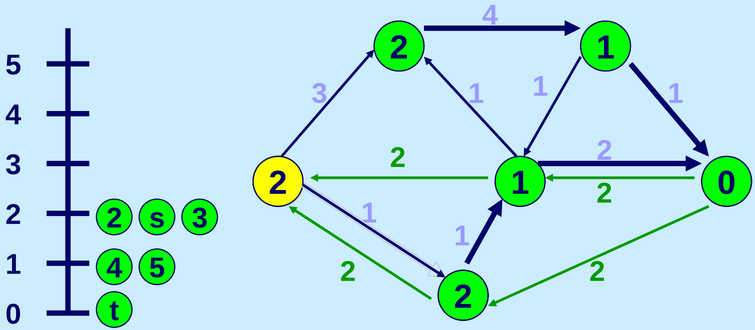
Search for a shortest s-t path



Start with s and do a depth first search using admissible arcs.

If there are no admissible arcs from i, then relabel(i) and reverse along the path leading to i.

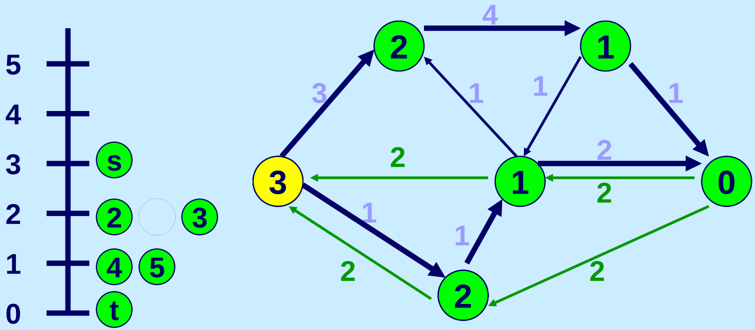
Update distances and path



Start with s and do a depth first search using admissible arcs.

If there are no admissible arcs from i, then relabel(i) and reverse one arc along the path leading from s .

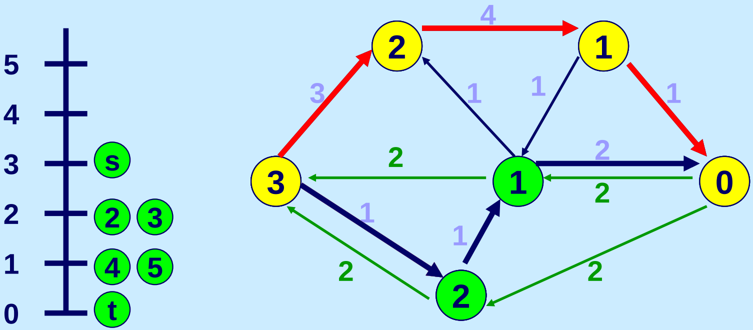
Update distances and path



Start with s and do a depth first search using admissible arcs.

If there are no admissible arcs from i, then relabel(i) and reverse one arc along the path leading from s.

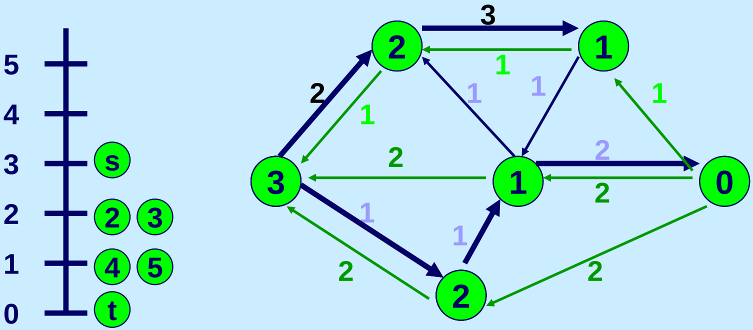
Look for a shortest s-t path



Continue the path from where it left off.

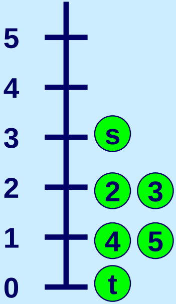
If the path reaches t, then send flow and update residual capacities.

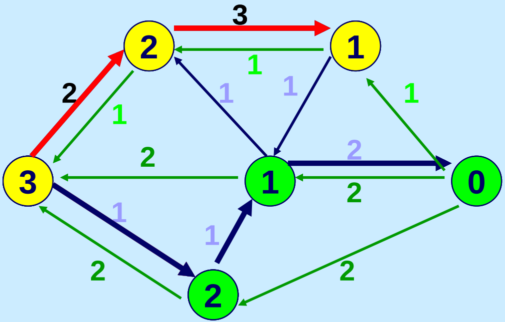
Update residual capacities



Here are the updated residual capacities.

Search for a shortest s-t path

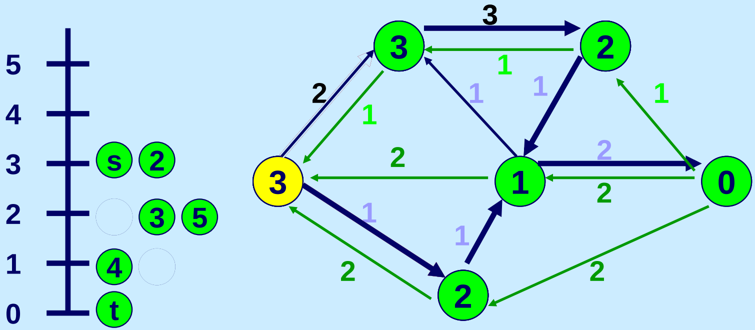




Search for a shortest s-t path starting from s

If there are no admissible arcs from i, then relabel(i) and reverse one arc along the path leading from s.

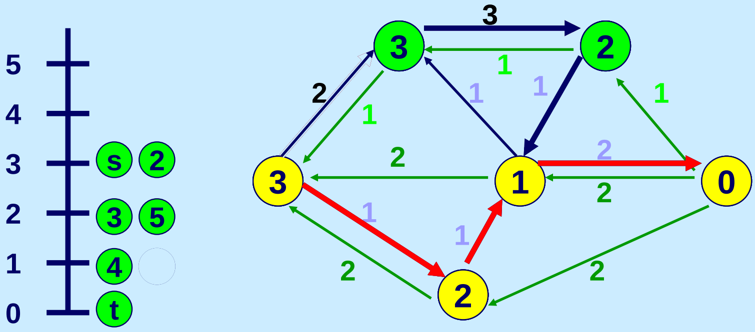
Search for a shortest s-t path



Search for a shortest s-t path starting from s

If there are no admissible arcs from i, then relabel(i) and reverse one arc along the path leading from s.

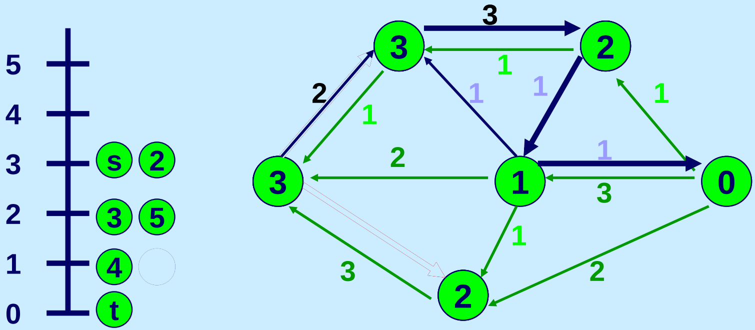
Search for a shortest s-t path



Search for a shortest s-t path starting from s

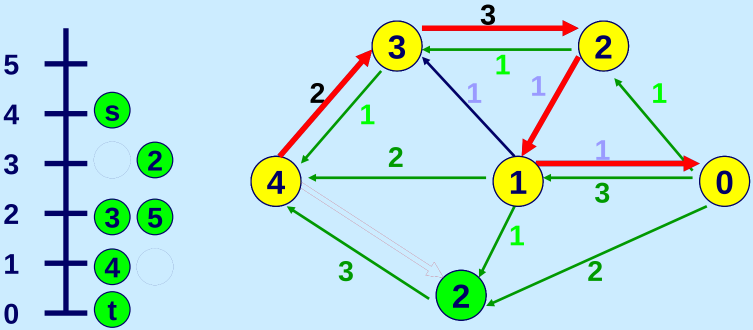
If the path reaches t, then send flow and update residual capacities.

update the residual capacities



Here are the updated residual capacities

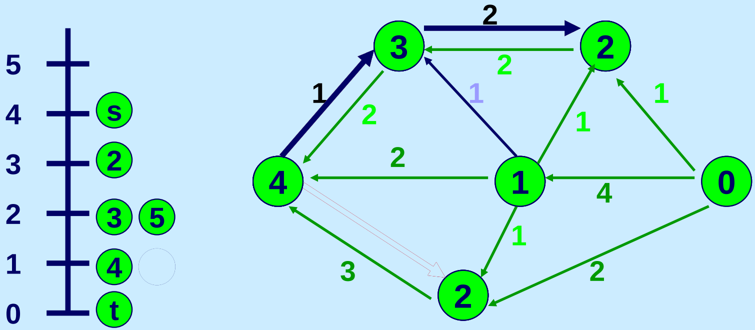
Search for a shortest s-t path



Search for a shortest s-t path

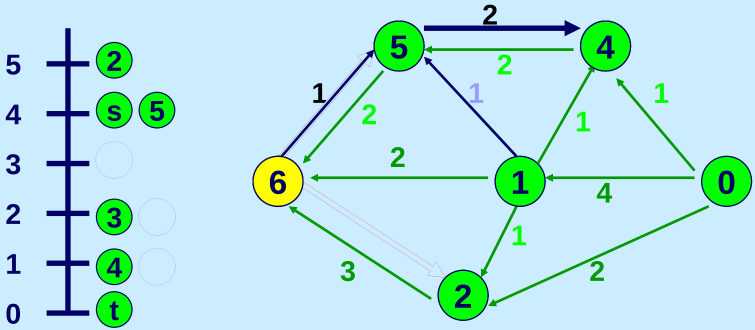
Next: update the residual capacities

Update the residual capacities



Here are the updated residual capacities

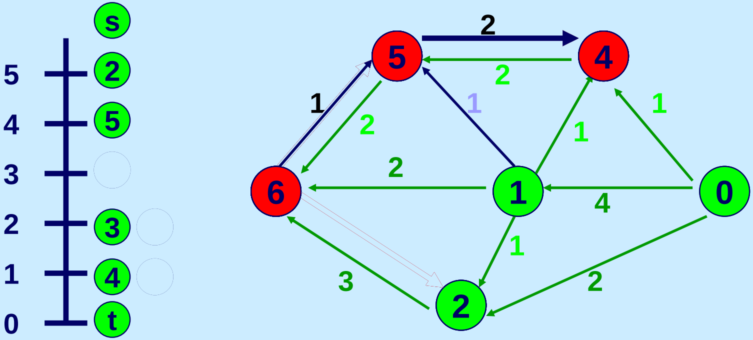
Look for a shortest s-t path



update distance labels and path

If d(s) > n-1, then there is no path from s to t

These are the residual capacities for the optimum flow



There is no s-t path in the residual network

A min cut has S = {s, 2, 5}.

Comments on the run time analysis

- **Bound the relabels, and the time for relabels**
  - **$O(n^2)$ relabels, $O(nm)$ time.**

- **Bound the number of augmentations, and the time to carry out the augmentations**
  - **$O(nm)$ augmentations**
  - **$O(n^2m)$ arcs in augmentations**
  - **$O(n^2m)$ time.**

- **Bound the time spent looking for augmentations.**
  - **$O(n^2m)$ time spent identifying the arcs in augmentations.**

Bounding the number of relabels.

Claim: after a relabel of node i, the distances are still valid, and the distance label of node i **strictly increased**.

Claim: Once d(i) > n-1, there is no path from node i to the sink node t, and so one can ignore node i subsequently.

Conclusion: There can be at most n relabels of node i, and at most n^2 relabels in total.

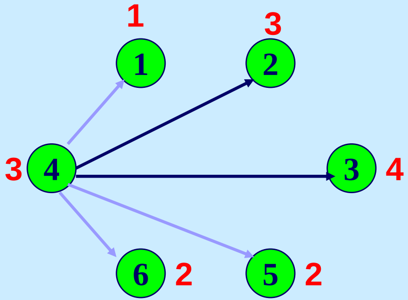
Bounding the time for relabels

Tail	Head	Res. Cap	Admissible ?
4	1	0	No
4	2	1	No
4	3	4	No
4	5	0	No
4	6	0	No

Maintain a current arc for each adjacency list.

Scan through A(4).

d(3) := 4



Each arc in A(4) is scanned once per relabel, at most n times over all relabels.

Total time for relabels:

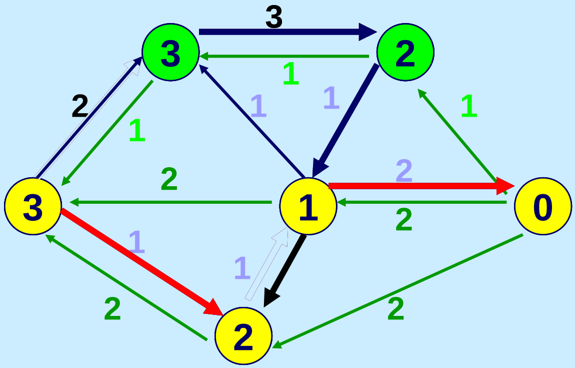
O(nm).

Bounding the Number of Augmentations

- **If an augmentation uses up the residual capacity of an arc, then the arc is said to be saturated.**
- **At least one arc is saturated at each augmentation.**
- **If arc (i,j) is saturated, then it is not admissible until flow is sent from j to i, and this cannot happen until d(j) increases. (see next slide)**
- **Conclusion: each arc is saturated at most n times.**
- **Corollary. There are O(nm) augmentations.**

- **The number of arcs in these augmentations is $O(n^2m)$.**

(2,1) is saturated in the augmentation



After the saturation, arc (2,1) is deleted from G(x).

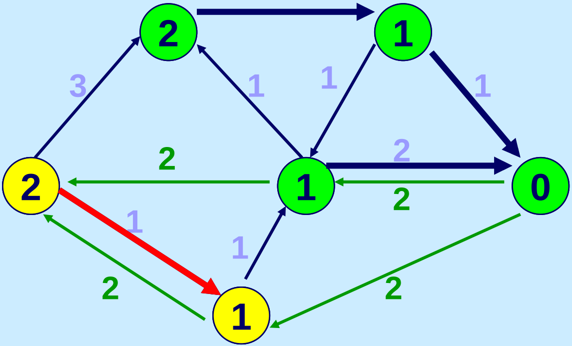
It doesn't get added until there is flow in (1,2)

But for that, the distance label must increase from 1 to 3.

And to send flow back, the distance label must increase from 2 to 4.

Time spent looking for augmentations

We need to find admissible arcs, and know when they do not exist.



Start with s and do a depth first search using admissible arcs.

If there are no admissible arcs from i, then relabel(i) and reverse along the path leading to i.

Bounding number of arcs in paths

Each arc added to a path either ends up being reversed or ends up in an augmentation.

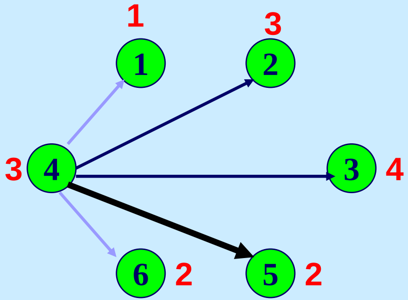
$O(n^2m)$ arcs in augmentation

$O(n^2)$ arcs in reversals, since a reversal immediately follows a relabel.

$O(n^2m)$ arcs added to paths in total.

Last step: finding admissible arcs

Tail	Head	Res. Cap	Admissible ?
4	1	0	No
4	2	1	No
4	3	4	No
4	5	2	Yes
4	6	0	No



Scan arcs in A(4) looking for an admissible arc.

key observation: if (4, j) is not admissible, it cannot be admissible again until after node 4 is relabeled.

So, current arc is moved at most |A(4)| times between relabels of node 4.

Summary

Applications of Maximum Flow, including implications of the max flow min cut theorem.

The shortest augmenting path algorithm has O(nm) augmentations, and takes $O(n^2m)$ time.

Use of distance labels to identify how to send flow.

Next lecture: an algorithm that does not rely on augmenting paths.

Review of Augmenting Paths

At each iteration: maintain a flow x

Let G(x) be the residual network

At each iteration, find a path from s to t in G(x).

In the shortest augmenting path algorithm, we kept distance labels d( ), and we sent flow along the shortest path in G(x).

Preflows

At each intermediate stages we permit more flow arriving at nodes than leaving (except for s)

A ***preflow*** is a function $x: A \rightarrow R$ s.t. $0 \leq x \leq u$ and such that

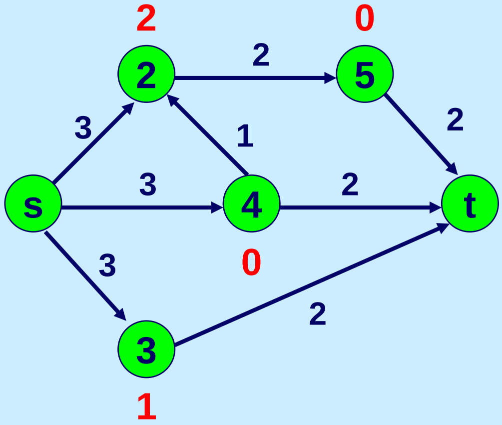
$$e(i) = \sum_{j \in N} x_{ji} - \sum_{j \in N} x_{ij} \geq 0,$$

for all $i \in N - \{s, t\}$.

i.e., $e(i)$ = ***excess*** at i = net excess flow into node i.

The excess is required to be nonnegative.

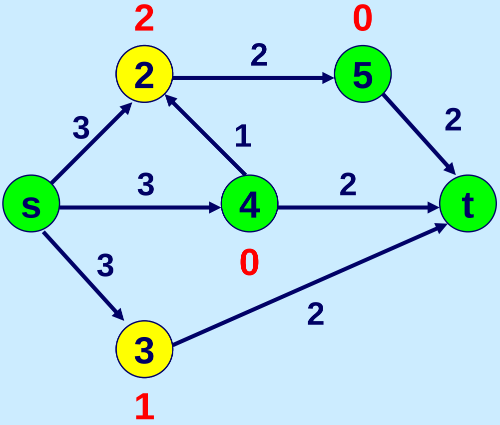
A Feasible Preflow



The *excess* *e(j)* at each node j ≠ s, t is the flow in minus the flow out.

Note: total excess = flow out of s minus flow into t.

Active nodes



Nodes with positive excess are called ***active***.

The preflow push algorithm will try to push flow from active nodes towards the sink, relying on d( ).

Review of Distance Labels

Distance labels **d( ) are *valid* for G(x) if**

i. $d(t) = 0$

ii. $d(i) \leq d(j) + 1$ **for each** $(i,j) \in G(x)$

Defn. An arc (i,j) is *admissible* if $r_{ij} > 0$ **and** $d(i) = d(j) + 1$.

Lemma. ***Let d( ) be a valid distance label. Then d(i) is a lower bound on the distance from i to t in the residual network.***

Push/Relabel, the fundamental subroutine

Suppose we have selected an active node i.

Procedure Push/Relabel(i)

begin

if the network contains an admissible arc (i,j) then
push $\delta := \min\{ e(i), r_{ij} \}$ units of flow from i to j;

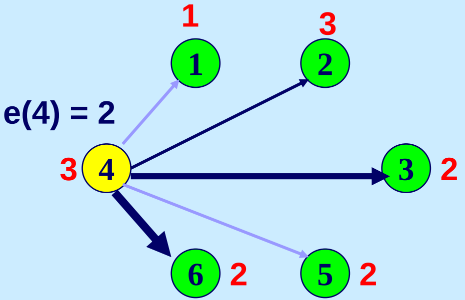
else replace d(i) by $\min\{d(j) + 1 : (i,j) \in A(i)$ and $r_{ij} > 0\}$

end;

Pushing using current arcs

Tail	Head	Res. Cap	Admissible ?
4	1	0	No
4	2	1	No
4	3	4	Yes
4	5	0	No
4	6	2	Yes

Suppose that node 4 is active, and has excess.



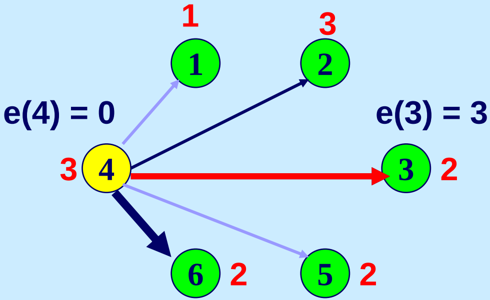
Scan arcs in A(4) one at a time using "Current Arc" till an admissible arc is found.

Push on (4,3)

Pushing on (4,3)

Tail	Head	Res. Cap	Admissible ?
4	1	0	No
4	2	1	No
4	3	2	Yes
4	5	0	No
4	6	2	Yes

Push on (4,3)



Send min (e(4), r_{43}) = 2 units of flow.

Update the residual capacities and excesses.

For the next push from node 4, start with arc (4,3).

Goldberg-Tarjan Preflow Push Algorithm

Procedure Preprocess

begin

  x :=0;

  compute the exact distance labels d(i) for each node;

  $x_{sj} := u_{sj}$ for each arc (s,j) $\in$ A(s);   d(s) := n;

end

Algorithm PREFLOW-PUSH;

begin

  preprocess;

  while there is an active node i **do**

  begin

    select an active node i;

    push/relabel(i);

  end;

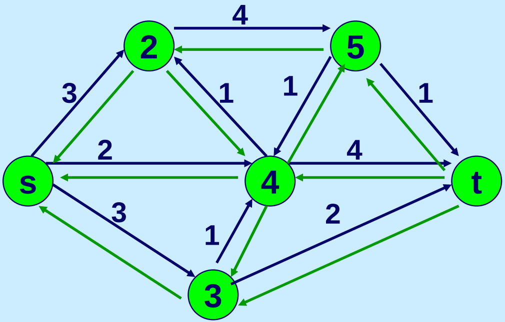
end;

Preflow Push Animation

15.082 and 6.855J

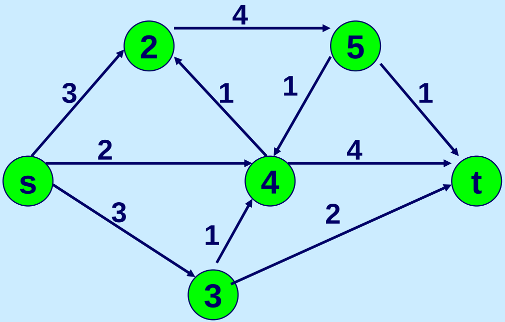
The Goldberg-Tarjan Preflow Push Algorithm for the Maximum Flow Problem

Preflow Push



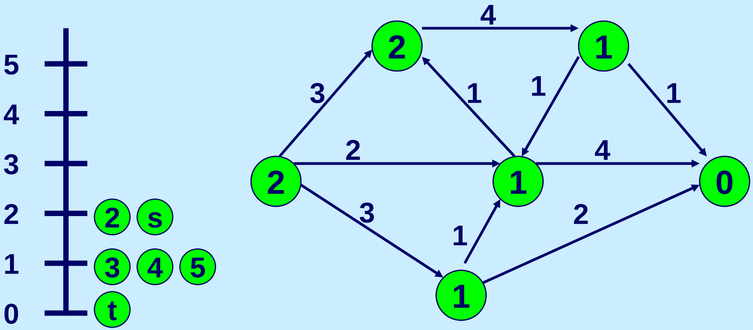
This is the original network, plus reversals of the arcs.

Preflow Push



This is the original network, and the original residual network.

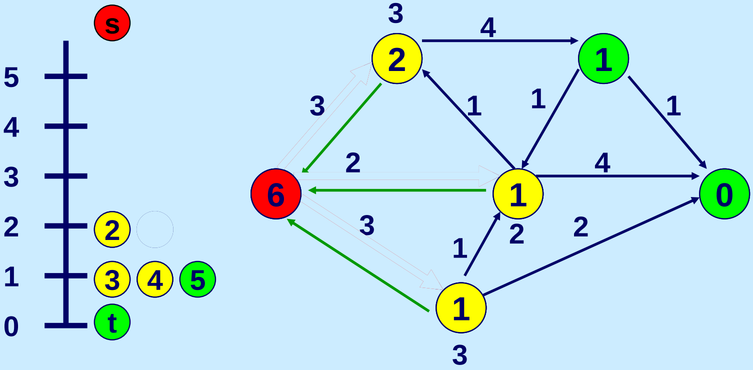
Initialize Distances



The node label henceforth will be the distance label.

d(j) is at most the distance of j to t in G(x)

Saturate Arcs out of node s

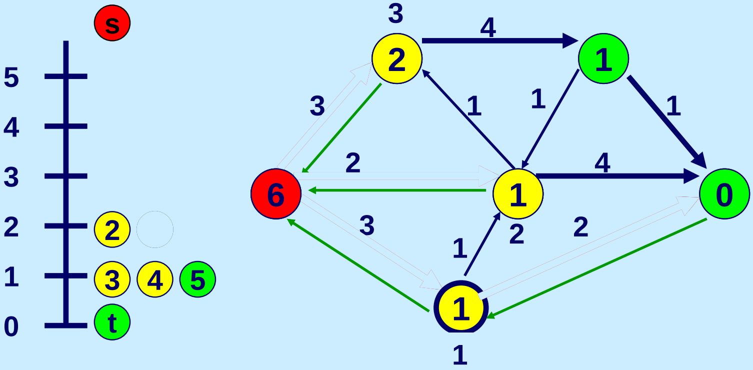


Saturate arcs out of node s.

Move excess to the adjacent arcs

Relabel node s after all incident arcs have been saturated.

Select, then relabel/push

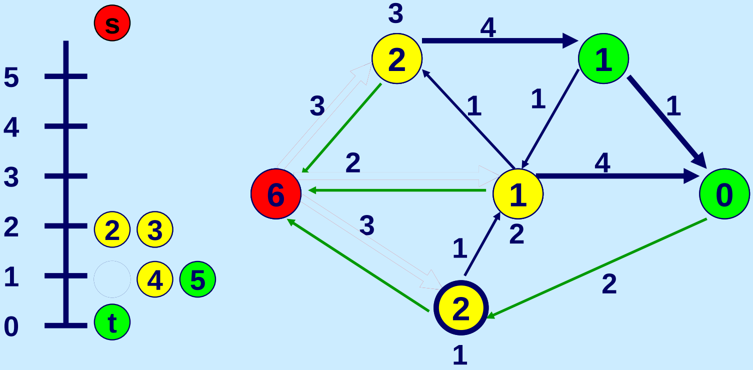


Select an active node, that is, one with excess

Push/Relabel

Update excess after a push

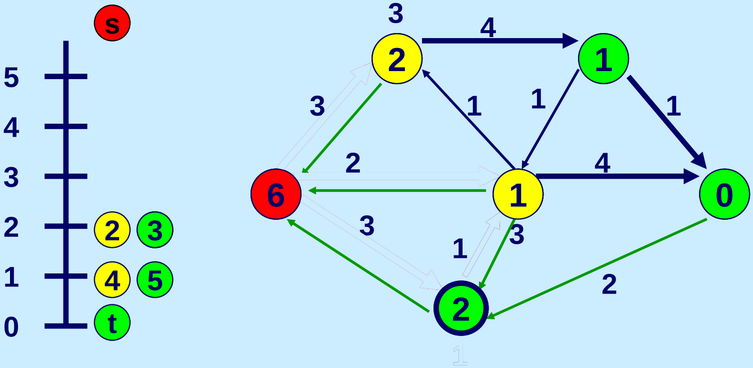
Select, then relabel/push



Select an active node, that is, one with excess

No arc incident to the selected node is admissible. So relabel.

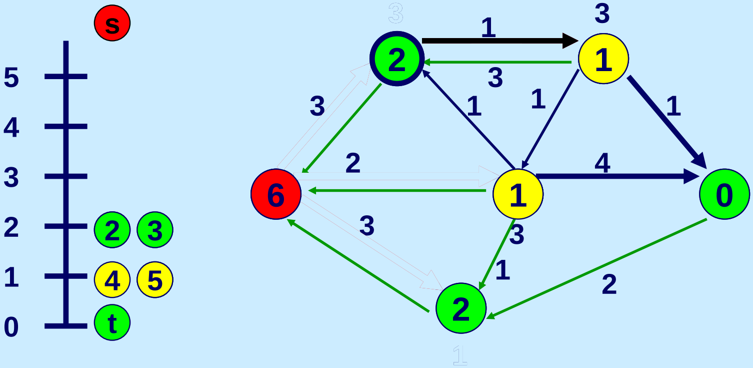
Select, then relabel/push



Select an active node, that is, one with excess

Push/Relabel

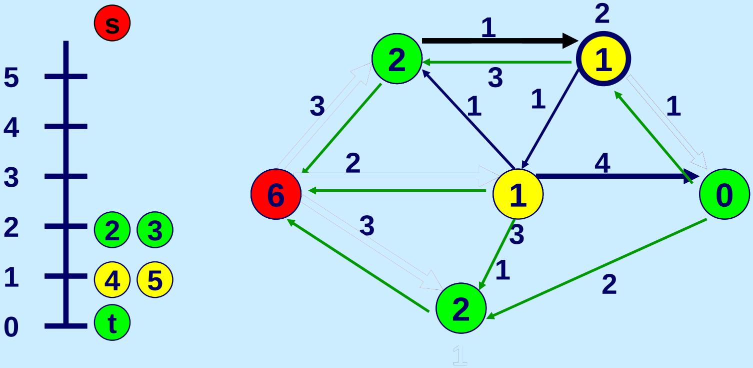
Select, then relabel/push



Select an active node.

Push/Relabel

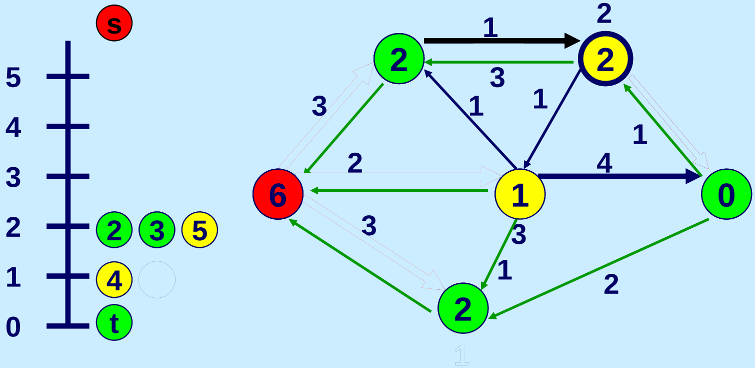
Select, then relabel/push



Select an active node.

Push/Relabel

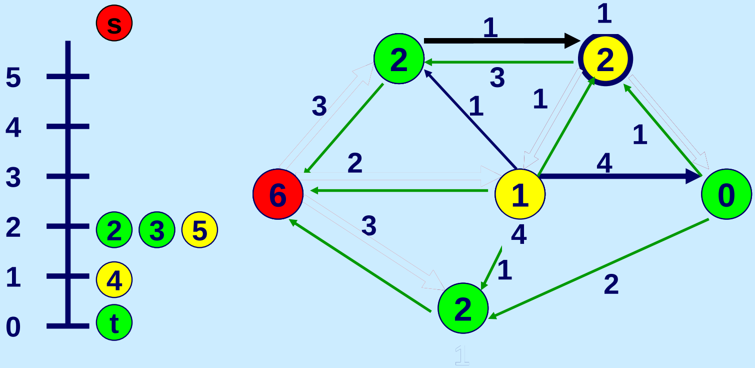
Select, then relabel/push



Select an active node.

There is no incident admissible arc. So Relabel.

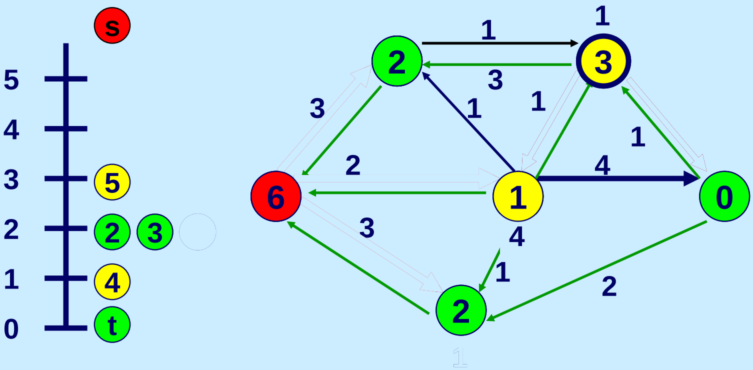
Select, then relabel/push



Select an active node.

Push/Relabel

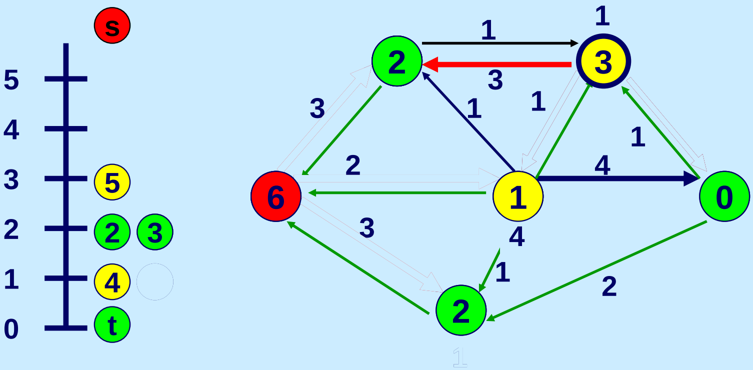
Select, then relabel/push



Select an active node.

There is no incident admissible arc. So relabel.

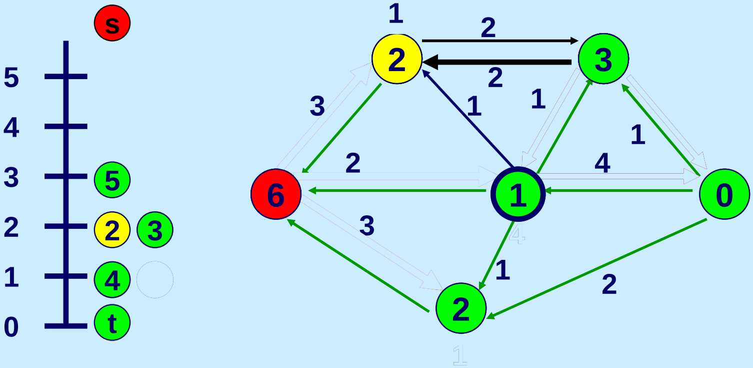
Select, then relabel/push



Select an active node.

Push/Relabel

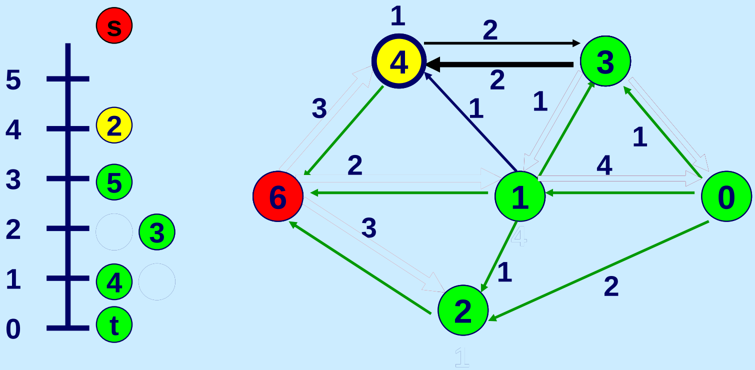
Select, then relabel/push



Select an active node.

Push/Relabel

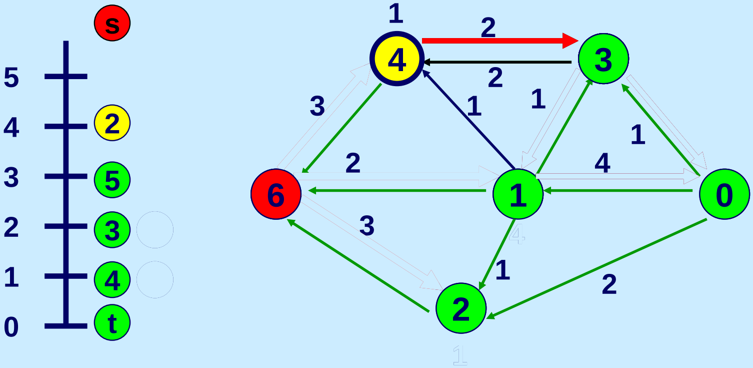
Select, then relabel/push



Select an active node.

Push/Relabel

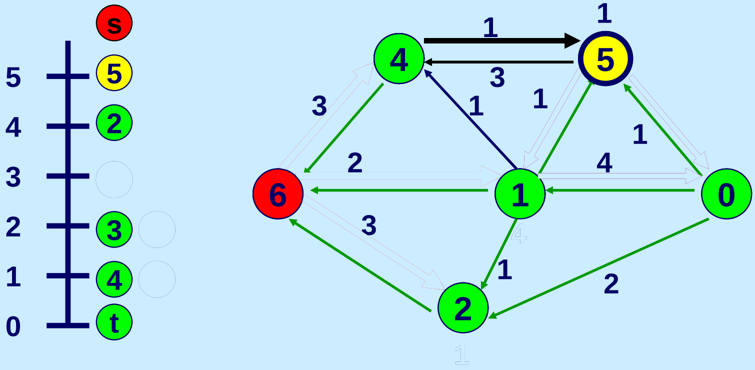
Select, then relabel/push



Select an active node.

Push/Relabel

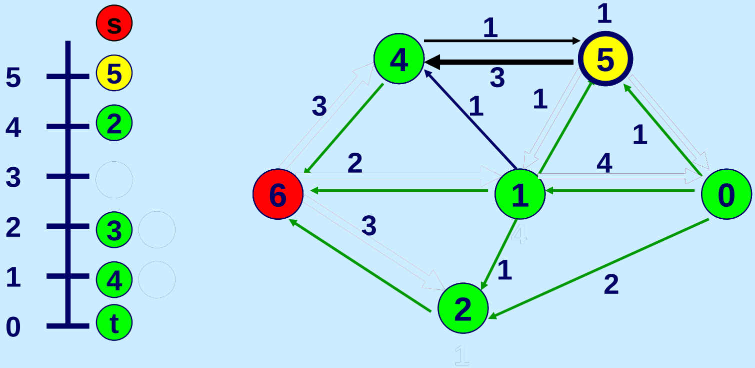
Select, then relabel/push



Select an active node.

Push/Relabel

Select, then relabel/push



One can keep pushing flow between nodes 2 and 5 until eventually all flow returns to node s.

There are no paths from nodes 2 and 5 to t, and there are ways to speed up the last iterations.

Preview of Results on Preflow Push

The preflow push algorithm is superb both in theory and in practice.

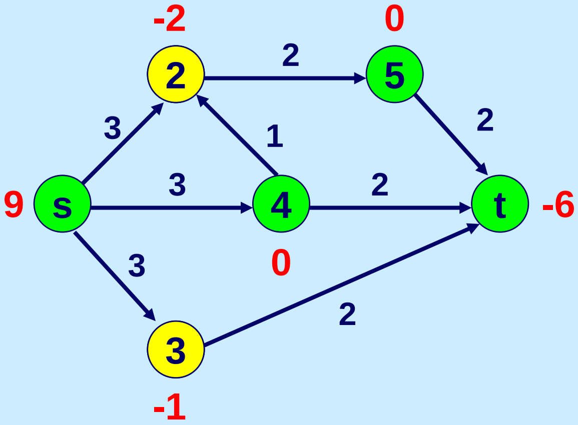
To prove:

1. $d(j) < 2n$ throughout the algorithm
2. The algorithm terminates with a maximum flow
3. The number of steps excluding non-saturating pushes is $O(nm)$
4. The number of non-saturating pushes is $O(n^2m)$.

Further improvements are possible.

A lemma needed for the time bounds

Lemma 7.11. At each stage of the algorithm, there is a path in G(x) from each node i with e(i) > 0 to node s.



Here is the same preflow as before

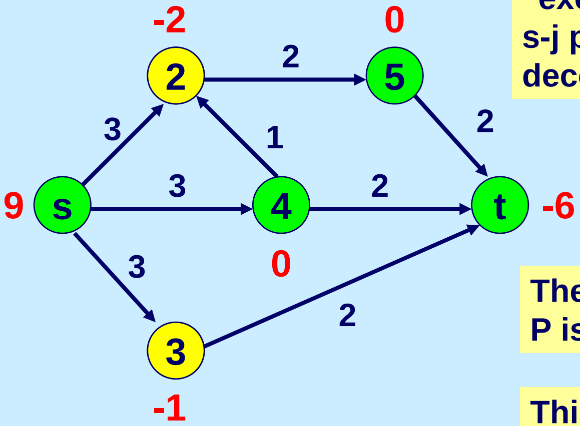
As a flow, the "excesses" are negative supplies.

Proof. Apply flow decomposition to the flow x.

2 units in s-2-5-t; 2 units in s-4-t 2 units in s-3-t

1 unit in s-2 1 unit in s-4-2

1 unit in s-3



For each node j with "excess" there is an s-j path P in the flow decomposition.

The reversal of P is in G(x).

This completes the proof.

Bounding d(j)

Lemma 7.12. ***For each node j with excess, $d(j) \leq 2n - 1$.***

Proof. **Let f(j) be the length of the shortest path in G(x) from node j to node s. then**

$$d(j) \leq d(s) + f(j) \leq n + (n-1) = 2n - 1.$$

Lemma 7.13. ***Each node is relabeled fewer than 2n times, and so the total number of relabels is fewer than $2n^2$.***

Theorem. ***The preflow push algorithm is finite and terminates with the maximum s-t flow.***

Proof. **The algorithm is finite because the distance labels can increase at most 2n times each.**

The algorithm ends with a flow (as opposed to preflow) because if there were any active node, the algorithm would not end.

At the end, d(s) = n. Since d(s) is a lower bound on the shortest s-t path in G(x), there is no s-t path in G(x), and the flow is maximum.

Bounding the time for relabels

Time to relabel.

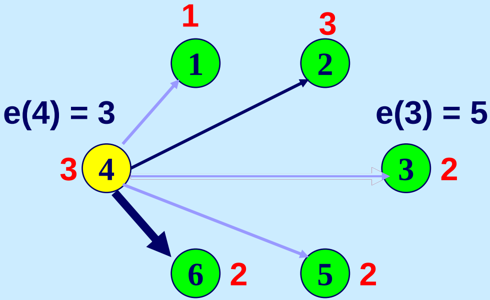
- **To relabel node j, one needs to scan each arc of A(j)**
- **So, each arc gets scanned at most 2n times**

- **Total relabel time is O(nm)**

Bounding the remaining steps

Tail	Head	Res. Cap	Admissible ?
4	1	0	No
4	2	1	No
4	3	0	Yes
4	5	0	No
4	6	9	Yes

Push Flow from node 4.



Send min (e(4), r_{43}) = 4 units of flow.

Update the residual capacities and excesses.

In a saturating push in (i,j) the flow is r_{ij}.

Computation time

Moving "current arc" for i. |A(i)| steps between relabels. This run time is no more than the time for a relabel of node i.

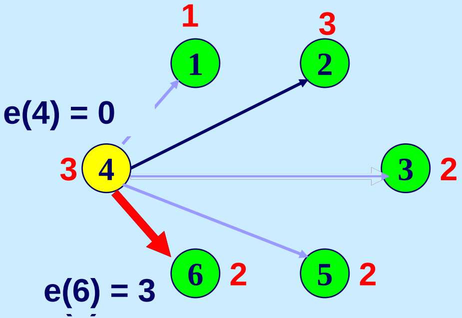
Saturating pushes for i. At most |A(i)| between relabels of i. Once an arc is saturated, the current arc will increase by 1.

But the bottleneck are the *non-saturating* pushes.

Bounding the remaining steps

Tail	Head	Res. Cap	Admissible ?
4	1	0	No
4	2	1	No
4	3	0	No
4	5	0	No
4	6	6	Yes

Push Flow from node 4.



Send min (e(4), r_{46}) = 3 units of flow.

Update the residual capacities and excesses.

Note: CurrentArc will stay on node (4,6)

Review of run time analysis

Time to select active nodes is O(1) per push since we can maintain a set of active nodes.

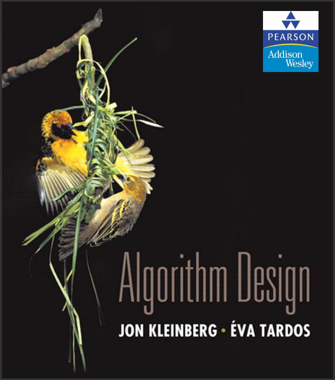
We bounded relabels and relabel time

We bounded all time in pushing, except for non-saturating pushes

- Time to advance “currentArc” is bounded by the time to relabel
- Time for saturating pushes is bounded by the time to advance “currentArc”

Non-saturating pushes really are the bottleneck.

- We will use a different analysis technique to bound them



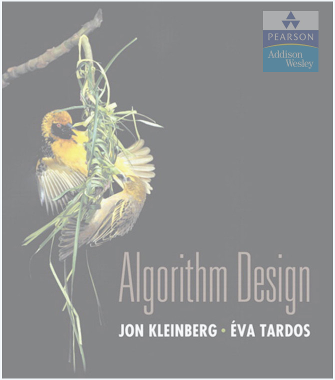
4. Greedy Algorithms II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on 3/5/18 2:00 PM



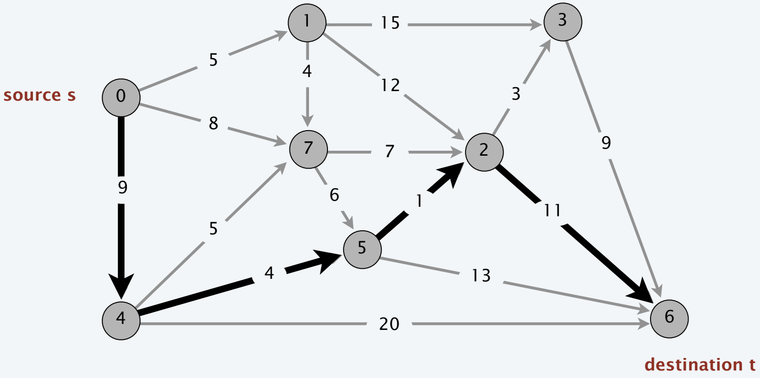
SECTION 4.4

4. GREEDY ALGORITHMS II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

Single-pair shortest path problem

Problem. Given a digraph $G = (V, E)$, edge lengths $\ell_e \geq 0$, source $s \in V$, and destination $t \in V$, find a shortest directed path from s to t.

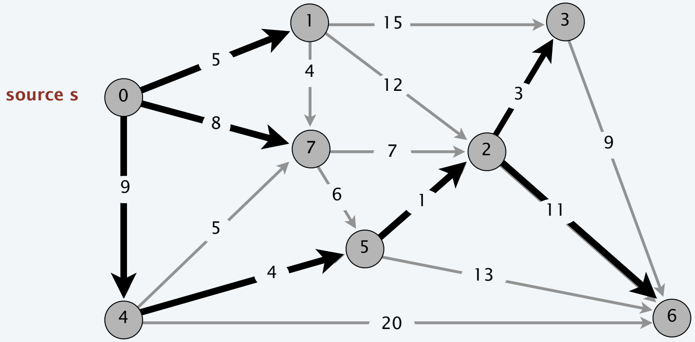


length of path = 9 + 4 + 1 + 11 = 25

Single-source shortest paths problem

Problem. Given a digraph $G = (V, E)$, edge lengths $\ell_e \geq 0$, source $s \in V$, find a shortest directed path from s to every node.

Assumption. There exists a path from s to every node.



shortest-paths tree

Shortest paths: quiz 1

Suppose that you change the length of every edge of G as follows. For which is every shortest path in G a shortest path in G′?

- **A.** Add 17.
- **B.** Multiply by 17.
- **C.** Either A or B.
- **D.** Neither A nor B.

Which variant in car GPS?

A. Single source: from one node s to every other node.

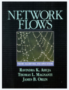
B. Single sink: from every node to one node t.

C. Source–sink: from one node s to another node t.

D. All pairs: between all pairs of nodes.

Shortest path applications

- PERT/CPM.
- Map routing.
- Seam carving.
- Robot navigation.
- Texture mapping.
- Typesetting in LaTeX.
- Urban traffic planning.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Optimal truck routing through given traffic congestion pattern.



Network Flows: Theory, Algorithms, and Applications,
by Ahuja, Magnanti, and Orlin, Prentice Hall, 1993.

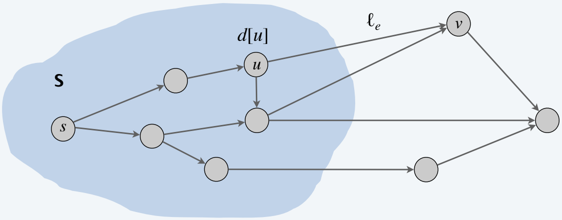
Dijkstra's algorithm (for single-source shortest paths problem)

Greedy approach. Maintain a set of explored nodes S for which algorithm has determined $d[u]$ = length of a shortest $s \rightarrow u$ path.

- Initialize $S \leftarrow \{ s \},\ d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) \;=\; \min_{e\,=\,(u,v)\,:\,u \in S} \boxed{d[u] + \ell_e}$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$



Dijkstra's algorithm (for single-source shortest paths problem)

Greedy approach. Maintain a set of explored nodes S for which algorithm has determined $d[u]$ = length of a shortest $s \rightsquigarrow u$ path.

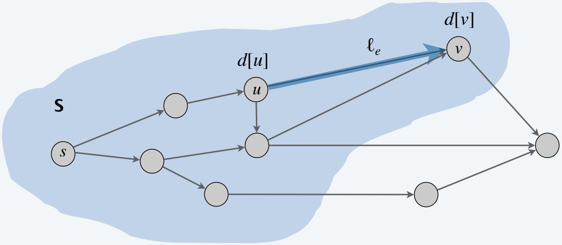
- Initialize $S \leftarrow \{ s \}$, $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

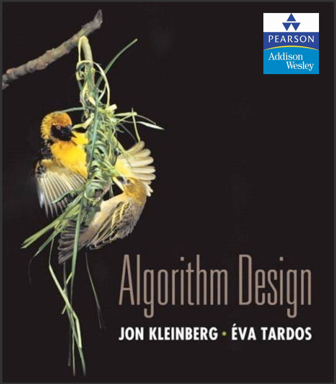
$$\pi(v) = \min_{e = (u,v) \,:\, u \in S} \boxed{d[u] + \ell_e}$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

  add v to S, and set $d[v] \leftarrow \pi(v)$.

- To recover path, set $pred[v] \leftarrow e$ that achieves min.





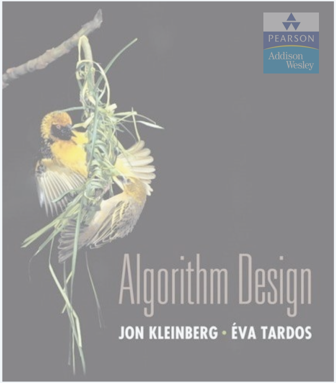
Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

4. GREEDY ALGORITHMS II

- *Dijkstra's algorithm demo*
- *Dijkstra's algorithm demo (efficient implementation)*

Last updated on 11/15/17 10:10 AM



4. Greedy Algorithms II

- *Dijkstra's algorithm demo*
- *Dijkstra's algorithm demo (efficient implementation)*

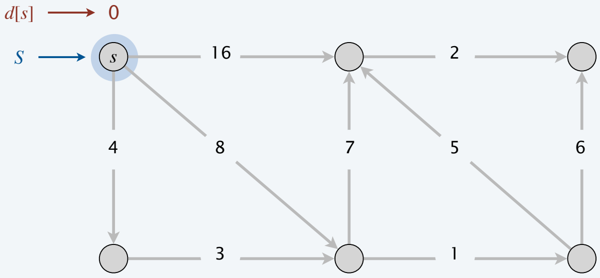
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) \;=\; \min_{e\,=\,(u,v)\;:\;u \in S} \boxed{d[u] + \ell_e}$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow \text{argmin}$.



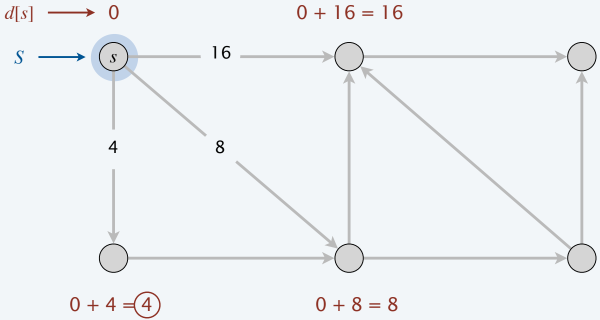
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) \;=\; \min_{e\,=\,(u,v)\,:\,u \in S} \boxed{d[u] + \ell_e}$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.



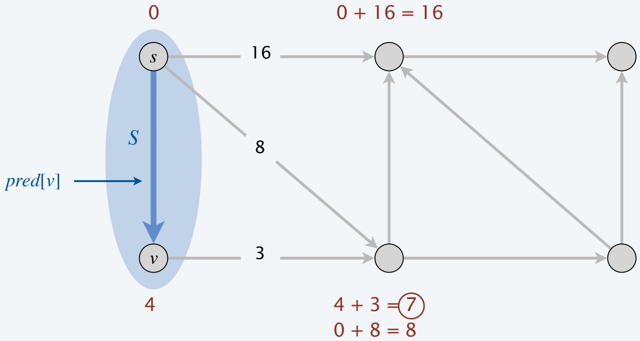
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) \; = \min_{e \,=\, (u,v) \,:\, u \in S} \boxed{d[u] + \ell_e}$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.



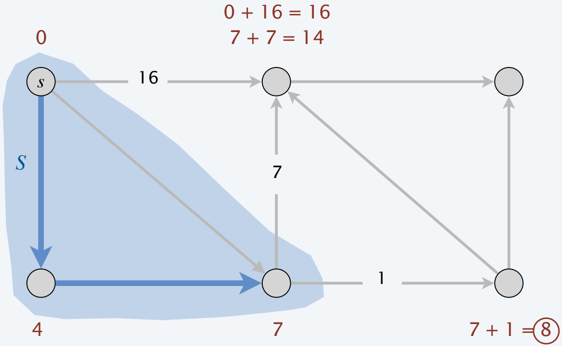
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) \;=\; \min_{e\,=\,(u,v)\,:\,u \in S} \boxed{d[u] + \ell_e}$$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow \text{argmin}$.

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$



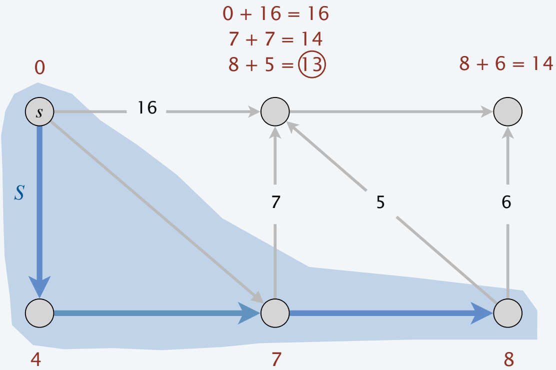
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) \;=\; \min_{e\,=\,(u,v)\;:\;u\in S} \boxed{d[u] + \ell_e}$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.



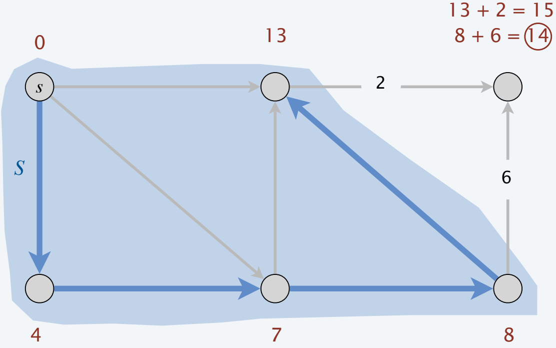
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) \;=\; \min_{e\,=\,(u,v)\,:\,u \in S} \boxed{d[u] + \ell_e}$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.

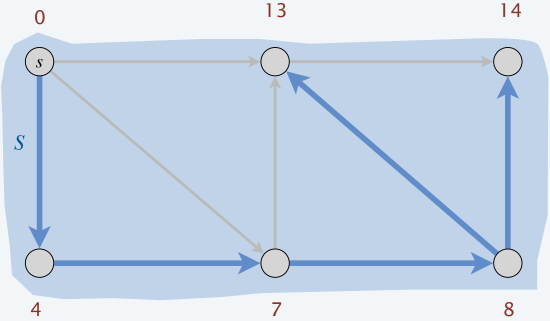


Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) \;=\; \min_{e\,=\,(u,v)\,:\,u\in S} \; d[u] + \ell_e$$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow \text{argmin}$.

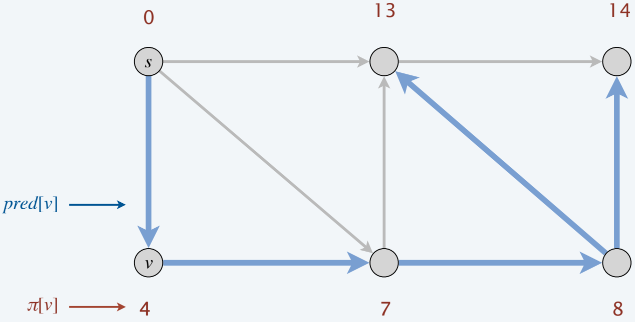


Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) \;=\; \min_{e\,=\,(u,v)\;:\;u \in S} \; d[u] + \ell_e$$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.



Dijkstra's algorithm: proof of correctness

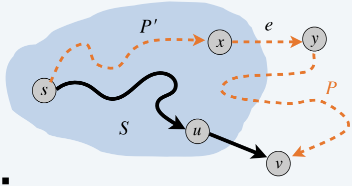
Invariant. For each node $u \in S$: $d[u]$ = length of a shortest $s \leadsto u$ path.

Pf. [by induction on $|S|$]

Base case: $|S| = 1$ is easy since $S = \{ s \}$ and $d[s] = 0$.

Inductive hypothesis: Assume true for $|S| \geq 1$.

- Let v be next node added to S, and let (u, v) be the final edge.
- A shortest $s \leadsto u$ path plus (u, v) is an $s \leadsto v$ path of length $\pi(v)$.
- Consider any other $s \leadsto v$ path P. We show that it is no shorter than $\pi(v)$.
- Let $e = (x, y)$ be the first edge in P that leaves S, and let P' be the subpath from s to x.
- The length of P is already $\geq \pi(v)$ as soon as it reaches y:



$$\underbrace{\ell(P) \;\geq\; \ell(P') + \ell_e}_{\text{non-negative lengths}} \;\;\underbrace{\geq\; d[x] + \ell_e}_{\text{inductive hypothesis}} \;\;\underbrace{\geq\; \pi(y)}_{\text{definition of } \pi(y)} \;\;\underbrace{\geq\; \pi(v)}_{\text{Dijkstra chose } v \text{ instead of } y} \quad \blacksquare$$

Dijkstra's algorithm: efficient implementation

Critical optimization 1. For each unexplored node $v \notin S$:
explicitly maintain $\pi[v]$ instead of computing directly from definition

$$\pi(v) \;=\; \min_{e\,=\,(u,v)\,:\,u \in S} \; d[u] + \ell_e$$

- For each $v \notin S$: $\pi(v)$ can only decrease (because set S increases).

- More specifically, suppose u is added to S and there is an edge $e = (u, v)$ leaving u. Then, it suffices to update:

$$\pi[v] \leftarrow \min \{\, \pi[v],\; \pi[u] + \ell_e) \,\}$$

recall: for each $u \in S$,
$\pi[u] = d[u] =$ length of shortest $s \rightarrow u$ path

Critical optimization 2. Use a min-oriented priority queue (PQ)
to choose an unexplored node that minimizes $\pi[v]$.

Dijkstra's algorithm: efficient implementation

Implementation.

- Algorithm maintains $\pi[v]$ for each node v.
- Priority queue stores unexplored nodes, using $\pi[\cdot]$ as priorities.
- Once u is deleted from the PQ, $\pi[u]$ = length of a shortest $s \rightsquigarrow u$ path.

DIJKSTRA (V, E, ℓ, s)

FOREACH $v \neq s$: $\pi[v] \leftarrow \infty$, $pred[v] \leftarrow null$; $\pi[s] \leftarrow 0$.

Create an empty priority queue pq.

FOREACH $v \in V$: INSERT$(pq, v, \pi[v])$.

WHILE (IS-NOT-EMPTY(pq))

  $u \leftarrow$ DEL-MIN(pq).

  FOREACH edge $e = (u, v) \in E$ leaving u:

    IF $(\pi[v] > \pi[u] + \ell_e)$

      DECREASE-KEY$(pq, v, \pi[u] + \ell_e)$.

      $\pi[v] \leftarrow \pi[u] + \ell_e$; $pred[v] \leftarrow e$.

Dijkstra's algorithm: which priority queue?

Performance. Depends on PQ: n INSERT, n DELETE-MIN, $\leq m$ DECREASE-KEY.

- Array implementation optimal for dense graphs. ⟵ $\Theta(n^2)$ edges
- Binary heap much faster for sparse graphs. ⟵ $\Theta(n)$ edges
- 4-way heap worth the trouble in performance-critical situations.

priority queue	INSERT	DELETE-MIN	DECREASE-KEY	total
unordered array	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
binary heap	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(m \log n)$
d-way heap (Johnson 1975)	$O(d \log_d n)$	$O(d \log_d n)$	$O(\log_d n)$	$O(m \log_{m/n} n)$
Fibonacci heap (Fredman–Tarjan 1984)	$O(1)$	$O(\log n)$ †	$O(1)$ †	$O(m + n \log n)$
integer priority queue (Thorup 2004)	$O(1)$	$O(\log \log n)$	$O(1)$	$O(m + n \log \log n)$

† amortized

How to solve the the single-source shortest paths problem in undirected graphs with positive edge lengths?

A. Replace each undirected edge with two antiparallel edges of same length. Run Dijkstra's algorithm in the resulting digraph.

B. Modify Dijkstra's algorithms so that when it processes node u, it consider all edges incident to u (instead of edges leaving u).

C. Either A or B.

D. Neither A nor B.

Theorem. [Thorup 1999] Can solve single-source shortest paths problem in undirected graphs with positive integer edge lengths in $O(m)$ time.

Remark. Does not explore nodes in increasing order of distance from s.

Undirected Single Source Shortest Paths with Positive Integer Weights in Linear Time

Mikkel Thorup
AT&T Labs—Research

The single source shortest paths problem (SSSP) is one of the classic problems in algorithmic graph theory: given a positively weighted graph G with a source vertex s, find the shortest path from s to all other vertices in the graph.

Since 1959 all theoretical developments in SSSP for general directed and undirected graphs have been based on Dijkstra's algorithm, visiting the vertices in order of increasing distance from s. Thus, any implementation of Dijkstra's algorithm sorts the vertices according to their distances from s. However, we do not know how to sort in linear time.

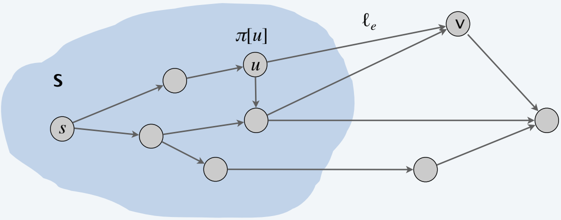
Here, a deterministic linear time and linear space algorithm is presented for the undirected single source shortest paths problem with positive integer weights. The algorithm avoids the sorting bottle-neck by building a hierarchical bucketing structure, identifying vertex pairs that may be visited in any order.

Extensions of Dijkstra's algorithm

Dijkstra's algorithm and proof extend to several related problems:

- Shortest paths in undirected graphs: $\pi[v] \le \pi[u] + \ell(u, v)$.
- Maximum capacity paths: $\pi[v] \ge \min\{\, \pi[u],\ c(u, v) \,\}$.
- Maximum reliability paths: $\pi[v] \ge \pi[u] \times \gamma(u, v)$.
- …

Key algebraic structure. Closed semiring (min-plus, bottleneck, Viterbi, …).

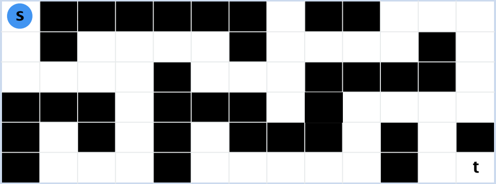


$$a + b = b + a$$
$$a + (b + c) = (a + b) + c$$
$$a + 0 = a$$
$$a \cdot (b \cdot c) = (a \cdot b) \cdot c$$
$$a \cdot 0 = 0 \cdot a = 0$$
$$a \cdot 1 = 1 \cdot a = a$$
$$a \cdot (b + c) = a \cdot b + a \cdot c$$
$$(a + b) \cdot c = a \cdot c + b \cdot c$$
$$a^* = 1 + a \cdot a^* = 1 + a^* \cdot a$$

GOOGLE'S FOO.BAR CHALLENGE

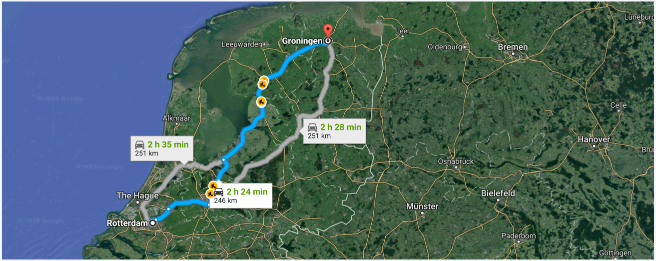
You have maps of parts of the space station, each starting at a prison exit and ending at the door to an escape pod. The map is represented as a matrix of 0s and 1s, where 0s are passable space and 1s are impassable walls. The door out of the prison is at the top left $(0, 0)$ and the door into an escape pod is at the bottom right $(w-1, h-1)$.

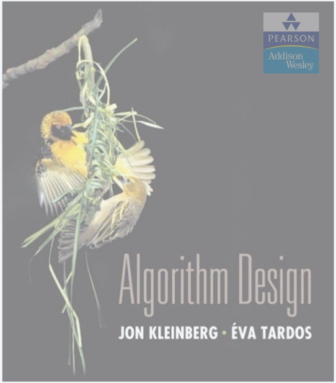
Write a function that generates the length of a shortest path from the prison door to the escape pod, where you are allowed to remove one wall as part of your remodeling plans.



Edsger Dijkstra

> “ *What’s the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path, which I designed in about 20 minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path.*” — Edsger Dijsktra





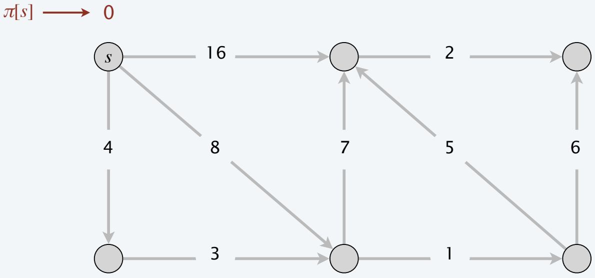
4. Greedy Algorithms II

- Dijkstra's algorithm demo
- Dijkstra's algorithm demo (efficient implementation)

Dijkstra's algorithm demo (efficient implementation)

Initialization.

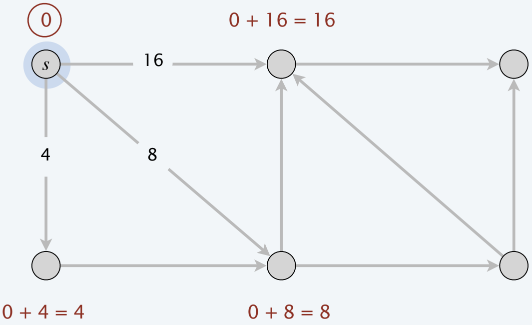
- For all $v \neq s$: $\pi[v] \leftarrow \infty$.
- For all $v \neq s$: $pred[v] \leftarrow null$.
- $S \leftarrow \emptyset$ and $\pi[s] \leftarrow 0$.



Dijkstra's algorithm demo (efficient implementation)

Basic step. Choose unexplored node $u \notin S$ with minimum $\pi[u]$.

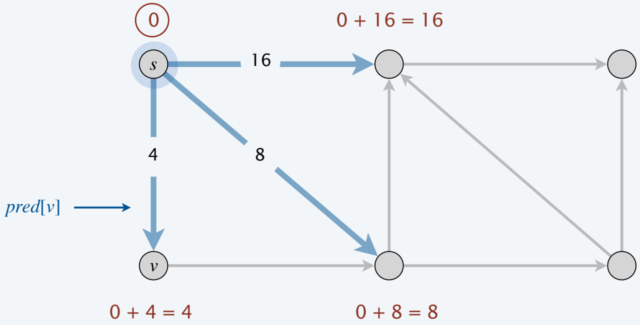
- Add u to S.
- For each edge $e = (u, v)$ leaving u, if $\pi[v] > \pi[u] + \ell_e$ then:
  - $\pi[v] \leftarrow \pi[u] + \ell_e$
  - $pred[v] \leftarrow e$



Dijkstra's algorithm demo (efficient implementation)

Basic step. Choose unexplored node $u \notin S$ with minimum $\pi[u]$.

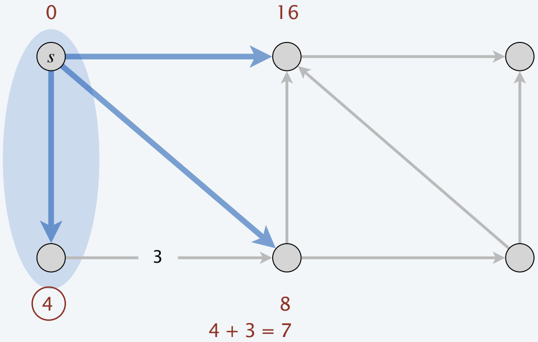
- Add u to S.
- For each edge $e = (u, v)$ leaving u, if $\pi[v] > \pi[u] + \ell_e$ then:
  - $\pi[v] \leftarrow \pi[u] + \ell_e$
  - $pred[v] \leftarrow e$



Dijkstra's algorithm demo (efficient implementation)

Basic step. Choose unexplored node $u \notin S$ with minimum $\pi[u]$.

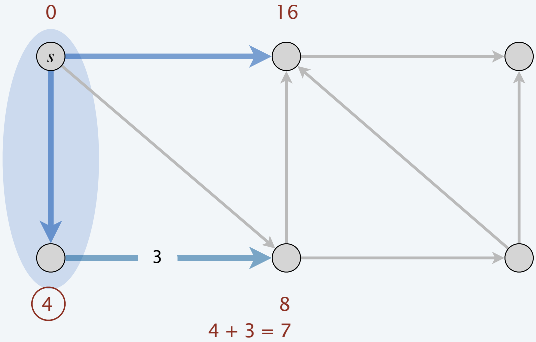
- Add u to S.
- For each edge $e = (u, v)$ leaving u, if $\pi[v] > \pi[u] + \ell_e$ then:
  - $\pi[v] \leftarrow \pi[u] + \ell_e$
  - $pred[v] \leftarrow e$



Dijkstra's algorithm demo (efficient implementation)

Basic step. Choose unexplored node $u \notin S$ with minimum $\pi[u]$.

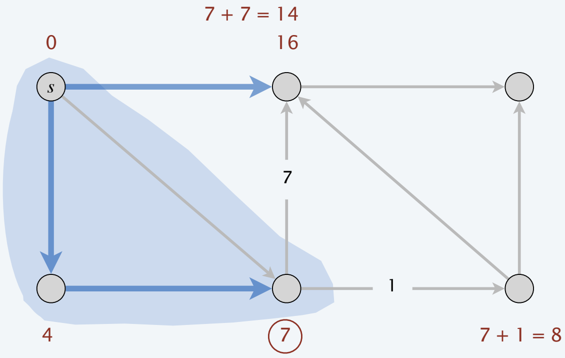
- Add u to S.
- For each edge $e = (u, v)$ leaving u, if $\pi[v] > \pi[u] + \ell_e$ then:
  - $\pi[v] \leftarrow \pi[u] + \ell_e$
  - $pred[v] \leftarrow e$



Dijkstra's algorithm demo (efficient implementation)

Basic step. Choose unexplored node $u \notin S$ with minimum $\pi[u]$.

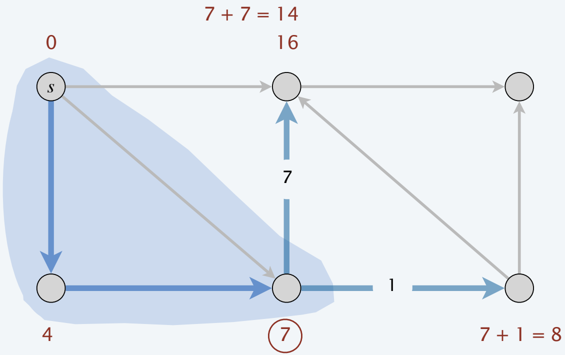
- Add u to S.
- For each edge $e = (u, v)$ leaving u, if $\pi[v] > \pi[u] + \ell_e$ then:
  - $\pi[v] \leftarrow \pi[u] + \ell_e$
  - $pred[v] \leftarrow e$



Dijkstra's algorithm demo (efficient implementation)

Basic step. Choose unexplored node $u \notin S$ with minimum $\pi[u]$.

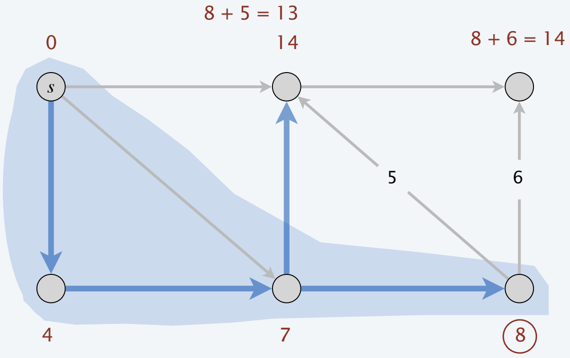
- Add u to S.
- For each edge $e = (u, v)$ leaving u, if $\pi[v] > \pi[u] + \ell_e$ then:
  - $\pi[v] \leftarrow \pi[u] + \ell_e$
  - $pred[v] \leftarrow e$



Dijkstra's algorithm demo (efficient implementation)

Basic step. Choose unexplored node $u \notin S$ with minimum $\pi[u]$.

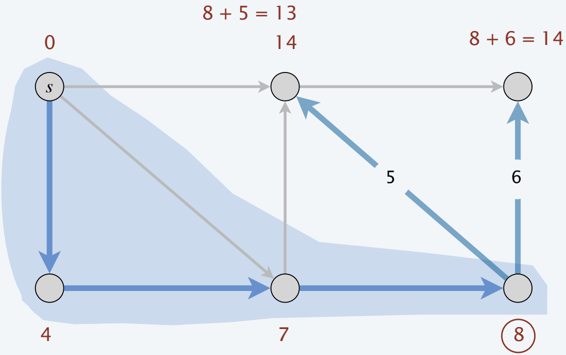
- Add u to S.
- For each edge $e = (u, v)$ leaving u, if $\pi[v] > \pi[u] + \ell_e$ then:
  - $\pi[v] \leftarrow \pi[u] + \ell_e$
  - $pred[v] \leftarrow e$



Dijkstra's algorithm demo (efficient implementation)

Basic step. Choose unexplored node $u \notin S$ with minimum $\pi[u]$.

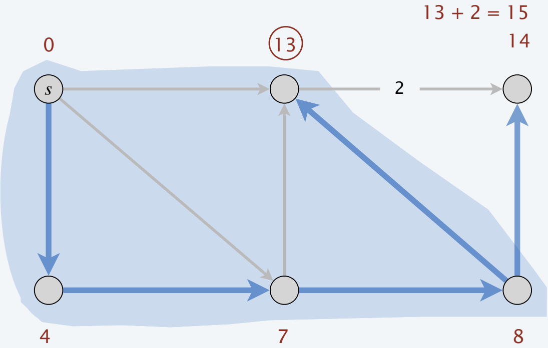
- Add u to S.
- For each edge $e = (u, v)$ leaving u, if $\pi[v] > \pi[u] + \ell_e$ then:
  - $\pi[v] \leftarrow \pi[u] + \ell_e$
  - $pred[v] \leftarrow e$



Dijkstra's algorithm demo (efficient implementation)

Basic step. Choose unexplored node $u \notin S$ with minimum $\pi[u]$.

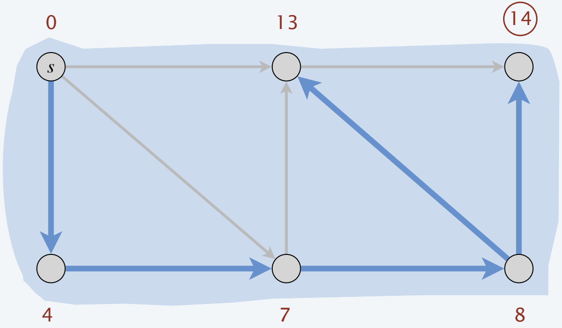
- Add u to S.
- For each edge $e = (u, v)$ leaving u, if $\pi[v] > \pi[u] + \ell_e$ then:
  - $\pi[v] \leftarrow \pi[u] + \ell_e$
  - $pred[v] \leftarrow e$



Dijkstra's algorithm demo (efficient implementation)

Basic step. Choose unexplored node $u \notin S$ with minimum $\pi[u]$.

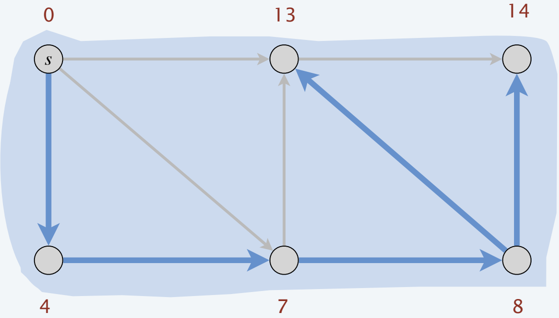
- Add u to S.
- For each edge $e = (u, v)$ leaving u, if $\pi[v] > \pi[u] + \ell_e$ then:
  - $\pi[v] \leftarrow \pi[u] + \ell_e$
  - $pred[v] \leftarrow e$



Dijkstra's algorithm demo (efficient implementation)

Basic step. Choose unexplored node $u \notin S$ with minimum $\pi[u]$.

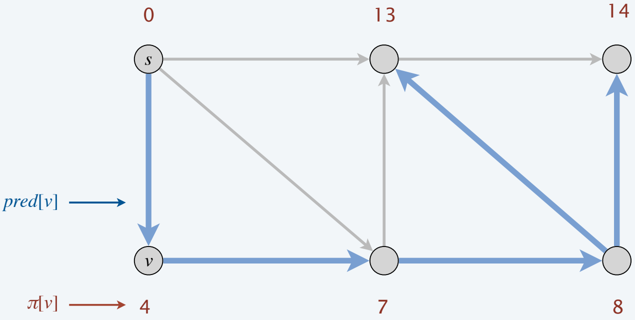
- Add u to S.
- For each edge $e = (u, v)$ leaving u, if $\pi[v] > \pi[u] + \ell_e$ then:
  - $\pi[v] \leftarrow \pi[u] + \ell_e$
  - $pred[v] \leftarrow e$

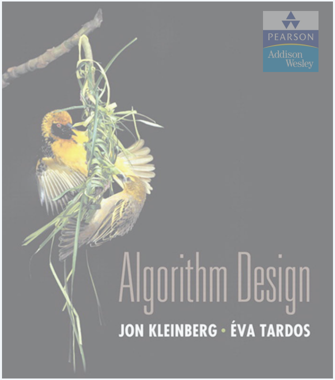


Dijkstra's algorithm demo (efficient implementation)

Termination.

- $\pi[v]$ = length of a shortest $s \rightarrow v$ path.
- $pred[v]$ = last edge on a shortest $s \rightarrow v$ path.





SECTION 7.12

7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Baseball elimination

TUESDAY, SEPTEMBER 10, 1996

San Francisco Chronicle

The Gate
Sports Online
▶ http://www.sfgate.com

SPORTING G

49ers, Young Get Big Brea

Quarterback m

By Gary Swan
Chronicle Staff Writer

The bye week has come at a perfect time for the 49ers and quarterback Steve Young. If they had a game next Sunday, there's a good chance Young would not play.

But the pulled groin muscle on his up-

Giants Officially Leave the NL West Race

By Nancy Gay
Chronicle Staff Writer

With the smack of another National League West bat 500 miles away, the Giants' run at the division title ended last night, just as they were handing the visiting St. Louis Cardinals an even bigger lead in the NL Central.

CARDINALS 6
GIANTS 2

In San Diego, Greg Vaughn's three-run homer in the eighth pushed the Padres over the Pirates and officially shoved the rest of the Giants' season into the background. On the heels of their tedious 6-2 loss before an announced crowd of 10,307 at Candlestick Park, the Giants fell 19½ games off the lead.

As it is, the worst the Padres (80-65) can finish is 80-82. The Giants have fallen to 59-83 with 20 games left; they cannot win 80 games. Coming off a miserable 2-8 mark on a three-city road trip that saw their road record drop to 27-47, the Giants were hoping to get off on the right foot in their longest homestand of the year (15 games, 14 days).

Financing in Place For Giants' New Stadium
SEE PAGE B1, MAIN NEWS

"Where we are, you're going to be eliminated sooner or later," Baker said quietly. "But it doesn't alter the fact that we've still got to play ball. You've still got to play hard, the fans come out to watch you play. You've got to play for the fact of loving to play, no matter where you are in the standings.

"You've got to play the role of spoiler, to not make it easier on

GIANTS: *Page D5 Col. 3*

Baseball elimination problem

Q. Which teams have a chance of finishing the season with the most wins?

i	team	wins	losses	to play	ATL	PHI	NYM	MON
0	Atlanta	83	71	8	–	1	6	1
1	Philly	80	79	3	1	–	0	2
2	New York	78	78	6	6	0	–	0
3	Montreal	77	82	3	1	2	0	–

Montreal is mathematically eliminated.

- Montreal finishes with ≤ 80 wins.
- Atlanta already has 83 wins.

Remark. This is the only reason sports writers appear to be aware of — conditions are sufficient but not necessary!

Baseball elimination problem

Q. Which teams have a chance of finishing the season with the most wins?

i	team	wins	losses	to play	ATL	PHI	NYM	MON
0	Atlanta	83	71	8	–	1	6	1
1	Philly	80	79	3	1	–	0	2
2	New York	78	78	6	6	0	–	0
3	Montreal	77	82	3	1	2	0	–

Philadelphia is mathematically eliminated.

- Philadelphia finishes with ≤ 83 wins.
- Either New York or Atlanta will finish with ≥ 84 wins.

Observation. Answer depends not only on how many games already won and left to play, but on whom they're against.

Baseball elimination problem

Current standings.

- Set of teams S.
- Distinguished team $z \in S$.
- Team x has won w_x games already.
- Teams x and y play each other r_{xy} additional times.

Baseball elimination problem. Given the current standings, is there any outcome of the remaining games in which team z finishes with the most (or tied for the most) wins?

SIAM Review
Vol. 8, No. 3, July, 1966

POSSIBLE WINNERS IN PARTIALLY COMPLETED TOURNAMENTS*

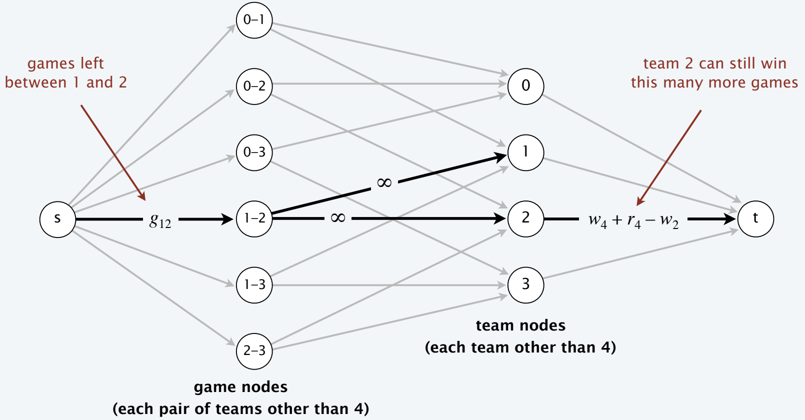
BENJAMIN L. SCHWARTZ†

1. Introduction. In this paper, we shall investigate certain questions in tournament scheduling. For definiteness, we shall use the terminology of baseball. We shall be concerned with the categorization of teams into three classes during the closing days of the season. A team may be definitely eliminated from pennant possibility; it may be in contention, or it may have clinched the championship. It will be our convention that a team that can possibly tie for the pennant is considered still in contention. In this paper necessary and sufficient conditions are developed to classify any team properly into the appropriate category.

Baseball elimination problem: max-flow formulation

Can team 4 finish with most wins?

- Assume team 4 wins all remaining games $\Rightarrow$ $w_4 + r_4$ wins.
- Divvy remaining games so that all teams have $\leq w_4 + r_4$ wins.

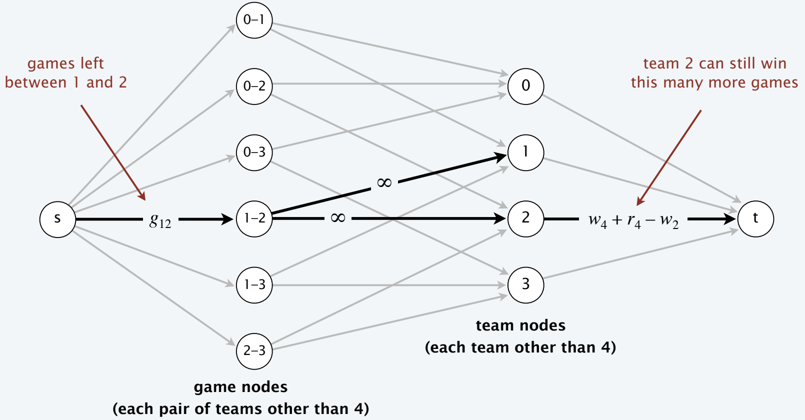


Baseball elimination problem: max-flow formulation

Theorem. Team 4 not eliminated iff max flow saturates all edges leaving s.

Pf.

- Integrality theorem $\Rightarrow$ each remaining game between x and y added to number of wins for team x or team y.
- Capacity on (x, t) edges ensure no team wins too many games. ▪



Baseball elimination: explanation for sports writers

Q. Which teams have a chance of finishing the season with the most wins?

i	team	wins	losses	to play	NYY	BAL	BOS	TOR	DET
0	New York	75	59	28	–	3	8	7	3
1	Baltimore	71	63	28	3	–	2	7	4
2	Boston	69	66	27	8	2	–	0	0
3	Toronto	63	72	27	7	7	0	–	0
4	Detroit	49	86	27	3	4	0	0	–

AL East (August 30, 1996)

Detroit is mathematically eliminated.

- Detroit finishes with ≤ 76 wins.
- Wins for $R = \{ \text{NYY}, \text{BAL}, \text{BOS}, \text{TOR} \} = 278$.
- Remaining games among $\{ \text{NYY}, \text{BAL}, \text{BOS}, \text{TOR} \} = 3 + 8 + 7 + 2 + 7 = 27$.
- Average team in R wins $305/4 = 76.25$ games.

Baseball elimination: explanation for sports writers

Certificate of elimination.

$$T \subseteq S, \;\; w(T) := \overbrace{\sum_{i \in T} w_i}^{\#\text{ wins}}, \quad g(T) := \overbrace{\sum_{\{x,y\} \subseteq T} g_{xy}}^{\#\text{ remaining games}},$$

Theorem. [Hoffman–Rivlin 1967] Team z is eliminated iff there exists a subset T^* such that

$$w_z + g_z \; < \; \frac{w(T^*) + g(T^*)}{|T^*|}$$

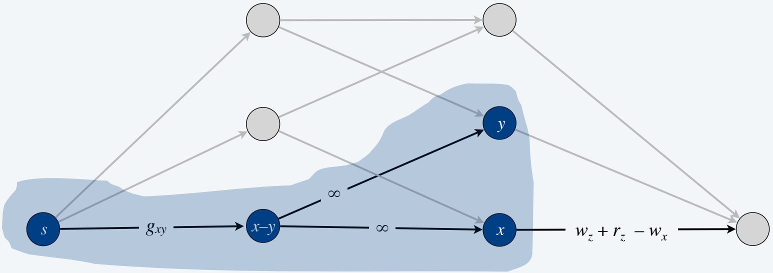
Pf. $\Leftarrow$

- Suppose there exists $T^* \subseteq S$ such that $w_z + g_z \; < \; \dfrac{w(T^*) + g(T^*)}{|T^*|}$.
- Then, the teams in T^* win at least $(w(T^*) + g(T^*)) \,/\, |T^*|$ games on average.
- This exceeds the maximum number that team z can win. ▪

Baseball elimination: explanation for sports writers

Pf. $\Rightarrow$

- Use max-flow formulation, and consider min cut (A, B).
- Let $T^* =$ team nodes on source side A of min cut.
- Observe that game node $x\text{–}y \in A$ iff both $x \in T^*$ and $y \in T^*$.
  - infinite capacity edges ensure if $x\text{–}y \in A$, then both $x \in A$ and $y \in A$
  - if $x \in A$ and $y \in A$ but $x\text{–}y \notin A$, then adding $x\text{–}y$ to A decreases the capacity of the cut by g_{xy}



Baseball elimination: explanation for sports writers

Pf. $\Rightarrow$

- Use max-flow formulation, and consider min cut (A, B).
- Let T^* = team nodes on source side A of min cut.
- Observe that game node $x\text{–}y \in A$ iff both $x \in T^*$ and $y \in T^*$.
- Since team z is eliminated, by max-flow min-cut theorem,

$$
\begin{aligned}
g(S - \{z\}) \quad &> \quad cap(A, B) \\
&= \quad \overbrace{g(S - \{z\}) - g(T^*)}^{\text{capacity of game edges leaving s}} \quad + \quad \overbrace{\sum_{x \in T^*} (w_z + g_z - w_x)}^{\text{capacity of team edges entering t}} \\
&= \quad g(S - \{z\}) - g(T^*) \quad - \quad w(T^*) \; + \; |T^*| (w_z + g_z)
\end{aligned}
$$

- Rearranging terms: $\quad w_z + g_z \; < \; \dfrac{w(T^*) + g(T^*)}{|T^*|}$ ▪

The problem

We are now able to cope with costs and capacities at the same time. Given

- a set of nodes,
- a set of links connecting them,
- a connection request between two nodes of the network,

(that is, an existing network).
I want to

- decide which links to use in the connection (route)
- maximizing the quality of service (e.g. minimizing delay time)

Assumptions

Some assumptions:

1. no *costs* involved: packets can also follow non-shortest paths,
2. $\rightarrow$ *cost* matters!
3. the capacity of each link is enough for the whole connection request.
4. $\rightarrow$ the capacity of links may not be enough for the whole connection request.

Assumptions

Some assumptions:

1. no *costs* involved: packets can also follow non-shortest paths,
2. $\rightarrow$ *cost* matters!
3. the capacity of each link is enough for the whole connection request.
4. $\rightarrow$ the capacity of links may not be enough for the whole connection request.

Assumptions

Some assumptions:

1. no *costs* involved: packets can also follow non-shortest paths,
2. $\rightarrow$ *cost* matters!
3. the capacity of each link is enough for the whole connection request.
4. $\rightarrow$ the capacity of links may not be enough for the whole connection request.

Assumptions

Some assumptions:

1. no *costs* involved: packets can also follow non-shortest paths,
2. $\rightarrow$ *cost* matters!
3. the capacity of each link is enough for the whole connection request.
4. $\rightarrow$ the capacity of links may not be enough for the whole connection request.

Recognizing a known problem ...

We observe

- when capacities are always large enough: Shortest Path Problems,
- when costs are not involved: Max Flow Problems.

we are facing a *Min Cost Flow (MCF) problem*.

N.B. Min Cost Flows generalize both Shortest Path and Max Flow problems.

Recognizing a known problem ...

We observe

- when capacities are always large enough: Shortest Path Problems,
- when costs are not involved: Max Flow Problems.

we are facing a *Min Cost Flow (MCF) problem*.
N.B. Min Cost Flows generalize both Shortest Path and Max Flow problems.

Graph model

Given a network, build a *directed* graph $G = (V, A)$ having

- one vertex $i \in V$ for each node of the network
- one arc $a \in A \subseteq V \times V$ for each link of the network
- capacities $u_{(i,j)}$ on each arc $(i, j) \in A$
- costs $c_{(i,j)}$ on each arc $(i, j) \in A$
- flow consumption b_i for each node $i \in V$

Mathematical Programming model

Let $x_{(i,j)}$ be *decision variables* representing the *amount of flow* sent on arc (i,j). Let v represent the total cost of routing packets in the network.

$$\begin{aligned} \text{minimize } v = & \sum_{(i,j)\in A} c_{(i,j)} x_{(i,j)} \\ \text{subject to } & \sum_{j\in V} x_{(i,j)} = \sum_{k\in V} x_{(k,i)} + b_i \qquad \forall i \in V, i \neq s,t \\ & 0 \leq x_{(i,j)} \leq u_{(i,j)} \qquad \forall (i,j) \in A \end{aligned}$$

The Minimum Cost Flow Problem

u_{ij} = capacity of arc (i,j).

c_{ij} = unit cost of shipping flow from node i to node j on (i,j).

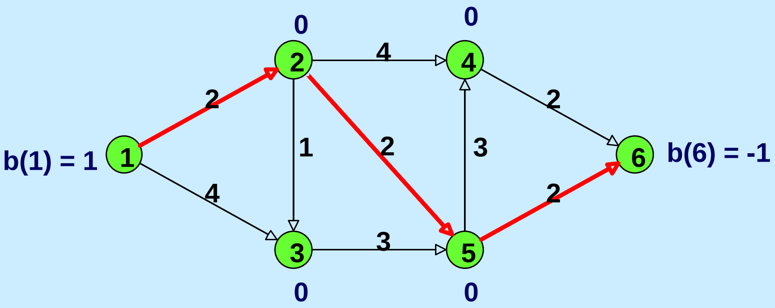
x_{ij} = amount shipped on arc (i,j)

Minimize $\sum_{(i,j)\in A} c_{ij}x_{ij}$

$$\sum_j x_{ij} \quad - \quad \sum_k x_{ki} = b_i \qquad \text{for all } i \in N.$$

$$\text{and } \; 0 \le x_{ij} \le u_{ij} \quad \text{for all } (i,j) \in A.$$

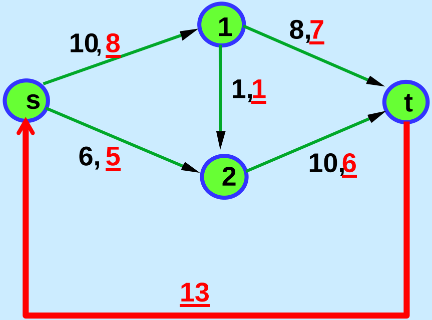
Find the shortest path from node 1 to node 6



The optimal flow is to send one unit of flow along 1-2-5-6.

This transformation works so long as there are no negative cost cycles in G.
(What if there are negative cost cycles?)

Find the Maximum Flow from s to t



b(i) = 0 for all i;

add arc (t,s) with a cost of -1 and large capacity.

The cost of every other arc is 0.

The optimal solution in the corresponding minimum cost flow problem will send as much flow in (t,s) as possible.

Transshipment Problems

Plants with given production capabilities for a product.

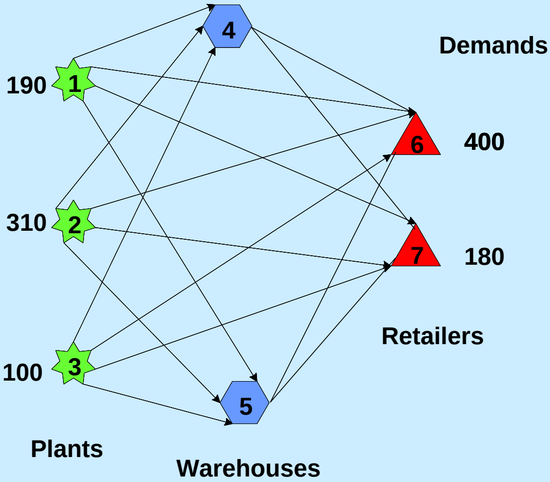
One can ship directly from the plants to retailers, or from plants to warehouses, and then from warehouses to retailers.

There is a given demand for each retailer.

Costs of shipment are given.

What is the minimum cost method for satisfying demands?

A Network Representation



The Caterer Problem

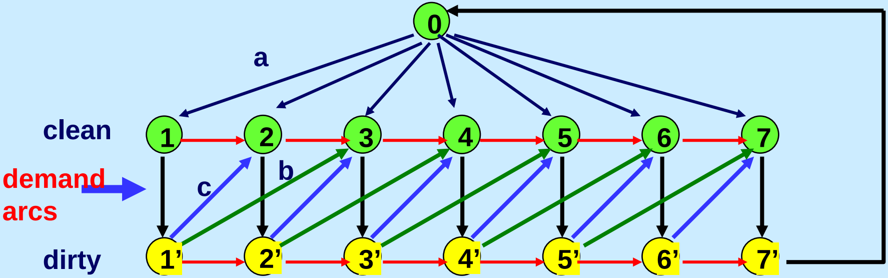
Demand for d_i napkins on day i for i = 1 to 7 (so, $j \in [1..7]$).

Cost of new napkins: *a* cents each,

2-day laundry: b cents per napkin

1-day laundry: c cents per napkin.

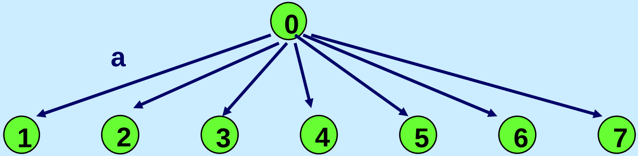
Minimize the cost of meeting demand.



Purchase arcs

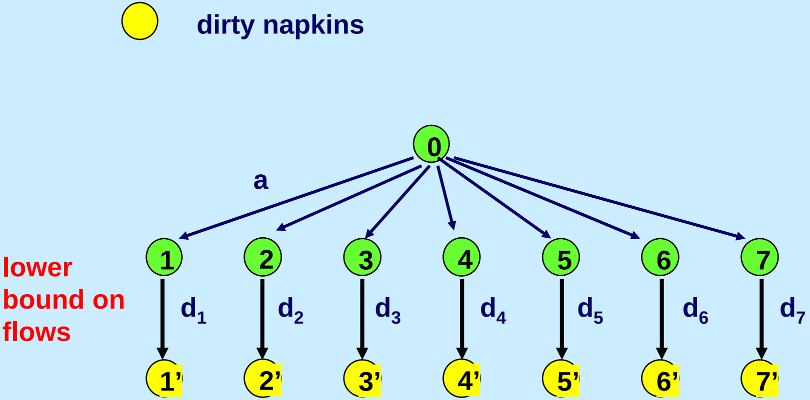
In any period of the seven periods, one can purchase napkins, at a cost of a cents per napkin.

clean napkins



Demand Arcs

You must use d_i napkins on day i



The rest of the arcs

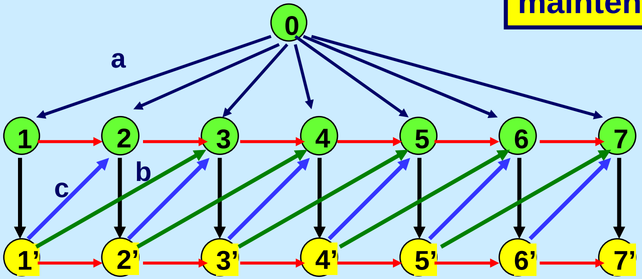
You may launder napkins in 2 days at b cents each

You may launder napkins in 1 day at c cents each

You may store clean napkins for free

You may store dirty napkins for free

Application to airplane maintenance.



Some Assumptions

1. **All data is integral. (Needed for some proofs, and some running time analysis).**

2. **The network is directed.**

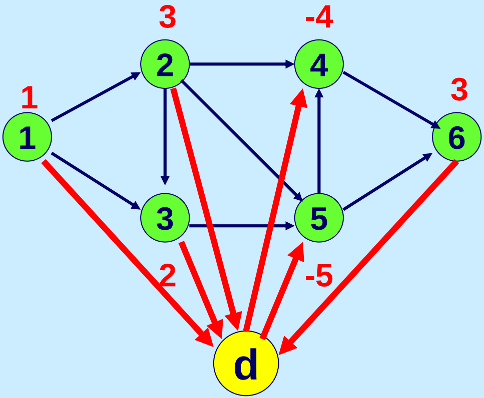
3. $\sum_{i=1 \text{ to } n} b(i) = 0$.
   (Otherwise, there cannot be a feasible solution)

4. **There is a feasible solution (see next slide)**

Artificial Solutions

To create a feasible solution, add a dummy node d.

Add an arc from d to each demand node, each with a large cost M, and large capacity.



Add an arc to d from each supply node, each with a large cost M, and a large capacity.

In an optimal solution, arcs with large cost will have a flow of 0.

Flow decomposition theorem

Theorem

(a) Every nonnegative (arc) flow x can be represented as a flow on paths and cycles (though not necessarily uniquely) with the following two properties:

- *every directed path with positive flow connects a deficit node to an excess node*
- *at most $n + m$ paths and cycles have nonzero flow; out of these, at most m cycles have nonzero flow.*

(b) Conversely, every path and cycle flow has a unique representation as nonnegative arc flows.

Proof: on the blackboard

Extendend model

Let P be the set of all s-t paths in G, c^p be the cost of each $p \in P$, $\bar{x}^p_{(i,j)}$ be 1 if $(i,j) \in p$, 0 otherwise. Let x^p be *decision variables* representing the *amount of flow* sent on path p. Let v represent the total cost of network flows.

$$\begin{aligned}
\text{minimize } v = & \sum_{p \in P} c_p x_p \\
\text{subject to } & \sum_{p \in P} \sum_{j \in V} \bar{x}^p_{(i,j)} x^p - \sum_{p \in P} \sum_{k \in V} \bar{x}_{(k,i)} x^p = b_i && \forall i \in V, i \neq s,t \\
& \sum_{p \in P} \bar{x}^p_{(i,j)} x^p \leq u_{(i,j)} && \forall (i,j) \in A \\
& 0 \leq x^p && \forall p \in P
\end{aligned}$$

A huge LP!

The problem

Let us generalize it a bit. Given

- a set of nodes,
- a set of links connecting them,
- a **set** of connection requests, between **pairs of nodes** of the network,

that is, an existing network, with realistic traffic.
I want to

- decide which links to use in the connections (route)
- maximizing the quality of service (e.g. minimizing delay time)

Assumptions

Some assumptions:

1. cost matters,
2. the capacity of links may not be enough for all connection requests,
3. different connections can be routed on the same links ...
4. ... provided the capacity of each link is enough.

Recognizing a known problem ...

We observe

- when a single connection request is made on the network, the problem is to compute a Min Cost Flow ...

we are facing a *multicommodity Min Cost Flow (MCF) problem*.

Application areas

Type of Network	Nodes	Arcs	Flow
Communic. Networks	O-D pairs for messages	Transmission lines	message routing
Computer Networks	storage dev. or computers	Transmission lines	data, messages
Railway Networks	yard and junction pts.	Tracks	Trains
Distribution Networks	plants warehouses,...	highways railway tracks etc.	trucks, trains, etc

Graph model

Given a network, build a *directed* graph $G = (V, A)$ having

- one vertex $i \in V$ for each node of the network,
- one arc $a \in A \subseteq V \times V$ for each link of the network,
- capacities $u_{(i,j)}$ on each arc $(i, j) \in A$.

Then, consider the set K of connection requests (commodities), and enrich the graph with

- costs $c^k_{(i,j)}$ on each arc $(i, j) \in A$ for each commodity $k \in K$,
- flow excess b^k_i for each node $i \in V$ and for each commodity $k \in K$.

Assumptions

We assume that

- **Homogeneous commodities:** each unit of flow of uses one unit of capacity on each arc, independently of k,
- **No congestion:** cost is linear in the amount of flow on each arc (until capacity limit is reached),
- **Fractional flows:** no integrality condition is imposed on flows.

WLOG we assume also that

- $b_i^k > 0$ for a unique $i \in V$ (origin of commodity $k \rightarrow s_k$),
- $b_i^k < 0$ for a unique $i \in V$ (destination of commodity $k \rightarrow t_k$).

We search for k min cost flows on the network, one for each commodity.

Mathematical Programming model

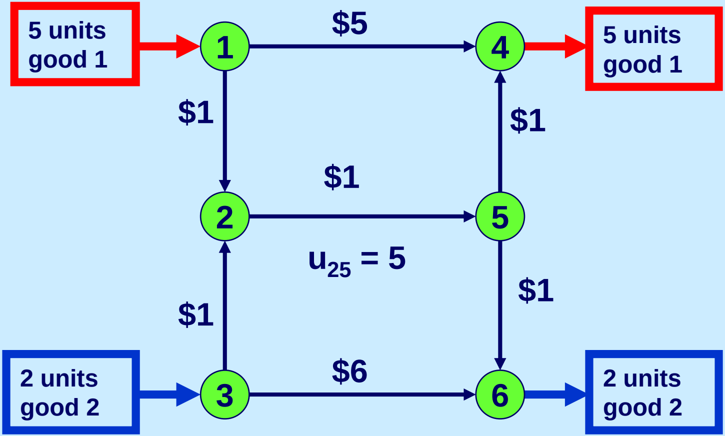
Let $x^k_{(i,j)}$ be *decision variables* representing the *amount of flow for commodity* k sent on arc (i,j). Let v represent the total cost of routing packets in the network.

$$
\begin{aligned}
\text{minimize } v = & \sum_{(i,j)\in A} \sum_{k \in K} c^k_{(i,j)} x^k_{(i,j)} & \\
\text{subject to } & \sum_{j \in V} x^k_{(i,j)} = \sum_{j \in V} x^k_{(j,i)} + b^k_i & \forall i \in V; \quad \forall k \in K \\
& \sum_{k \in K} x^k_{(i,j)} \leq u_{(i,j)} & \forall (i,j) \in A \\
& 0 \leq x^k_{(i,j)} \leq u_{(i,j)} & \forall (i,j) \in A; \quad \forall k \in K
\end{aligned}
$$

An example of multicommodity flow

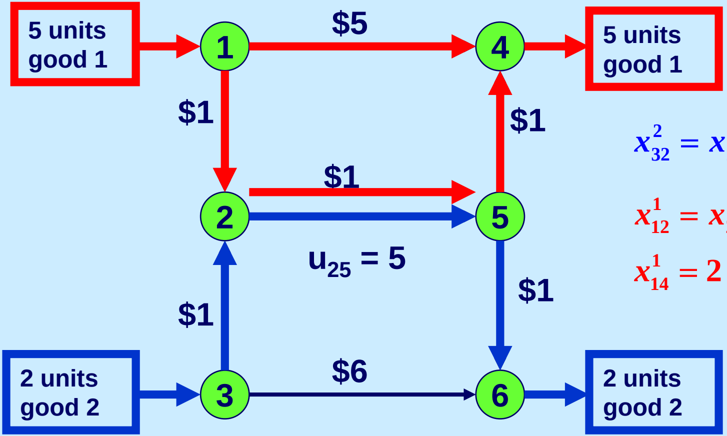
See Orlin's slides 22,3-4

A Linear Multicommodity Flow Problem



Quick exercise: determine the optimal multicommodity flow.

A Linear Multicommodity Flow Problem



$$x^2_{32} = x^2_{25} = x^2_{56} = 2$$

$$x^1_{12} = x^1_{25} = x^1_{54} = 3$$

$$x^1_{14} = 2$$

Discrete and fractional flows

See Orlin's slides 22,8-10

On Fractional Flows

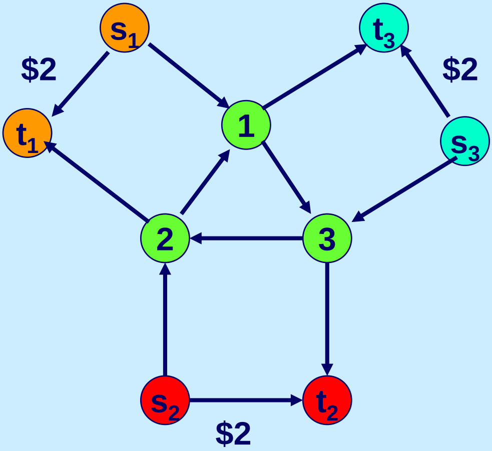
- In general, multicommodity flow problems have fractional flows, even if all data is integral.
- The integer multicommodity flow problem is difficult to solve to optimality.

A fractional multicommodity flow

$u_{ij} = 1$ for all arcs

$c_{ij} = 0$ except as listed.

1 unit of flow must be sent from s_i to t_i for i = 1, 2, 3.

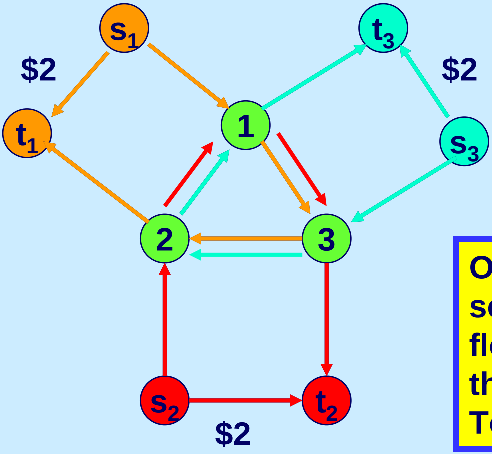


A fractional multicommodity flow

u_{ij} = 1 for all arcs

c_{ij} = 0 except as listed.

1 unit of flow must be sent from s_i to t_i for i = 1, 2, 3.



Optimal solution: send ½ unit of flow in each of these 15 arcs. Total cost = $3.

Ways of solving MMCF

There are many ways of solving MMCF:

- price (cost) directed decompositions,
- resource (capacity) directed decompositions,
- simplex based approaches.

Price directed decompositions

Idea behind price directed decompositions:

- modify costs on arcs ...
- ... such that solving k MCF independently gives a full MMCF solution ...
- ... that automatically satisfies capacity constraints.

We will see:

- Lagrangean relaxation,
- column generation.

Optimality conditions: partial dualization

Theorem: The multicommodity flow (x_{ij}^k) is *optimal* if there exist non-negative prices (w_{ij}) on the arcs, so that the following is true:

- if $w_{ij} > 0$ then $\sum_{k \in K} x_{ij}^k = u_{ij}$,
- each flow k is (independently) optimal for commodity k if each cost c_{ij}^k is replaced by

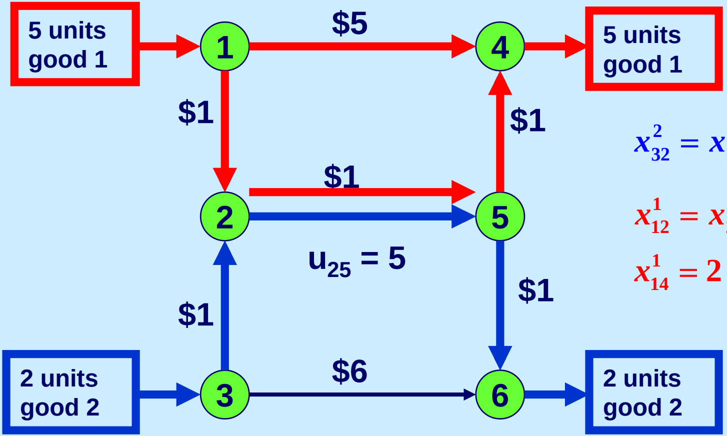
$$c_{ij}^{w,k} = c_{ij}^k + w_{ij}$$

.

Recall: flow k is optimal for commodity k if there is no negative cost cycle in the residual network for commodity k.

See Orlin's slides 22,14-16

A Linear Multicommodity Flow Problem



$x^2_{32} = x^2_{25} = x^2_{56} = 2$

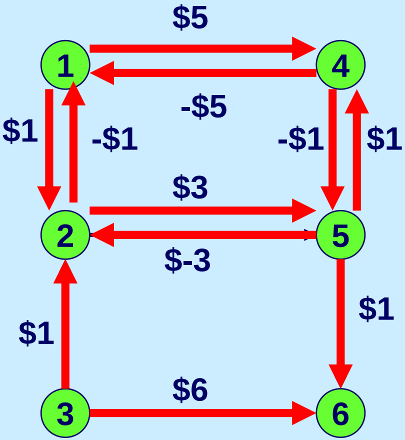
$x^1_{12} = x^1_{25} = x^1_{54} = 3$

$x^1_{14} = 2$

Set $w_{2,5} = 2$

Create the residual networks

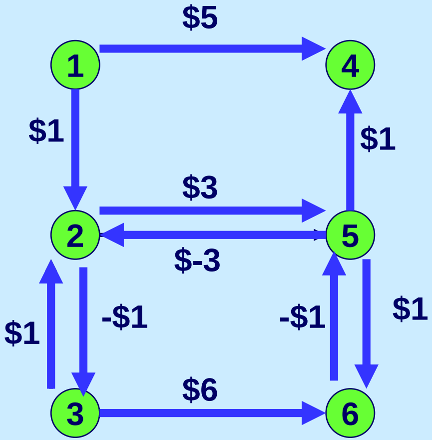
The residual network for commodity 1



Set $w_{2,5}$ = $2

There is no negative cost cycle.

The residual network for commodity 2



Set $w_{2,5}$ = $2

There is no negative cost cycle.

Lagrangean algorithm for MMCF

Idea: update w and solve MCF until the partial dualization conditions are satisfied.

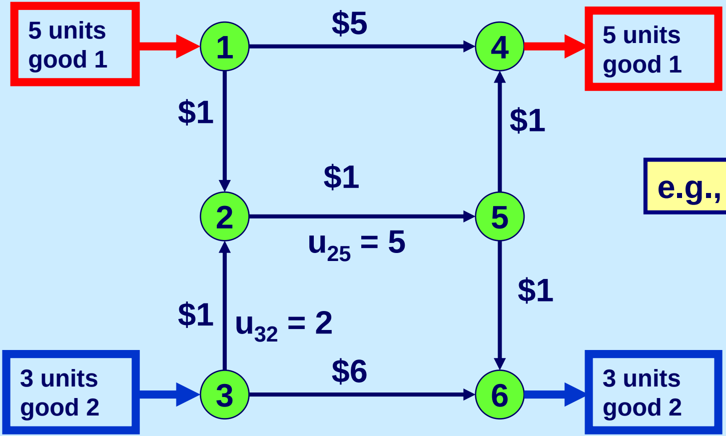
Lagrangean algorithm for MMCF

BEGIN
$x := 0; w := 0$
$\theta := 1$
while **partial dualization optimality conditions** are not satisfied
begin
    set $c_{ij}^{w,k} := c_{ij}^k + w_{ij}$ for each $k \in K$ and for each $(i,j) \in A$
    for each $k \in K$
        build a residual network $G^k(x)$
        solve a MCF problem on $G^k(x)$ using costs $c_{ij}^{w,k}$
        obtain a flow x_{ij}^k
    update prices w: for each $(i,j) \in A$
        $w_{ij} := \max\{0, w_{ij} + \theta \cdot (\sum_{k \in K} x_{ij}^k - u_{ij})\}$
    reduce θ
end
END

Solving MMCF by Lagrangean relaxation

See Orlin's slides 22,21-28

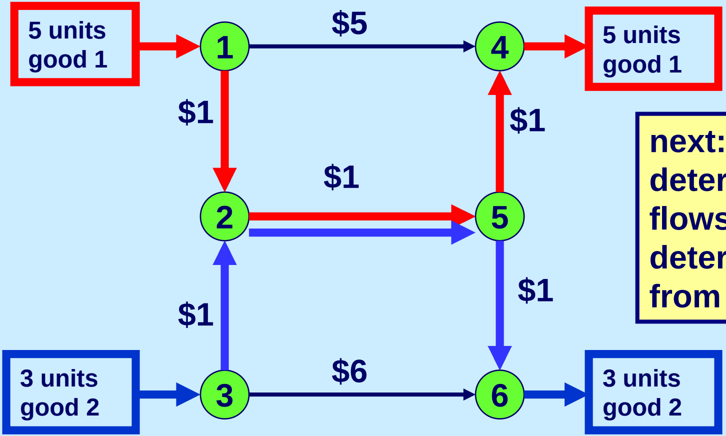
Subgradient Optimization for solving the Lagrangian Multiplier Problem



e.g., set $w^0 = 0$.

Choose an initial value w^0 of the "tolls" w, and find the optimal solution for L(w).

Subgradient Optimization for solving the Lagrangian Multiplier Problem



next: determine the flows, and then determine w^1 from w^0

The flow on (2,5) = 8 > u_{25} = 5.

The flow on (3,2) = 3 > u_{32} = 2.

Choosing a search direction

$$r^+ = \max(0, r)$$

$$y_{ij} = \sum_k x_{ij}^k \quad = \text{flow in arc (i,j)}$$

$$w_{ij}^{q+1} = [w_{ij}^q + \theta_q (y_{ij} - u_{ij})]^+$$

$(y\text{-}u)^+$ is called the ***search direction***.

$$w_{25}^1 = [w_{25}^0 + \theta_0 (8 - 5)]^+ = 3\theta_0$$

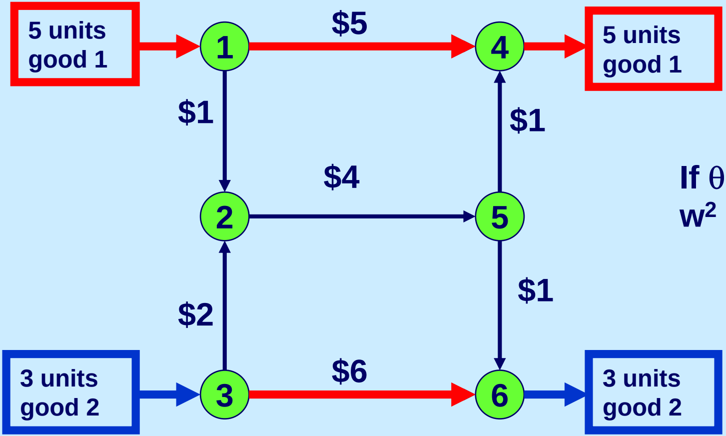
θ_q is called the ***step size***.

$$w_{32}^1 = [w_{32}^0 + \theta_0 (3 - 2)]^+ = \theta_0$$

So, if we choose $\theta_0 = 1$, then $w_{25}^1 = 3$ and $w_{32}^1 = 1$

Then solve $L(w^1)$.

Solving $L(w^1)$



If $\theta^1 = 1$, then $w^2 = 0$.

$$w_{25}^2 = [w_{25}^1 + \theta_1(0-5)]^+ = [3 - 5\theta_1]^+$$

$$w_{32}^2 = [w_{32}^1 + \theta_1(0-2)]^+ = [1 - 2\theta_1]^+$$

Comments on the step size

- The search direction is a good search direction.
- But the step size must be chosen carefully.
- Too large a step size and the solution will oscillate and not converge
- Too small a step size and the solution will not converge to the optimum.

On choosing the step size

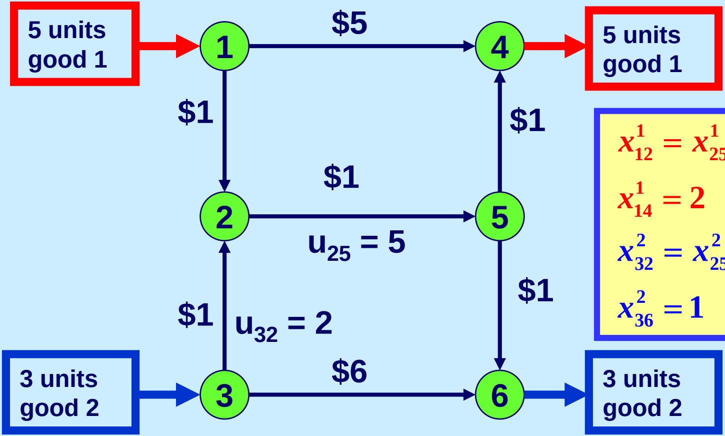
The step size θ_q should be chosen so that

$$\lim_{q\to\infty} \theta_q = 0 \quad \text{and} \quad \sum_{q=1}^{\infty} \theta_q = \infty \qquad (1)$$

e.g., take θ_q = 1/q.

***Theorem*. If the step size is chosen as on the previous slides, and if (θ_q) satisfies (1), then the w^q converges to the optimum for the Lagrangian dual.**

The optimal multipliers and flows.



$$x_{12}^1 = x_{25}^1 = x_{54}^1 = 3$$

$$x_{14}^1 = 2$$

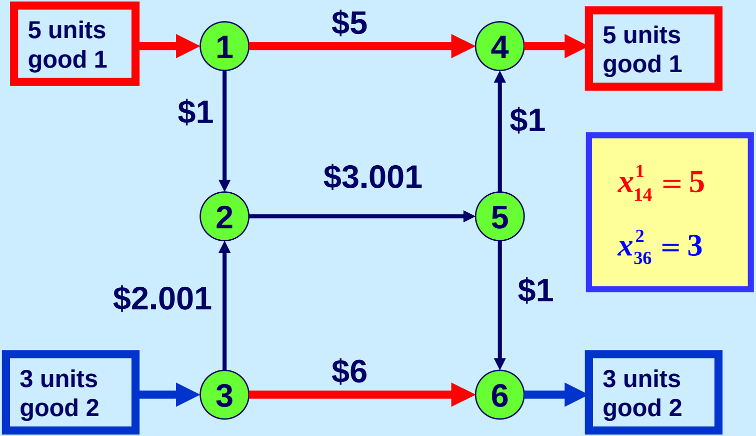
$$x_{32}^2 = x_{25}^2 = x_{56}^2 = 2$$

$$x_{36}^2 = 1$$

$$\lim_{q\to\infty} w_{32}^q = 1$$

$$\lim_{q\to\infty} w_{25}^q = 2$$

Suppose that $w_{32} = 1.001$ and $w_{25} = 2.001$



Conclusion: Near Optimal Multipliers do not always lead to near optimal (or even feasible) flows.

A path-based model

Idea: represent overall flow as sum of partial flows, each following a single path, and combine them in a feasible way.

A path-based model

$$\text{minimize } v = \sum_{k \in K} \sum_{P \in \mathcal{P}^k} c^P \cdot x^P$$

$$\text{subject to } \sum_{k \in K} \sum_{P \in \mathcal{P}^k} \bar{x}_{ij}^P \cdot x^P \le u_{ij} \qquad \forall (i,j) \in A$$

$$\sum_{P \in \mathcal{P}^k} x^P = b^k{}_{s_k} \qquad \forall k \in K$$

$$x^P \ge 0 \qquad \forall k \in K, \forall P \in \mathcal{P}^k$$

where:

- $\mathcal{P}^k$ is the set of *all* paths from s_k to t_k
- c^P is the cost of path P
- x^P is the amount of flow sent on path P
- $\bar{x}_{ij}^P = 1$ if path P includes arc (i,j), and $= 0$ otherwise

A path-based model

$$
\begin{aligned}
\text{minimize } v = \sum_{k\in K} \sum_{P\in\mathcal{P}^k} c^P \cdot x^P & \\
\text{subject to } \sum_{k\in K} \sum_{P\in\mathcal{P}^k} \bar{x}^P_{ij} \cdot x^P \le u_{ij} & \qquad \forall (i,j)\in A \\
\sum_{P\in\mathcal{P}^k} x^P = {b^k}_{s_k} & \qquad \forall k \in K \\
x^P \ge 0 & \qquad \forall k\in K, \forall P \in \mathcal{P}^k
\end{aligned}
$$

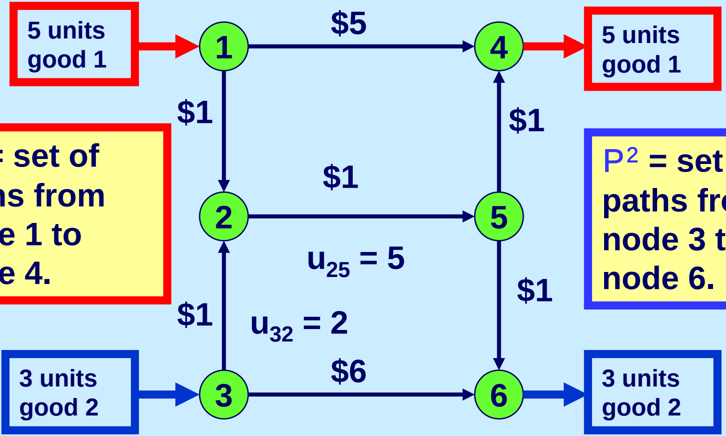
where:

- $\mathcal{P}^k$ is the set of *all* paths from s_k to t_k
- c^P is the cost of path P
- x^P is the amount of flow sent on path P
- $\bar{x}^P_{ij} = 1$ if path P includes arc (i,j), and $= 0$ otherwise

A path-based model

Orlin's slides 23,8-10

A Linear Multicommodity Flow Problem



P^1 = set of paths from node 1 to node 4.

P^2 = set of paths from node 3 to node 6.

$P^1 = \{1\text{-}4,\ 1\text{-}2\text{-}5\text{-}4\}$

$P^2 = \{3\text{-}6,\ 3\text{-}2\text{-}5\text{-}6\}$

A path based formulation

$f(P)$ = flow in path P

$c(P)$ = cost of path P

c(1-4)	=	5
c(1-2-5-4)	=	3
c(3-6)	=	6
c(3-2-5-6)	=	3

Minimize $5\, f(1\text{-}4) + 3\, f(1\text{-}2\text{-}5\text{-}4) + 6\, f(3\text{-}6) + 3\, f(3\text{-}2\text{-}5\text{-}6)$

subject to

$$f(1\text{-}4) + f(1\text{-}2\text{-}5\text{-}4) = 5$$

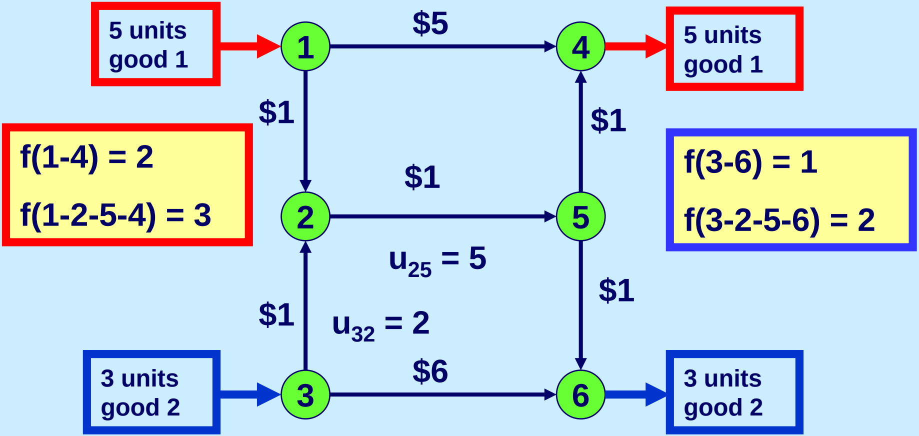
$$f(3\text{-}6) + f(3\text{-}2\text{-}5\text{-}6) = 3$$

$$f(1\text{-}2\text{-}5\text{-}4) + f(3\text{-}2\text{-}5\text{-}6) \leq u_{25} = 5$$

$$f(3\text{-}2\text{-}5\text{-}6) \leq u_{32} = 2$$

$$f(P) \geq 0 \text{ for all paths } P$$

Optimal solution for the path based version



The path based LP can be solved using the simplex method.

A path-based model

$$
\begin{aligned}
\text{minimize } v = & \sum_{k\in K} \sum_{P\in\mathcal{P}^k} c^P \cdot x^P \\
\text{subject to } & \sum_{k\in K} \sum_{P\in\mathcal{P}^k} \bar{x}_{ij}^P \cdot x^P \leq u_{ij} && \forall (i,j) \in A \\
& \sum_{P\in\mathcal{P}^k} x^P = b_{s_k}^k && \forall k \in K \\
& x^P \geq 0 && \forall k \in K, \forall P \in \mathcal{P}^k
\end{aligned}
$$

Is it possible to straightly optimize it?

$|\mathcal{P}^k|$ grows combinatorially with problem dimension: we need an iterative approach (column generation).

A path-based model

$$
\begin{aligned}
\text{minimize } v = \sum_{k \in K} \sum_{P \in \mathcal{P}^k} c^P \cdot x^P \\
\text{subject to } \sum_{k \in K} \sum_{P \in \mathcal{P}^k} \bar{x}_{ij}^P \cdot x^P \leq u_{ij} && \forall (i,j) \in A \\
\sum_{P \in \mathcal{P}^k} x^P = b_{s_k}^k && \forall k \in K \\
x^P \geq 0 && \forall k \in K, \forall P \in \mathcal{P}^k
\end{aligned}
$$

Is it possible to straightly optimize it?
$|\mathcal{P}^k|$ grows combinatorially with problem dimension: we need an iterative approach (column generation).

A path-based model

$$
\begin{aligned}
\text{minimize } v = \sum_{k \in K} \sum_{P \in \mathcal{P}^k} c^P \cdot x^P & \\
\text{subject to } \sum_{k \in K} \sum_{P \in \mathcal{P}^k} \bar{x}_{ij}^P \cdot x^P \leq u_{ij} & \qquad \forall (i,j) \in A \\
\sum_{P \in \mathcal{P}^k} x^P = b_{s_k}^k & \qquad \forall k \in K \\
x^P \geq 0 & \qquad \forall k \in K, \forall P \in \mathcal{P}^k
\end{aligned}
$$

Idea: in a good MMCF solution, only *very few good paths* are chosen.

We replace each $\mathcal{P}^k$ with a "well chosen" subset $\mathcal{S}^k \subset \mathcal{P}^k$

If we are lucky, all useful paths are in $\mathcal{S}^k$, otherwise we iteratively enlarge it.

A path-based model

$$
\begin{aligned}
\text{minimize } v = & \sum_{k\in K} \sum_{P\in\mathcal{S}^k} c^P \cdot x^P \\
\text{subject to } & \sum_{k\in K} \sum_{P\in\mathcal{S}^k} \bar{x}_{ij}^P \cdot x^P \le u_{ij} && \forall (i,j) \in A \\
& \sum_{P\in\mathcal{S}^k} x^P = b_{s_k}^k && \forall k \in K \\
& x^P \ge 0 && \forall k \in K, \forall P \in \mathcal{S}^k
\end{aligned}
$$

Idea: in a good MMCF solution, only *very few good paths* are chosen.
We replace each $\mathcal{P}^k$ with a "well chosen" subset $\mathcal{S}^k \subset \mathcal{P}^k$
If we are lucky, all useful paths are in $\mathcal{S}^k$, otherwise we iteratively enlarge it.

A path-based model

$$\begin{aligned}
\text{mn.} & \sum_{k\in K}\sum_{P\in\mathcal{P}^k} c^P \cdot x^P \\
\text{st.} & \sum_{k\in K}\sum_{P\in\mathcal{P}^k} -\bar{x}^P_{ij} \cdot x^P \geq -u_{ij} && \forall (i,j) \in A (\lambda_{ij}) \\
& \sum_{P\in\mathcal{P}^k} x^P = b^k_{s_k} && \forall k \in K (\mu_k) \\
& (x^P \geq 0 \forall k \in K, \forall P \in \mathcal{P}^k)
\end{aligned}$$

$$\begin{aligned}
\text{mx.} & \sum_{(i,j)\in A} -u_{ij}\lambda_{ij} + \sum_{k\in K} b^k_{s_k}\mu_k \\
\text{st.} & \sum_{(i,j)\in P} -\bar{x}^P_{ij}\lambda_{ij} + \mu_k \leq c^P && \forall k \in K, \forall P \in \mathcal{P}^k \\
& \lambda_{ij} \geq 0 && \forall (i,j) \in A
\end{aligned}$$

This is a Linear Programming model, having a corresponding dual:

- the *reduced cost* of each variable x^P is

$$\bar{c}^P := c^P - \sum_{(i,j)\in A} (-\lambda_{ij} \cdot \bar{x}^P_{ij}) - \mu_k = \sum_{(i,j)\in A} (c^k_{ij} + \lambda_{ij}) \cdot \bar{x}^P_{ij} - \mu_k$$

- searching for the variable with most negative reduced cost is *minimum cost s-t path*

Column Generation Algorithm for MMCF

BEGIN
Initialize $\mathcal{S}^k$
do
    solve the **restricted** LP model, considering $\mathcal{S}^k$
    get the values of dual variables $\lambda_{ij} \geq 0$ and μ_k
    for each $k \in K$
        find a **shortest path** on G using (red.) costs
            $\bar{c}_{ij} = c_{ij}^k + \lambda_{ij}$
        obtain a path P of (reduced) cost $\bar{c}^P$
        if $\bar{c}^P - \mu_k < 0$, add P to $\mathcal{S}^k$
while (*any new path has been added to* $\mathcal{S}^k$)
END

Column Generation Example

Orlin's slides 23,21-31

Restricted Master Problem 1

$f(P)$ = flow in path P

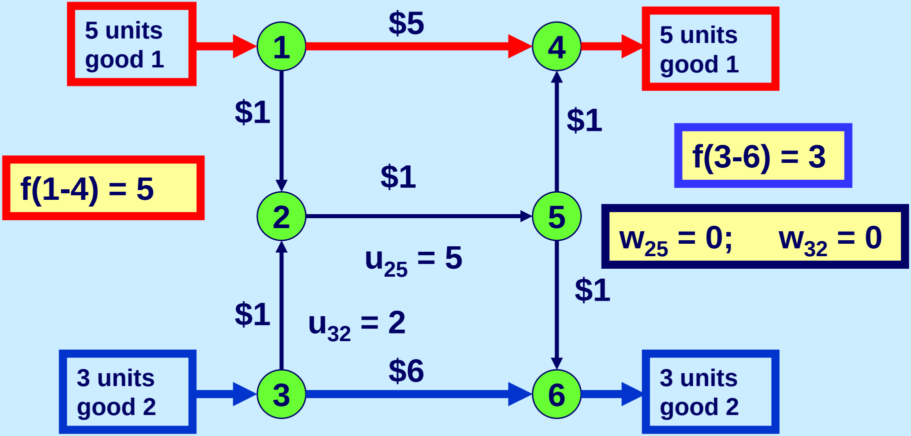
$c(P)$ = cost of path P

$c(1\text{-}4)$	=	5
$c(3\text{-}6)$	=	6

$$
\begin{array}{lll}
\text{Minimize} & 5\,f(1\text{-}4) + 6\,f(3\text{-}6) & \\
\text{subject to} & f(1\text{-}4) & = 5 \\
 & f(3\text{-}6) & = 3
\end{array}
$$

$f(P) \geq 0$ for all paths P

Optimal solution for restricted master 1



The unique shortest path for commodity 1 is 1-2-5-4.

The unique shortest path for commodity 2 is 3-2-5-6.

Restricted Master Problem 2

Suppose we add path 3-2-5-6 to the restricted master

f(P) = flow in path P

c(P) = cost of path P

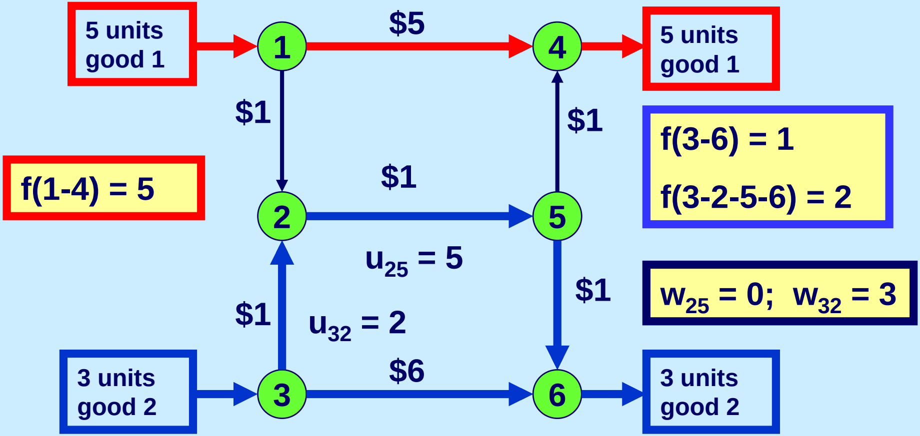
c(1-4) = 5

c(3-6) = 6

c(3-2-5-6) = 3

$$\begin{array}{lll}
\text{Minimize} & 5\,f(1\text{-}4) + 6\,f(3\text{-}6) + 3\,f(3\text{-}2\text{-}5\text{-}6) & \\
\text{subject to} & f(1\text{-}4) = 5 & \\
 & f(3\text{-}6) + f(3\text{-}2\text{-}5\text{-}6) = 3 & \\
 & f(3\text{-}2\text{-}5\text{-}6) \leq u_{25} = 5 & \\
 & f(3\text{-}2\text{-}5\text{-}6) \leq u_{32} = 2 & \\
 & f(P) \geq 0 \text{ for all paths } P &
\end{array}$$

Optimal solution for restricted master 2



The unique shortest path for commodity 1 is 1-2-5-4.

The shortest paths for commodity 2 are 3-2-5-6 and 3-6

Restricted Master Problem 3

We next add path 1-2-5-4 to the restricted master

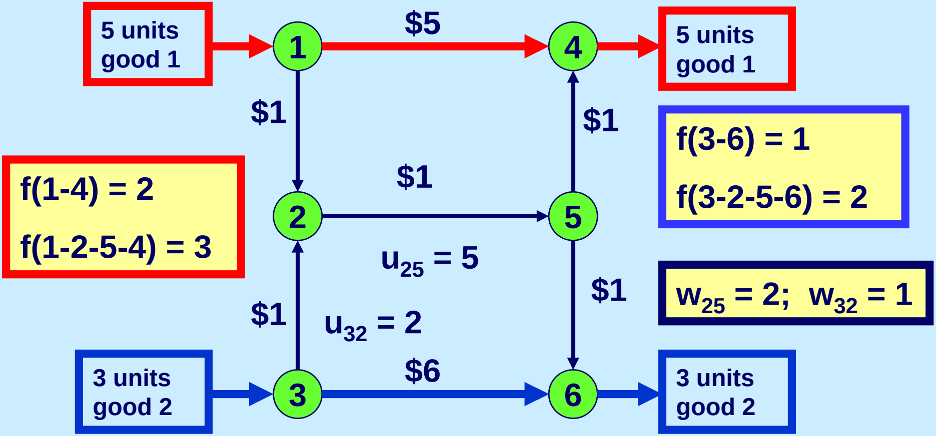
f(P) = flow in path P

c(P) = cost of path P

c(1-4)	=	5
c(1-2-5-4)	=	3
c(3-6)	=	6
c(3-2-5-6)	=	3

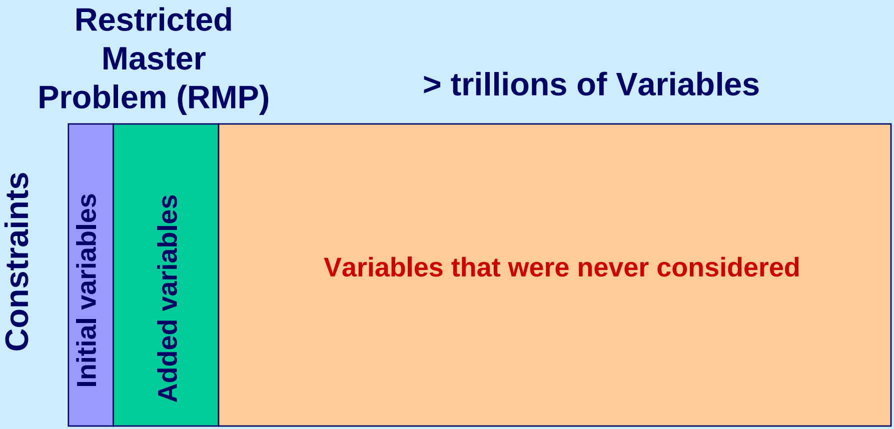
$$\begin{array}{lll} \text{Minimize} & 5\ f(1\text{-}4) + 3\ f(1\text{-}2\text{-}5\text{-}4) + 6\ f(3\text{-}6) + 3\ f(3\text{-}2\text{-}5\text{-}6) & \\ \text{subject to} & f(1\text{-}4) + f(1\text{-}2\text{-}5\text{-}4) & = 5 \\ & f(3\text{-}6) + f(3\text{-}2\text{-}5\text{-}6) & = 3 \\ & f(1\text{-}2\text{-}5\text{-}4) + f(3\text{-}2\text{-}5\text{-}6) & \leq u_{25} = 5 \\ & f(3\text{-}2\text{-}5\text{-}6) & \leq u_{32} = 2 \\ & f(P) \geq 0 \text{ for all paths } P & \end{array}$$

Optimal solution for the path based version



The solution is optimal for the entire problem.

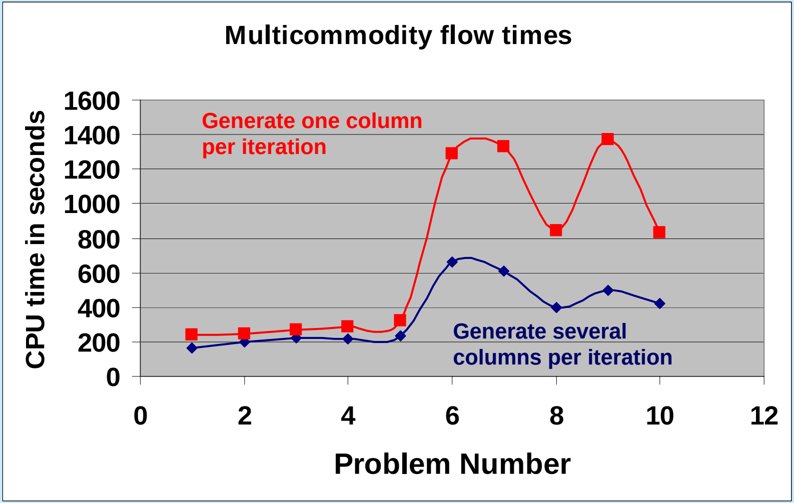
Column Generation



Choices in running column generation

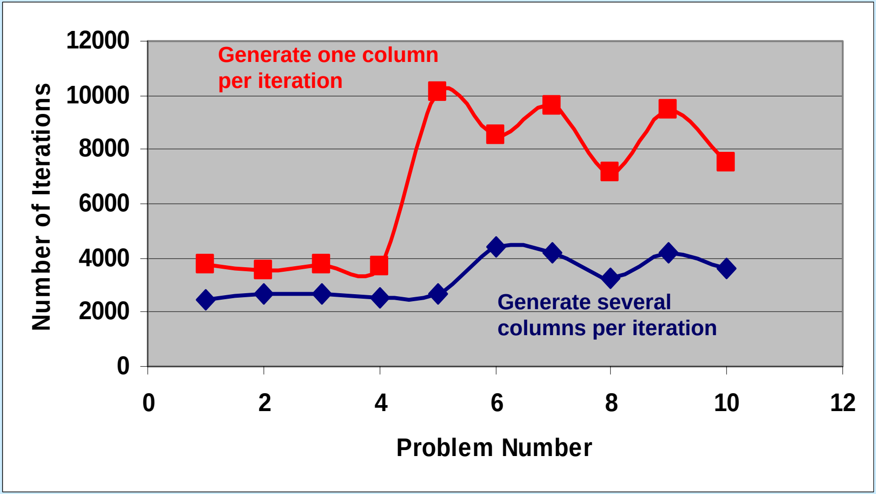
- Starting columns
- How many columns to generate at a time
- Which LP solver to use
- and more

Some running times

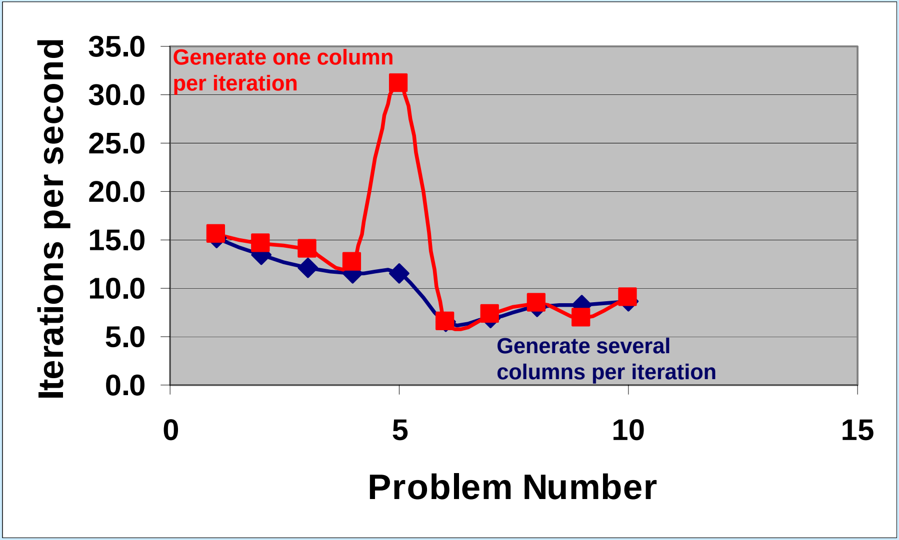


301 nodes, 497 arcs, 1320 commodities.
Times are on an IBM RS6000/590.

The number of iterations per problem



Number of iterations per second



MMCF lab session

Implementing Lagrangean Relaxation and Column Generation MMCF algorithms in AMPL.