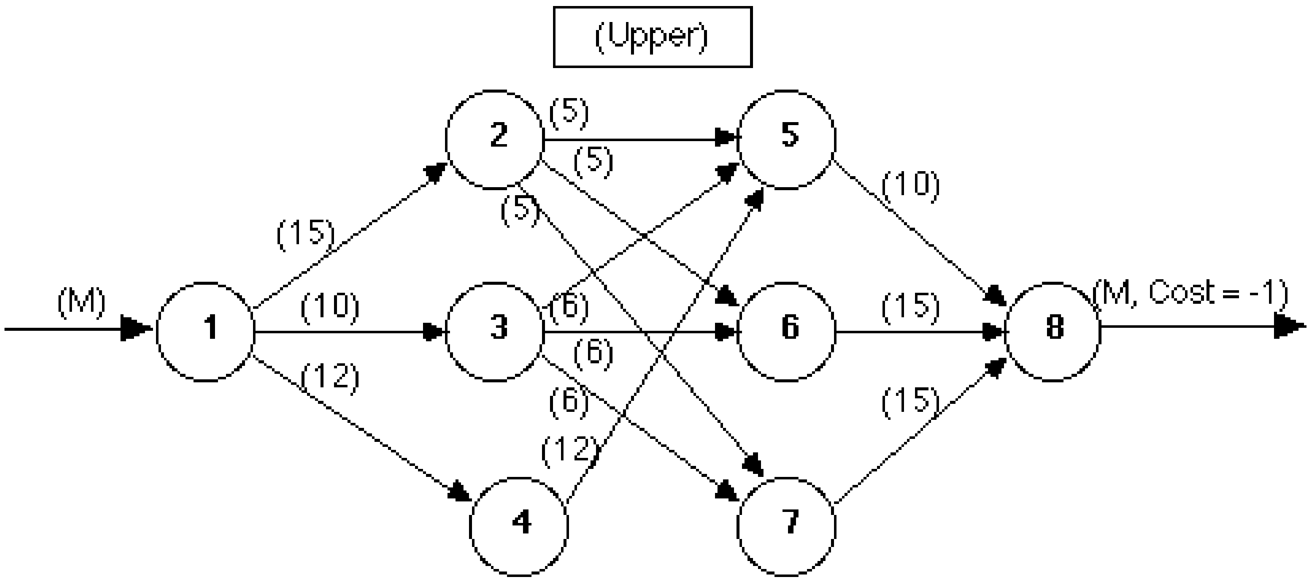
Maximum Flows

Definitions

- Network = directed weighted graph with source node s and sink node t
- s has no incoming edges, t has no outgoing edges
- Weight c_e of an edge e = capacity of e (nonnegative!)



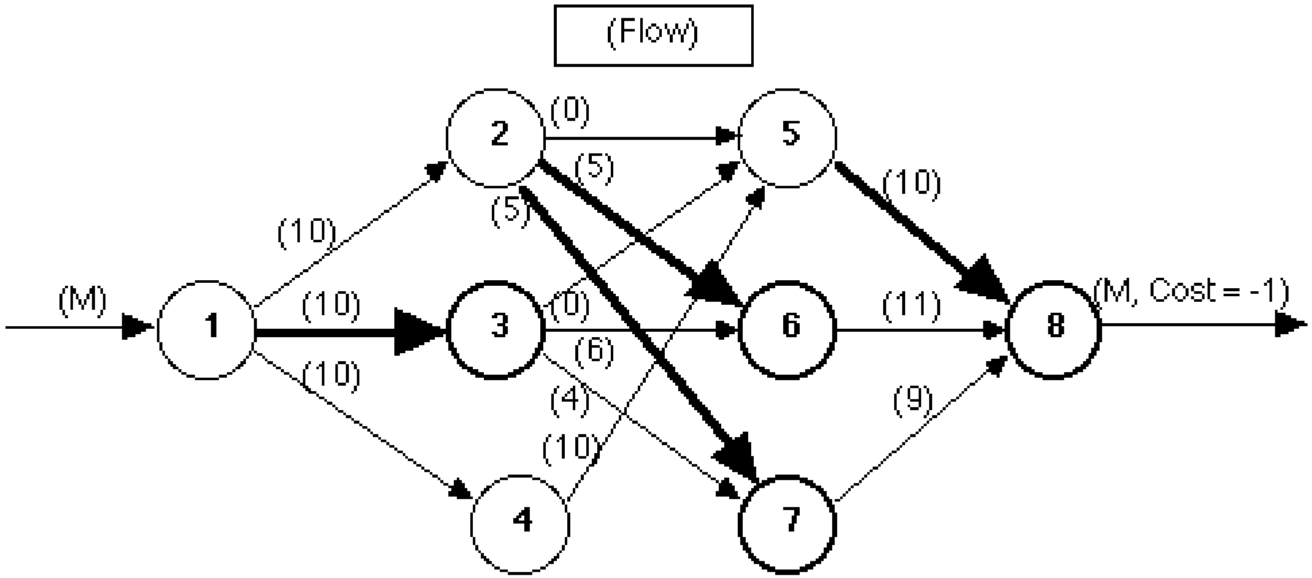
Definitions

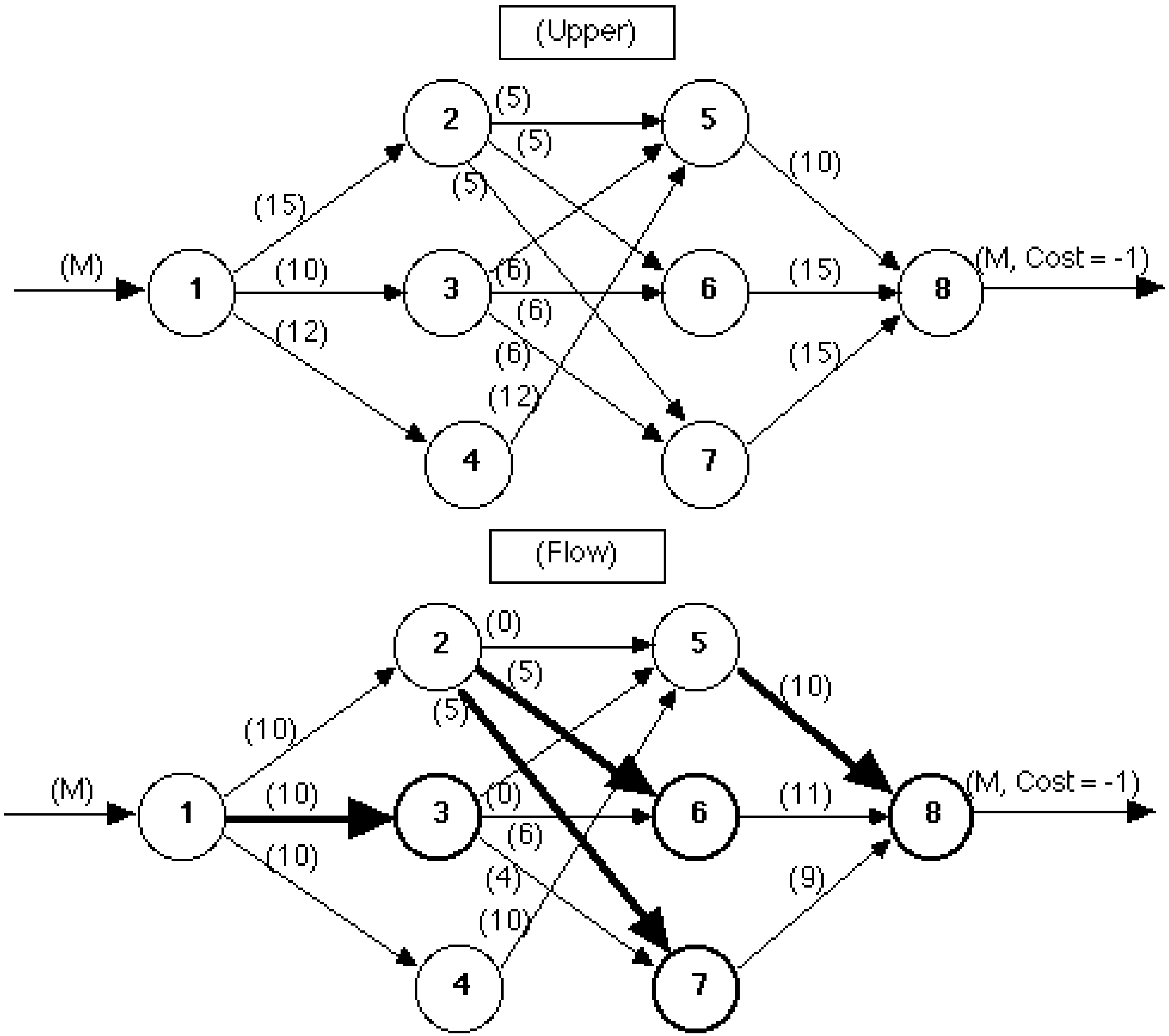
- Flow = function f_e on the edges, $0 \leq f_e \leq c_e \forall e$

  For each node: total incoming flow = total outgoing flow

- Value of a flow = total outgoing flow from s

- Goal: find a flow with *maximum value*





(Upper)
(M)
(15)
(10)
(12)
(5)
(5)
(5)
(6)
(6)
(6)
(12)
(10)
(15)
(15)
(M, Cost = -1)
1
2
3
4
5
6
7
8
(Flow)
(M)
(10)
(10)
(10)
(0)
(5)
(5)
(0)
(6)
(4)
(10)
(10)
(11)
(9)
(M, Cost = -1)
1
2
3
4
5
6
7
8

Applications

- Oil pipes
- Traffic flows on highways
- Machine scheduling. Example:

Job	1	2	3	4
Size	1.5	1.25	2.1	3.6
Release date	3	1	3	5
Due date	5	4	7	9

Suppose we have three machines. Does a feasible schedule exist?

Machine scheduling as a maxflow problem

- First layer of nodes contains the jobs

  Each arc from s to a job has capacity equal to that job size

- Second layer of nodes contains intervals without release dates or due dates

  Arc from job to admissible interval I has capacity equal to length of interval $\ell(I)$

  Arc from each interval to t has capacity $3\ell(I) =$ total amount of work we can do in this interval (there are 3 machines)

Matrix rounding

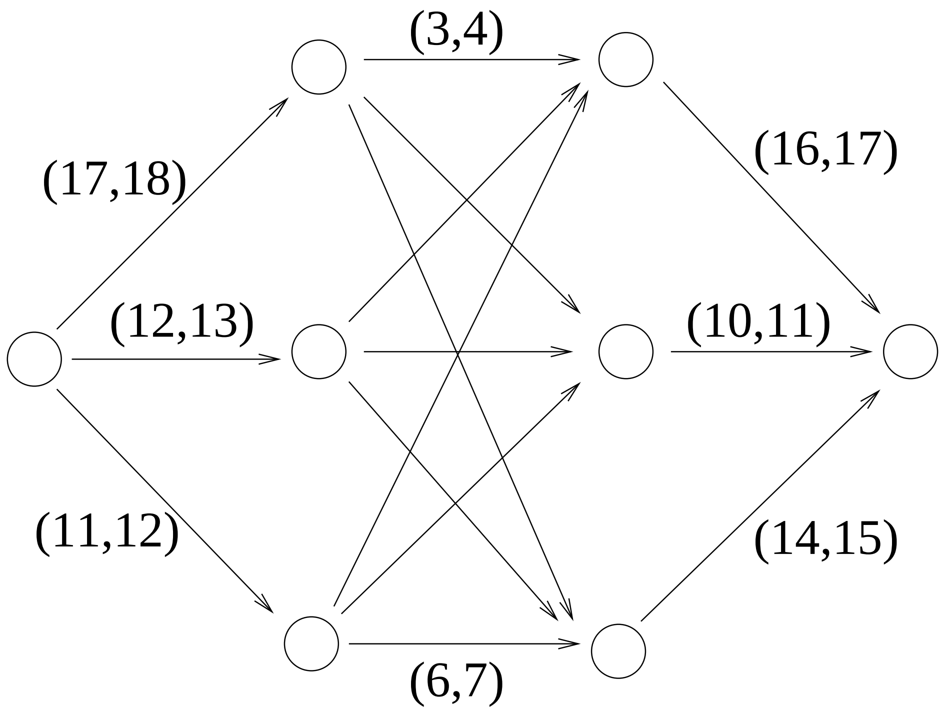
3.1	6.8	7.3	17.2
9.6	2.4	0.7	12.7
3.6	1.2	6.5	11.3
16.3	10.4	14.5	

- Matrix with real numbers, column sums, row rums
- We can round each number up or down
- We want to get a *consistent* rounding

  sum of rounded numbers in each row = rounded row sum

Matrix rounding as a feasible flow problem

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	



Feasible flow in this network = consistent rounding

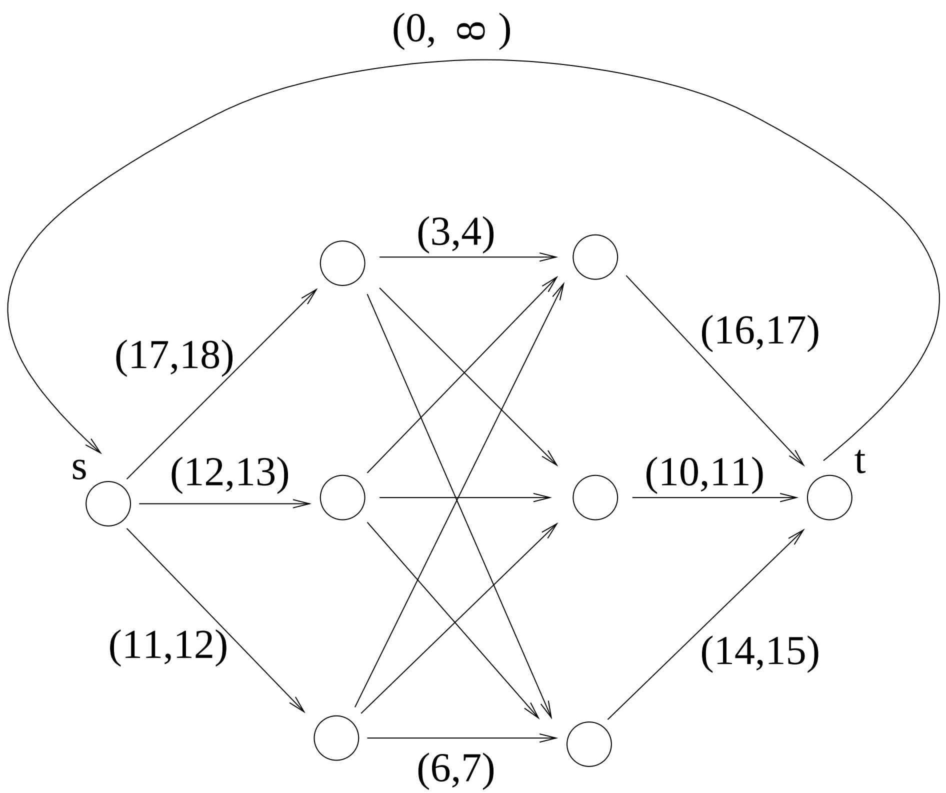
Feasible flow as a maxflow problem

Feasible circulation: for each node i, incoming flow minus outgoing flow $= 0$.

Upper and lower bounds on flow on each arc

New flow variables: *subtract lower bound* from all flow variables and constraints

Now, for each node i, incoming flow minus outgoing flow $= b(i)$



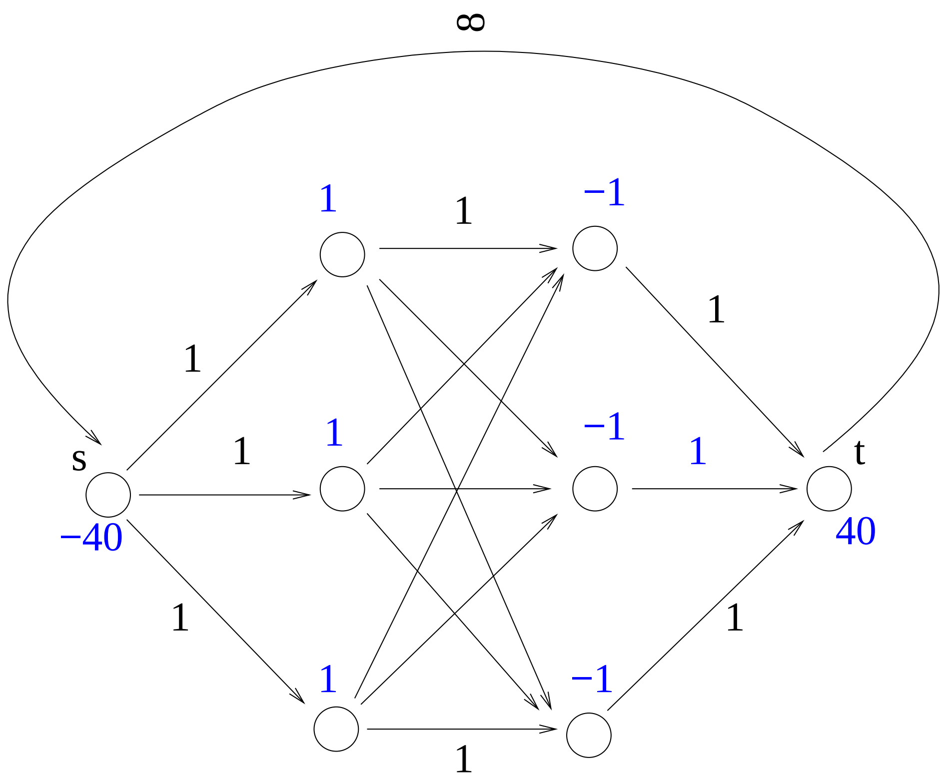
Feasible flow as a maxflow problem

Feasible circulation: for each node i, incoming flow minus outgoing flow $= 0$.

Upper and lower bounds on flow on each arc

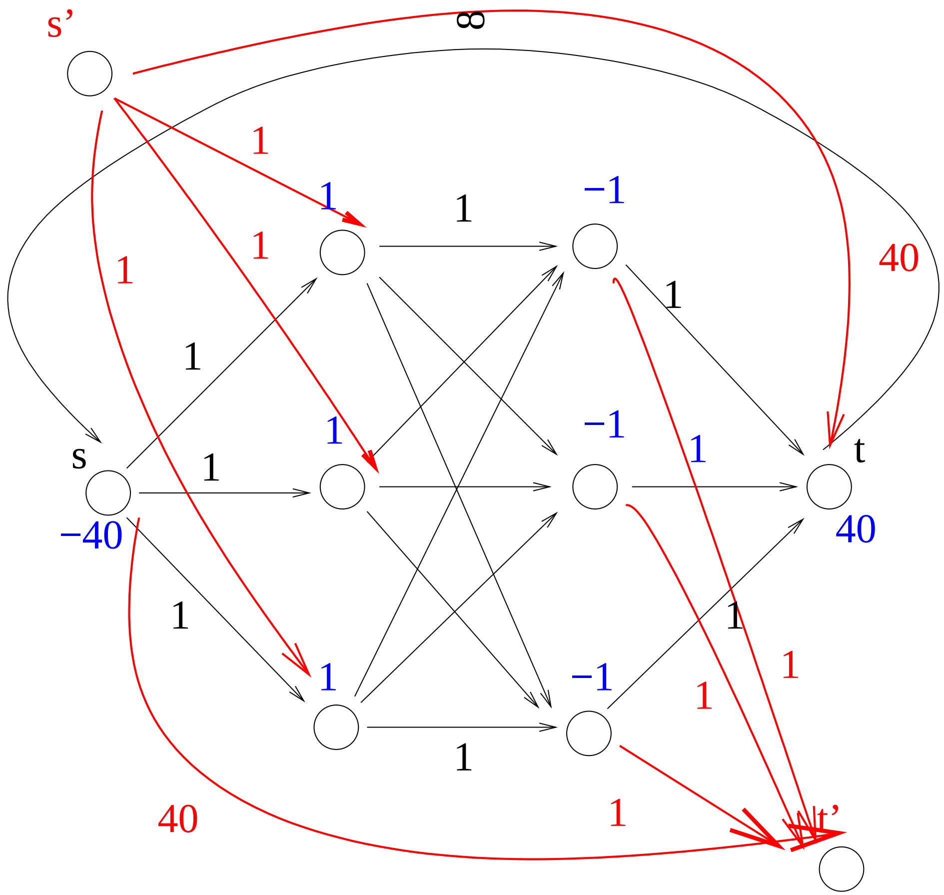
New flow variables: *subtract lower bound* from all flow variables

Now, for each node i, incoming flow minus outgoing flow $= b(i)$



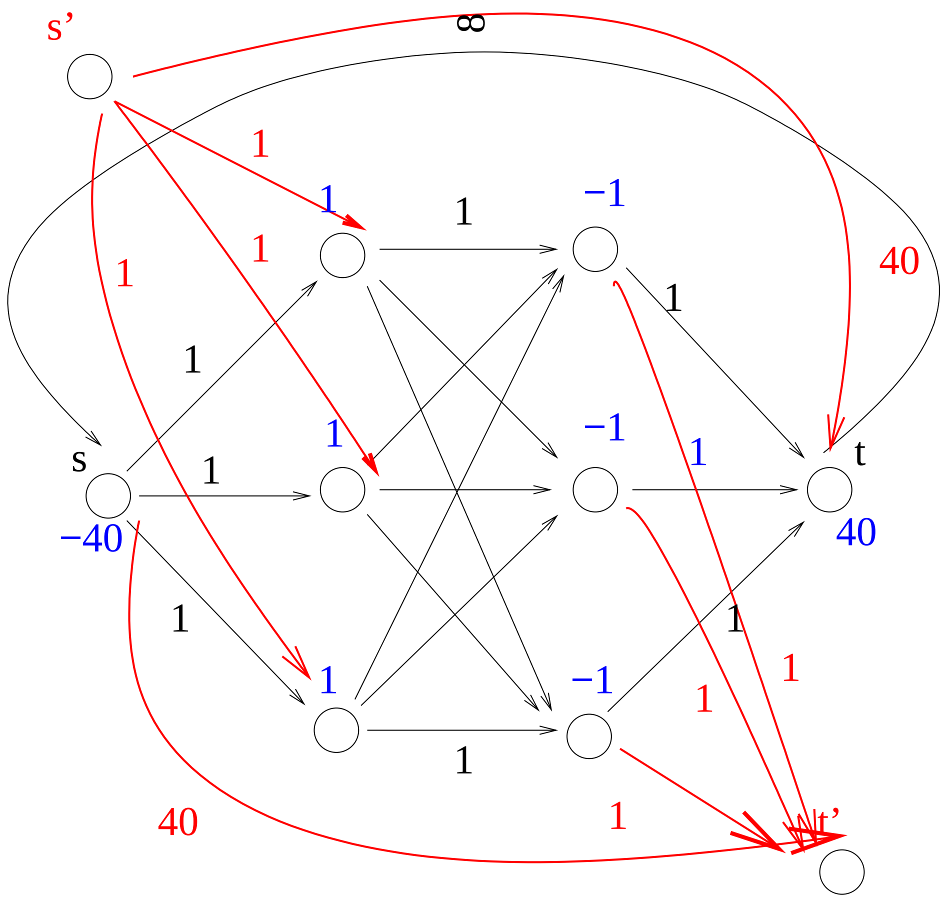
Feasible flow as a maxflow problem

- Add new sink s' and new source t'
- For each node i with $b(i) > 0$, add arc with capacity $b(i)$ from s'
- For each node i with $b(i) < 0$, add arc with capacity $-b(i)$ to t'
- Find maximum flow from s' to t'



Feasible flow as a maxflow problem

- If we find a flow that saturates all source and sink arcs, we have a feasible flow in the original network
- If the maximum flow does not saturate those edges, no feasible flow exists!



Option 1: linear programming

- Flow variables x_e for each edge e
- Flow on each edge is at most its capacity
- Incoming flow at each vertex = outgoing flow from this vertex
- Maximize outgoing flow from starting vertex

We can do better!

Algorithms 1956–now

Year	Author	Running time
1956	Ford-Fulkerson	$O(mnU)$
1969	Edmonds-Karp	$O(m^2n)$
1970	Dinic	$O(mn^2)$
1973	Dinic-Gabow	$O(mn\log U)$
1974	Karzanov	$O(n^3)$
1977	Cherkassky	$O(n^2\sqrt{m})$
1980	Galil-Naamad	$O(mn\log^2 n)$
1983	Sleator-Tarjan	$O(mn\log n)$

$n =$ number of nodes

$m =$ number of arcs

$U =$ largest capacity

Year	Author	Running time
1986	Goldberg-Tarjan	$O(mn\log(n^2/m))$
1987	Ahuja-Orlin	$O(mn+n^2\log U)$
1987	Ahuja-Orlin-Tarjan	$O(mn\log(2+n\sqrt{\log U}/m))$
1990	Cheriyan-Hagerup-Mehlhorn	$O(n^3/\log n)$
1990	Alon	$O(mn+n^{8/3}\log n)$
1992	King-Rao-Tarjan	$O(mn+n^{2+e})$
1993	Philipps-Westbrook	$O(mn\log n/\log\frac{m}{n}+n^2\log^{2+\varepsilon} n)$
1994	King-Rao-Tarjan	$O(mn\log n/\log\frac{m}{n\log n})$ if $m\geq 2n\log n$
1997	Goldberg-Rao	$O(min\{m^{1/2},n^{2/3}\}m\log(n^2/m)\log U)$

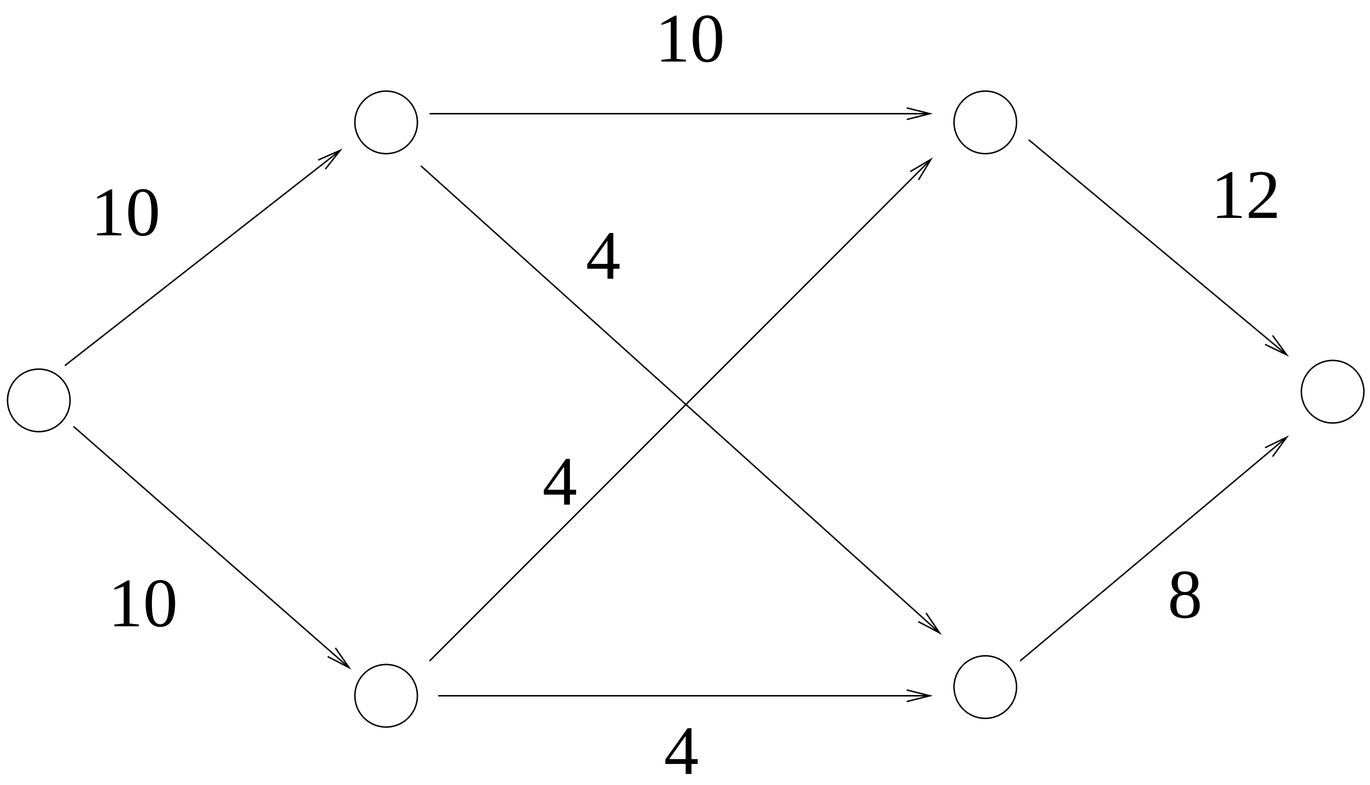
Augmenting paths

Find a path from s to t such that each edge has some spare capacity

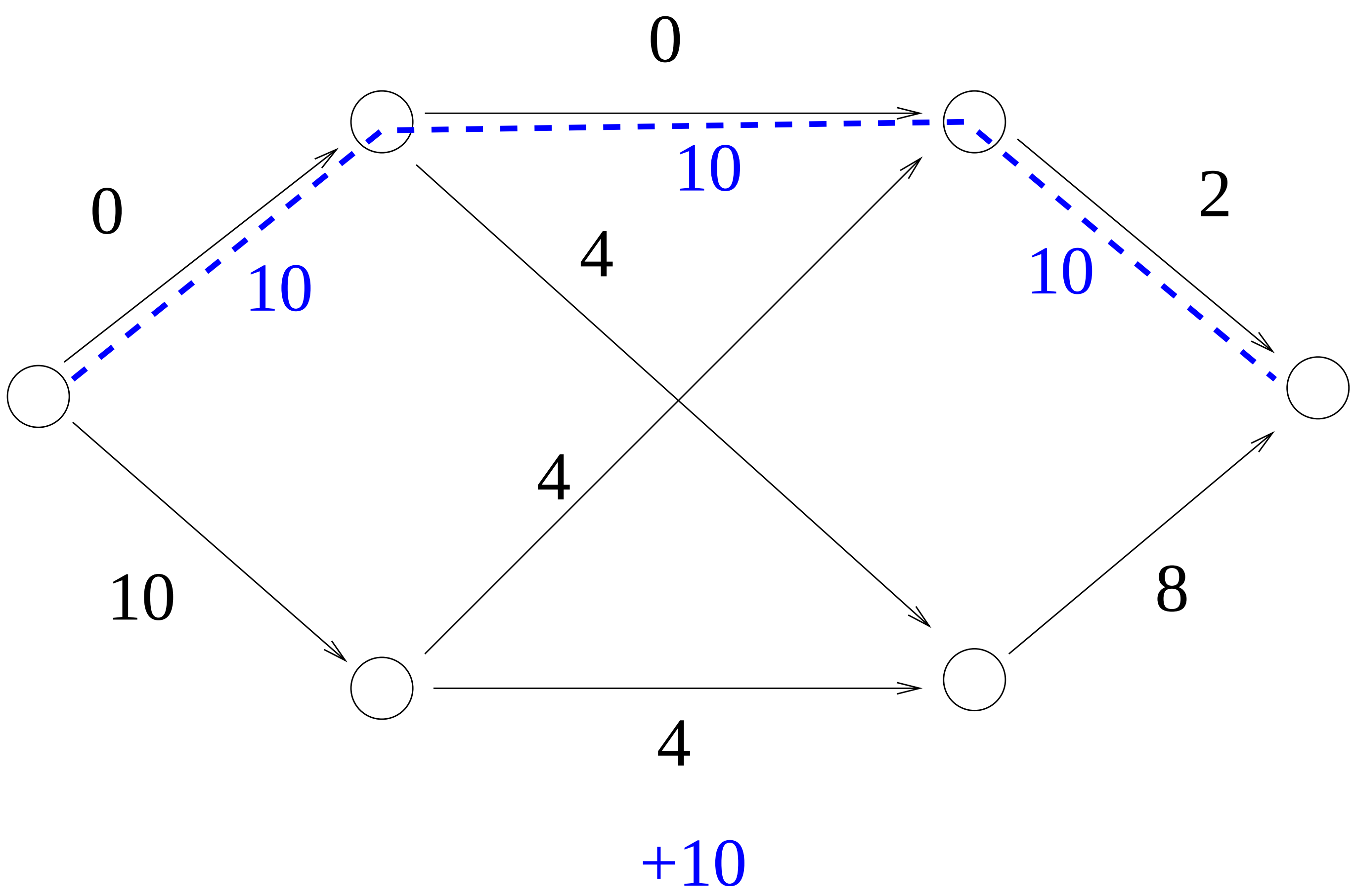
On this path, fill up the edge with the smallest spare capacity

Adjust capacities for all edges (create residual graph) and repeat

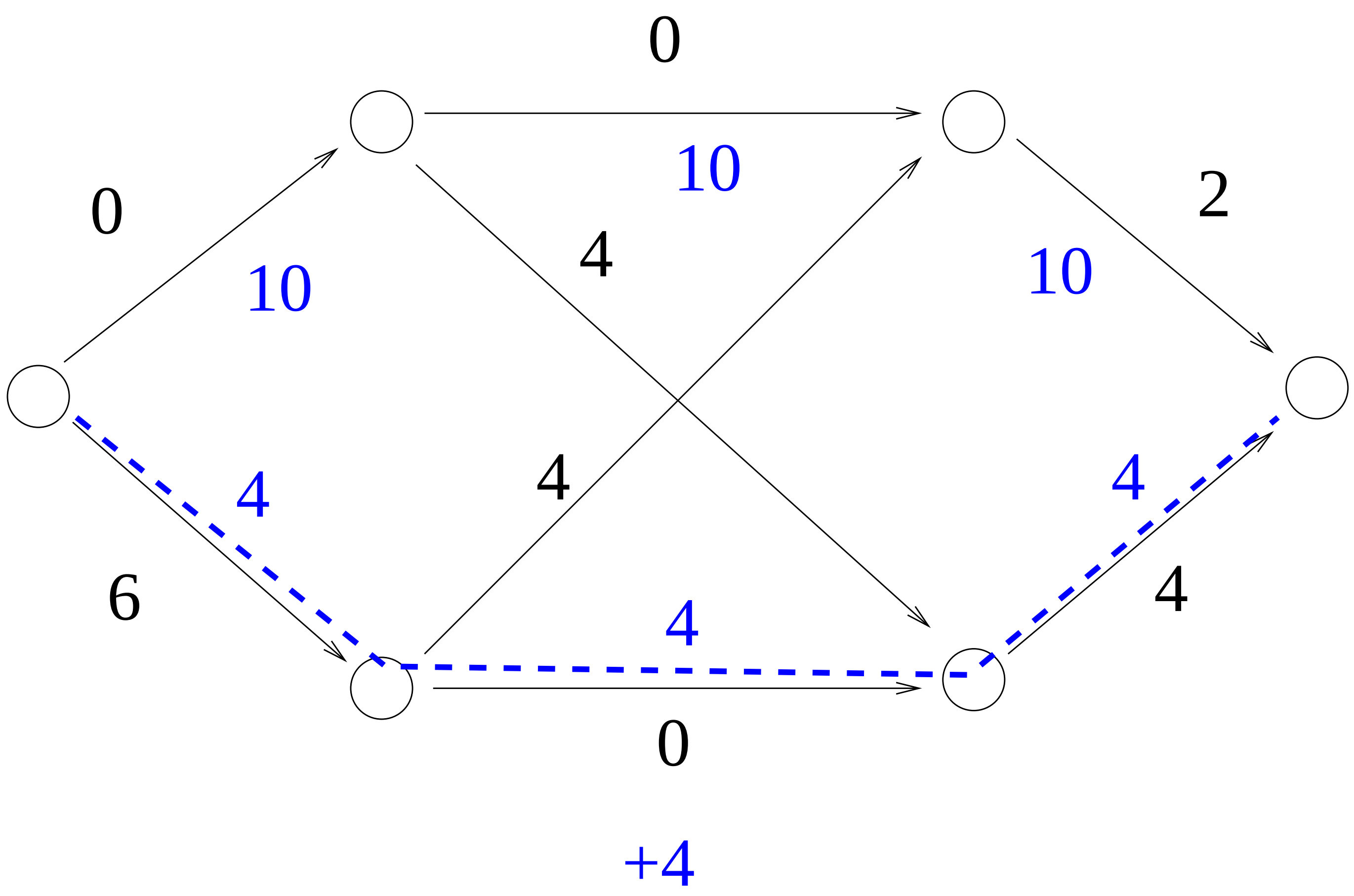
Example



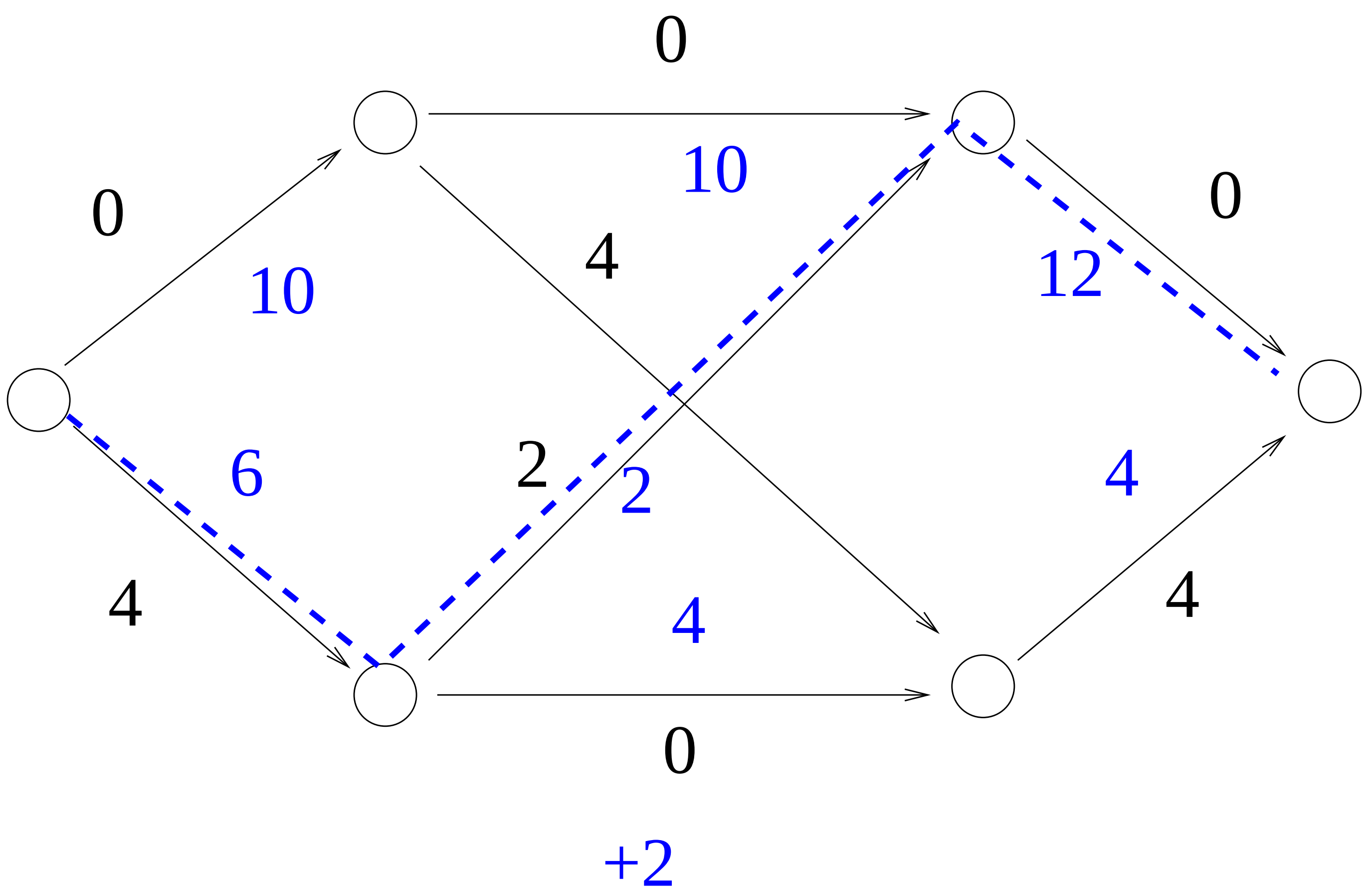
Example



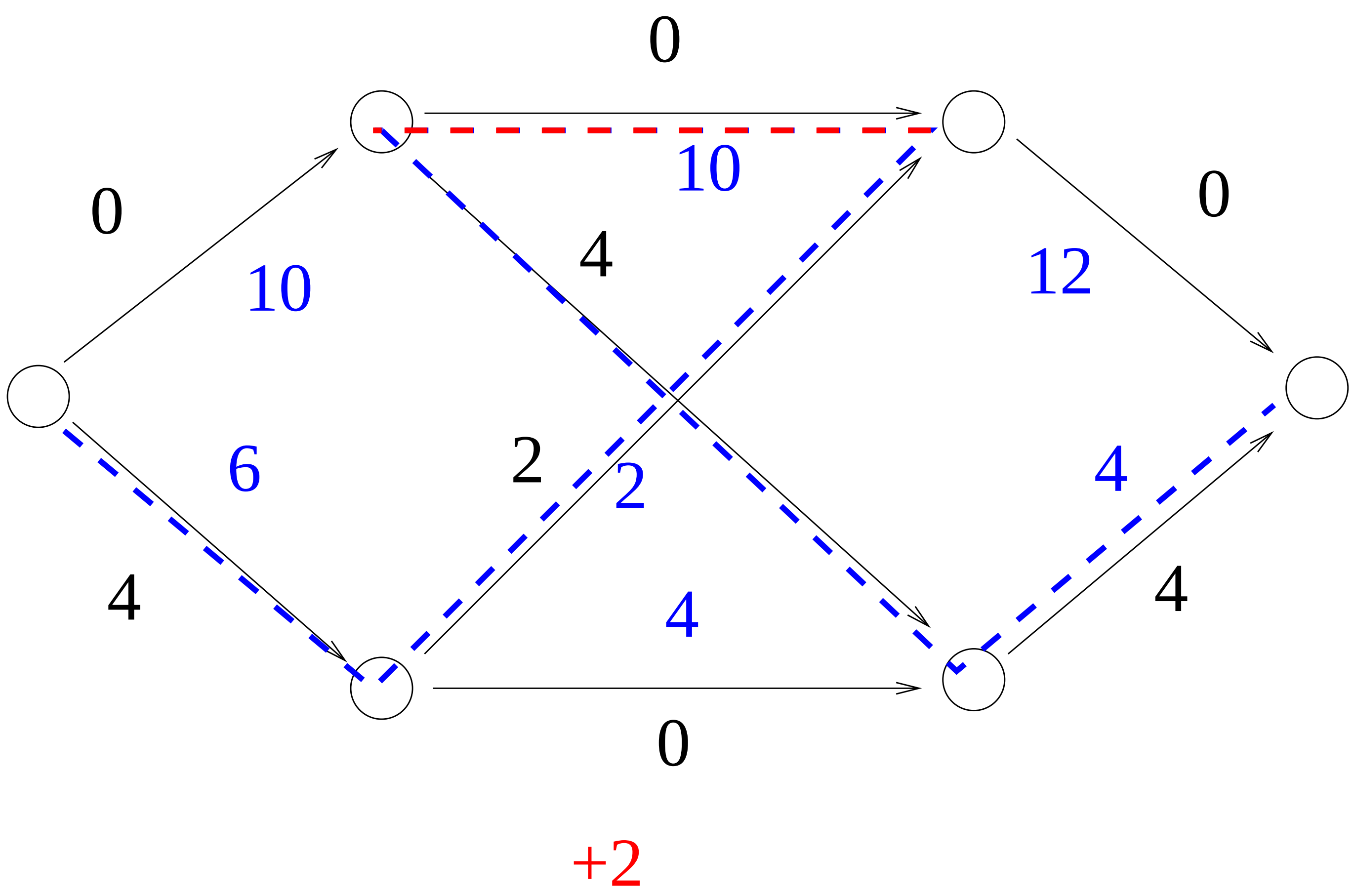
Example



Example



Example



Ford Fulkerson Algorithm

Function FFMaxFlow$(G=(V,E),s,t,\mathsf{cap}:E\to\mathbb{R}):E\to\mathbb{R}$

  $f:=0$

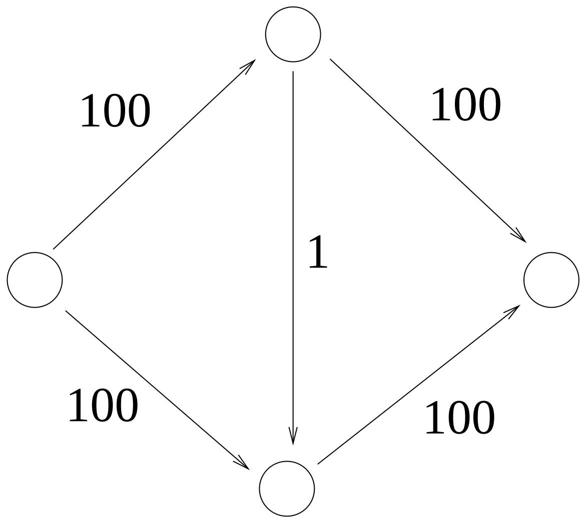
  while $\exists$path $p=(s,\dots,t)$ in G_f **do**

    augment f along p

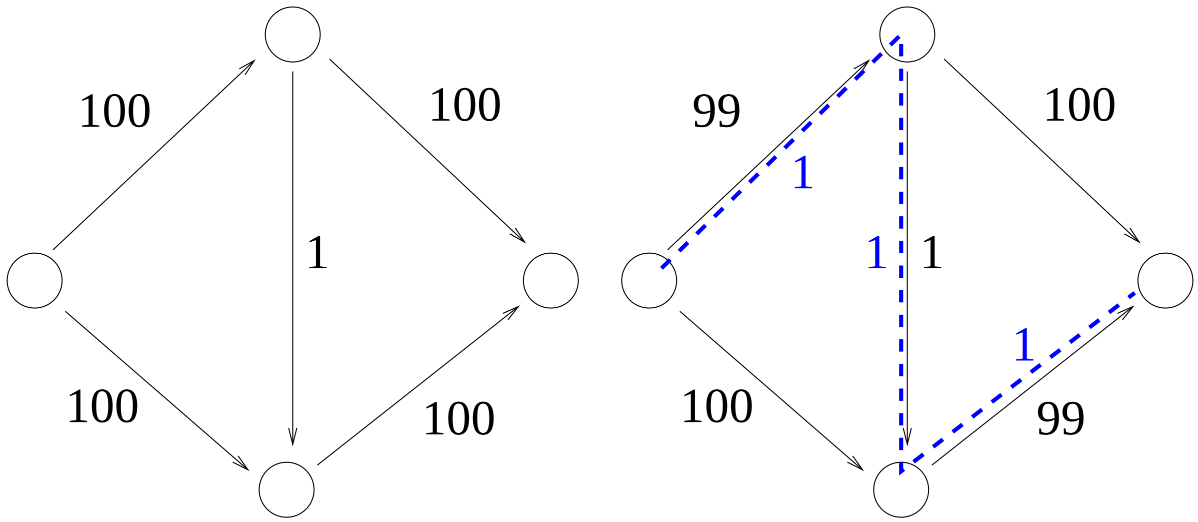
  return f

time $\mathrm{O}(m\,\mathrm{val}(f))$

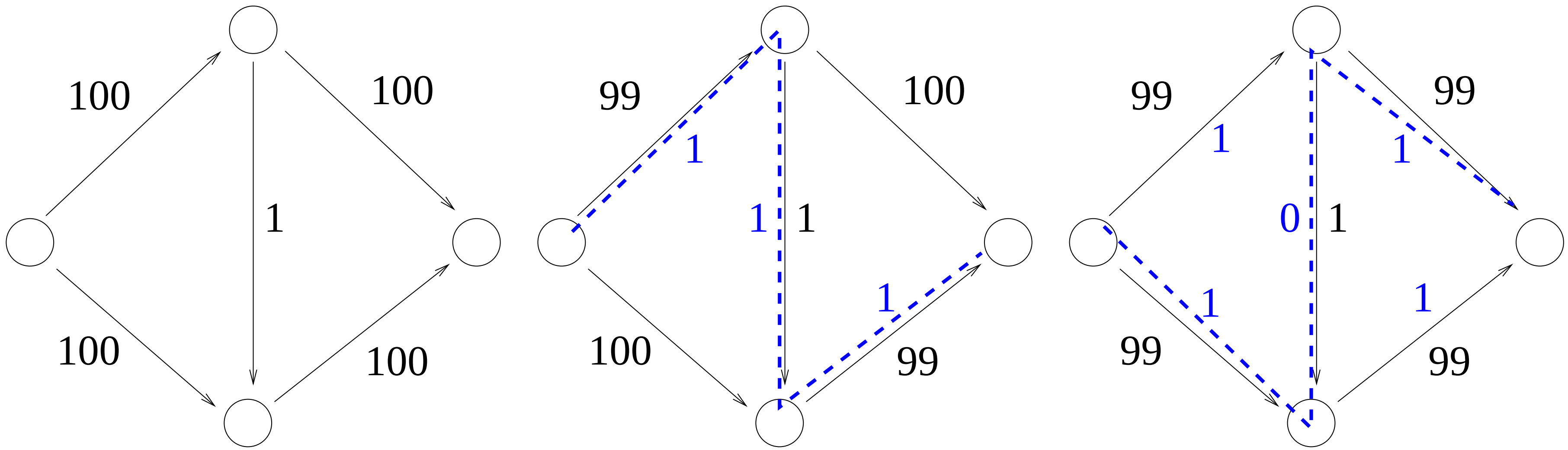
A Bad Example for Ford Fulkerson



A Bad Example for Ford Fulkerson



A Bad Example for Ford Fulkerson

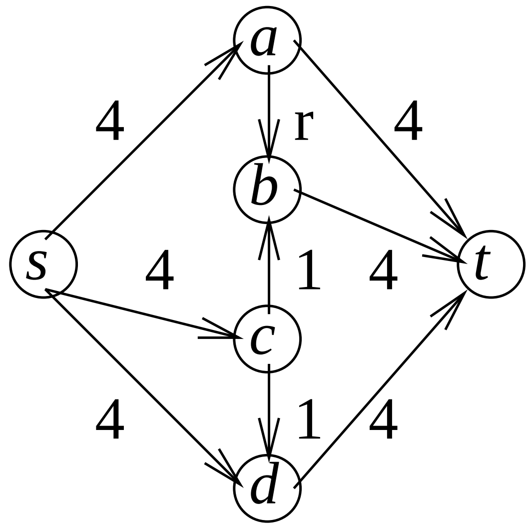


An Even Worse Example for Ford Fulkerson

[U. Zwick, TCS 148, p. 165–170, 1995]

Let $r = \dfrac{\sqrt{5}-1}{2}$.

Consider the graph



And the augmenting paths

$p_0 = \langle s, c, b, t \rangle$

$p_1 = \langle s, a, b, c, d, t \rangle$

$p_2 = \langle s, c, b, a, t \rangle$

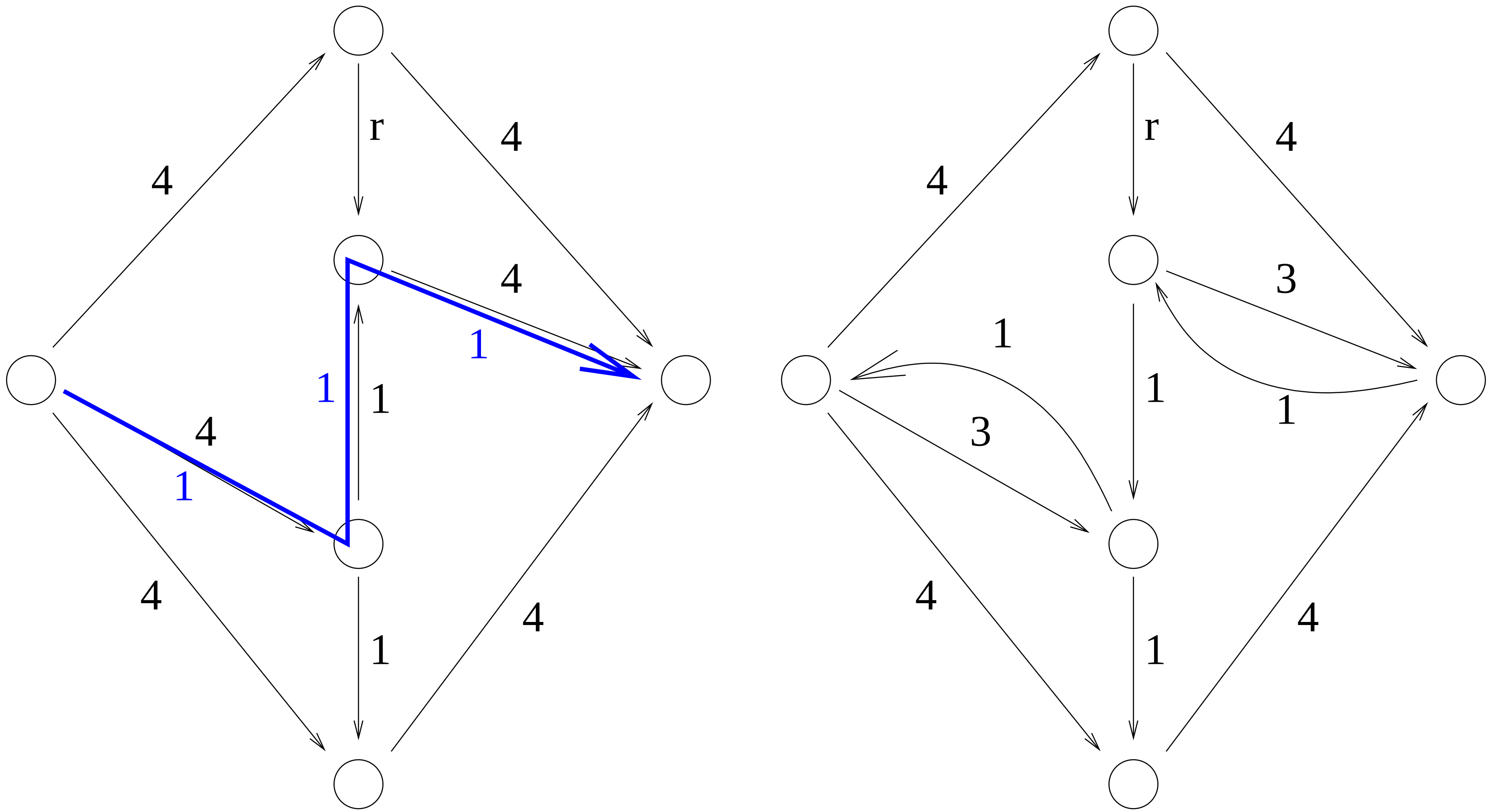
$p_3 = \langle s, d, c, b, t \rangle$

The sequence of augmenting paths $p_0(p_1, p_2, p_1, p_3)^*$ is an infinite sequence of positive flow augmentations.

The flow value does not converge to the maximum value 9.

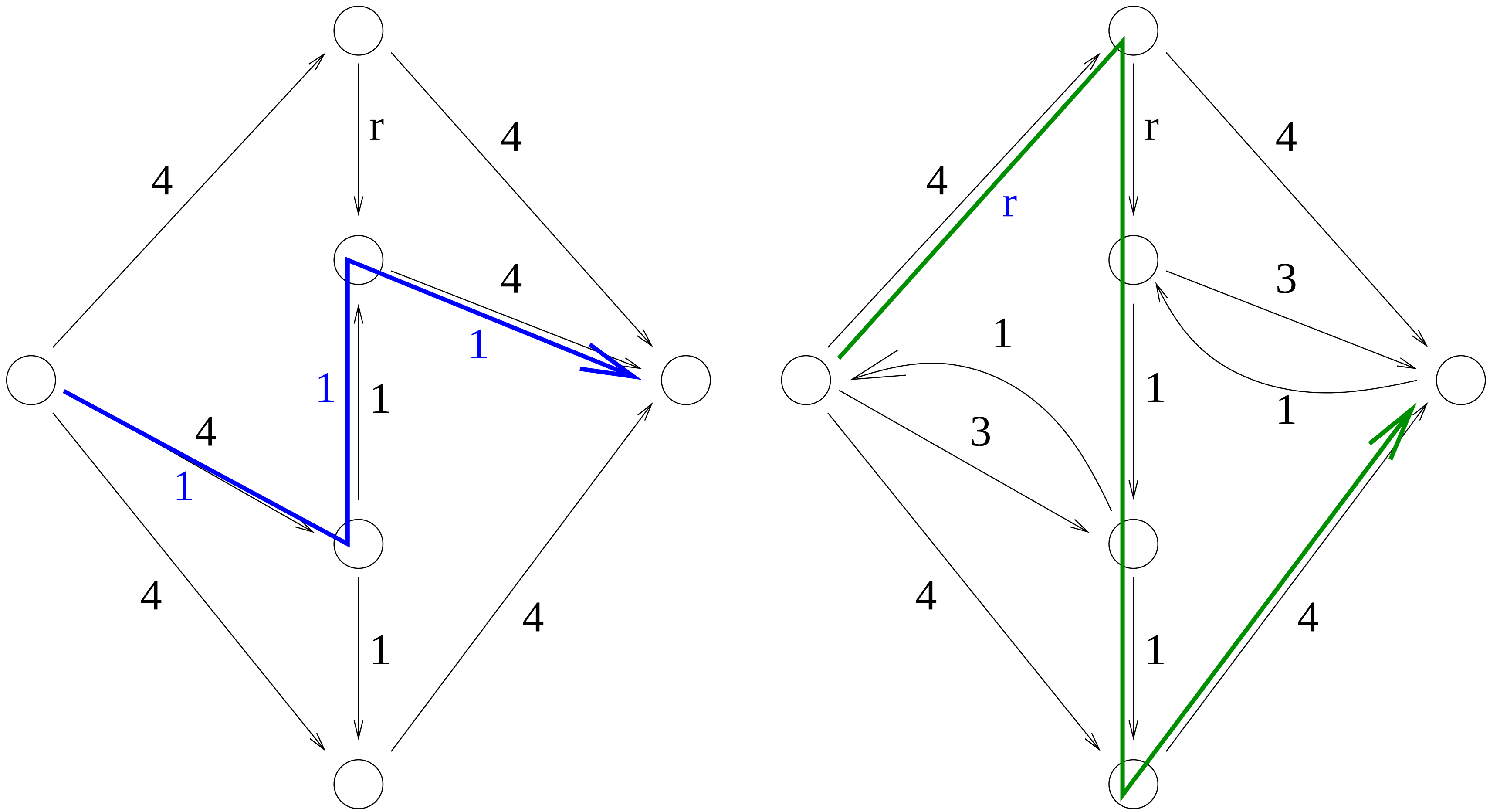
An Even Worse Example for Ford Fulkerson

[U. Zwick, TCS 148, p. 165–170, 1995]



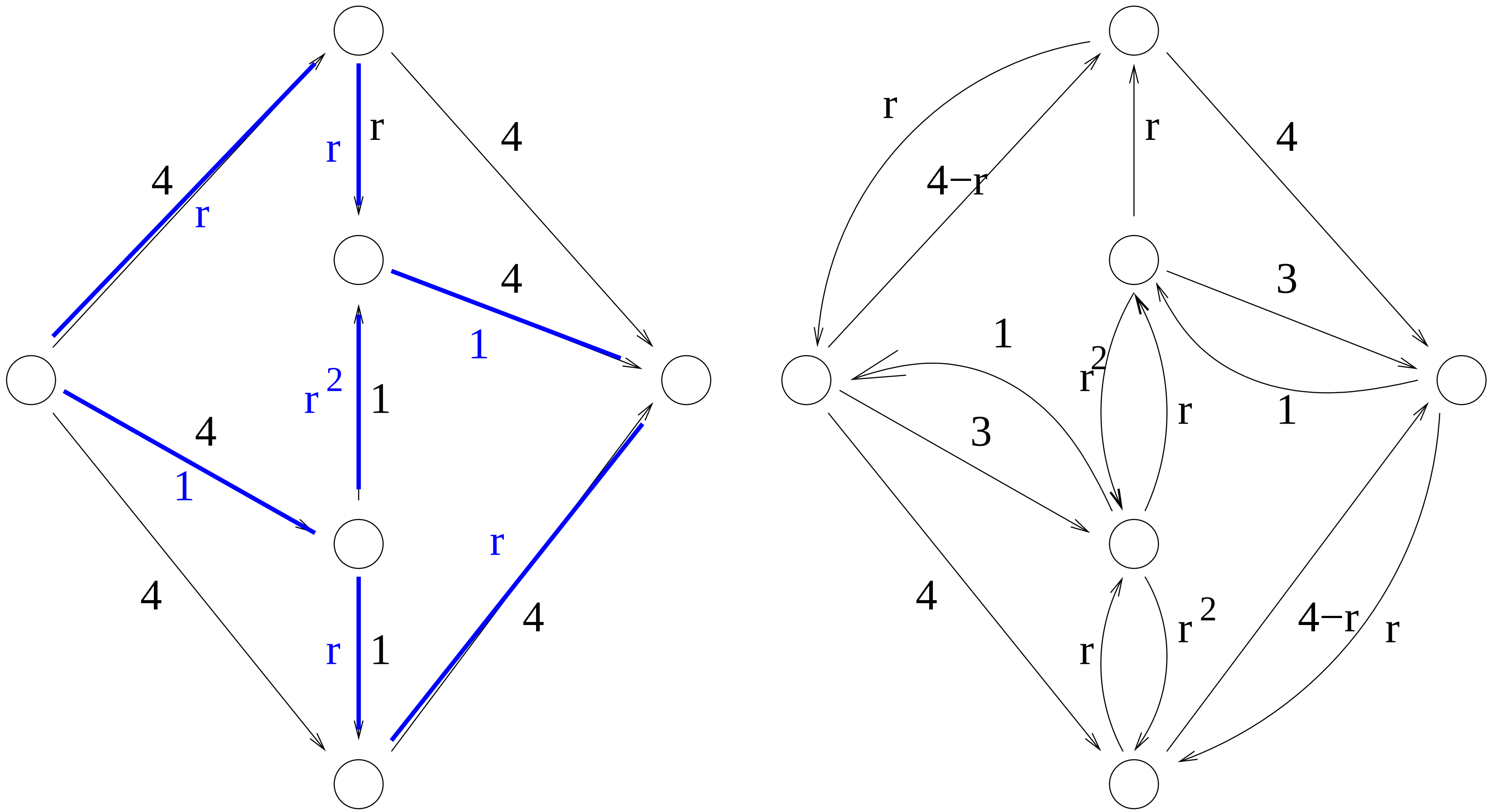
An Even Worse Example for Ford Fulkerson

[U. Zwick, TCS 148, p. 165–170, 1995]



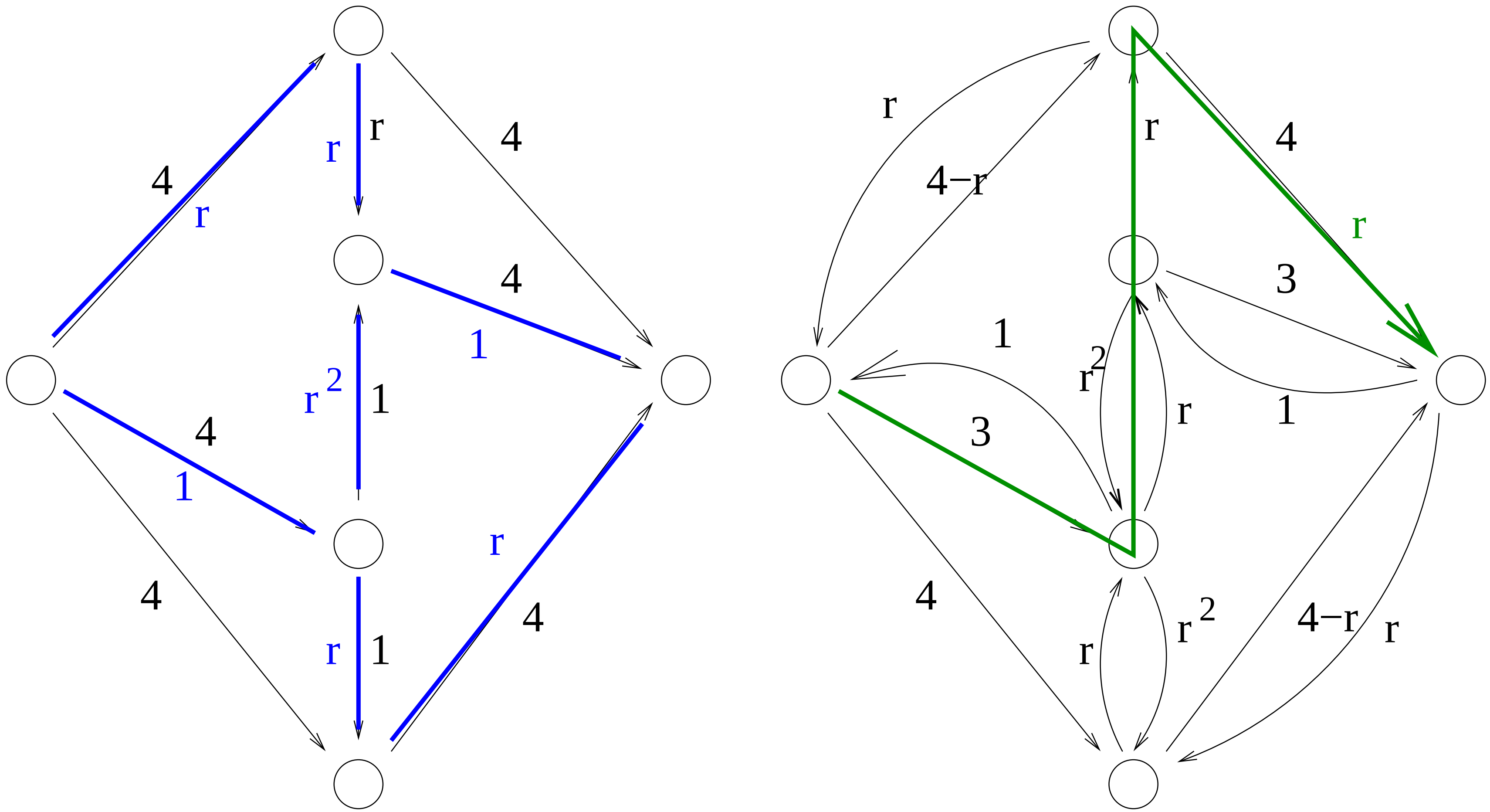
An Even Worse Example for Ford Fulkerson

[U. Zwick, TCS 148, p. 165–170, 1995]



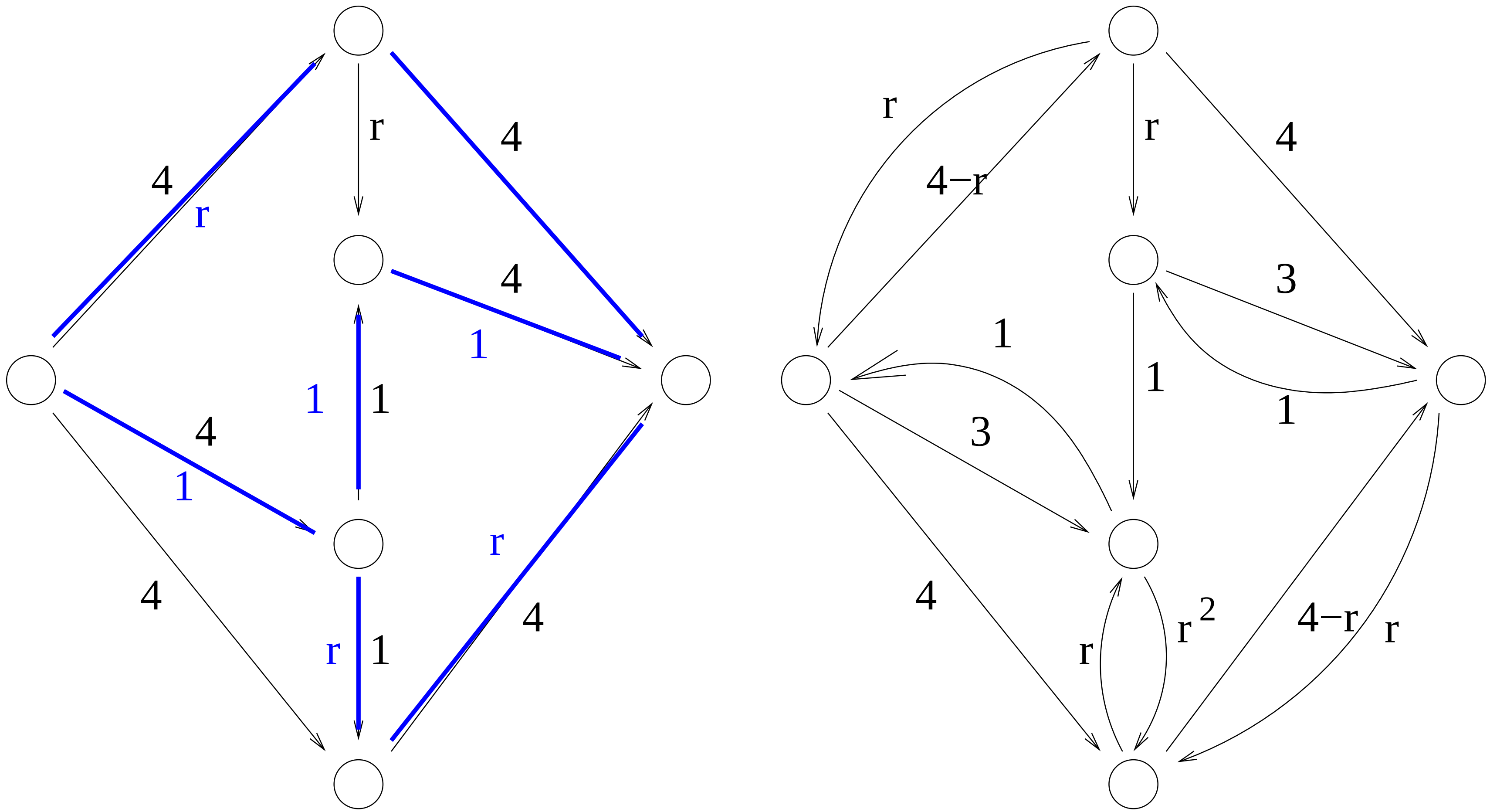
An Even Worse Example for Ford Fulkerson

[U. Zwick, TCS 148, p. 165–170, 1995]



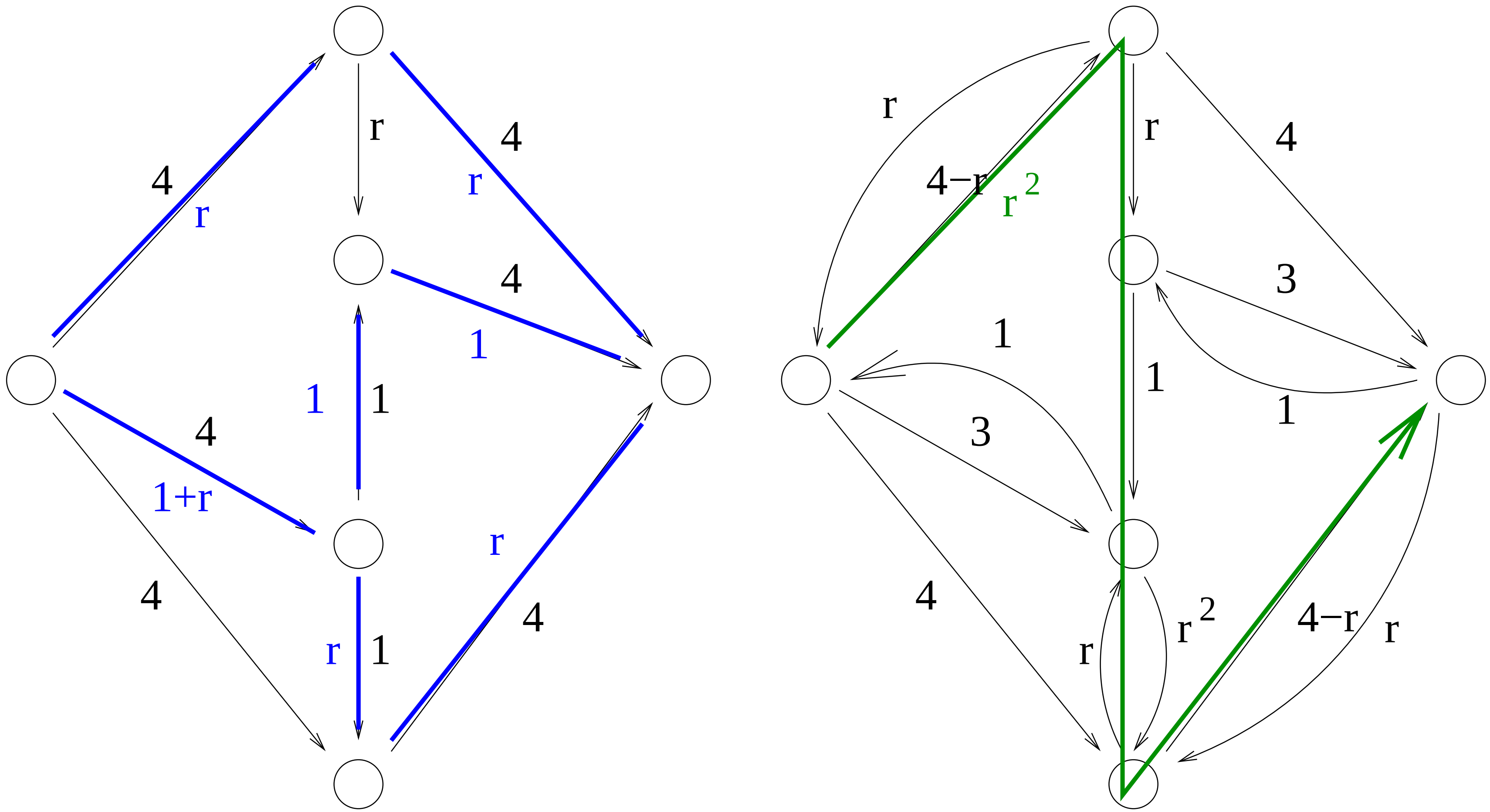
An Even Worse Example for Ford Fulkerson

[U. Zwick, TCS 148, p. 165–170, 1995]



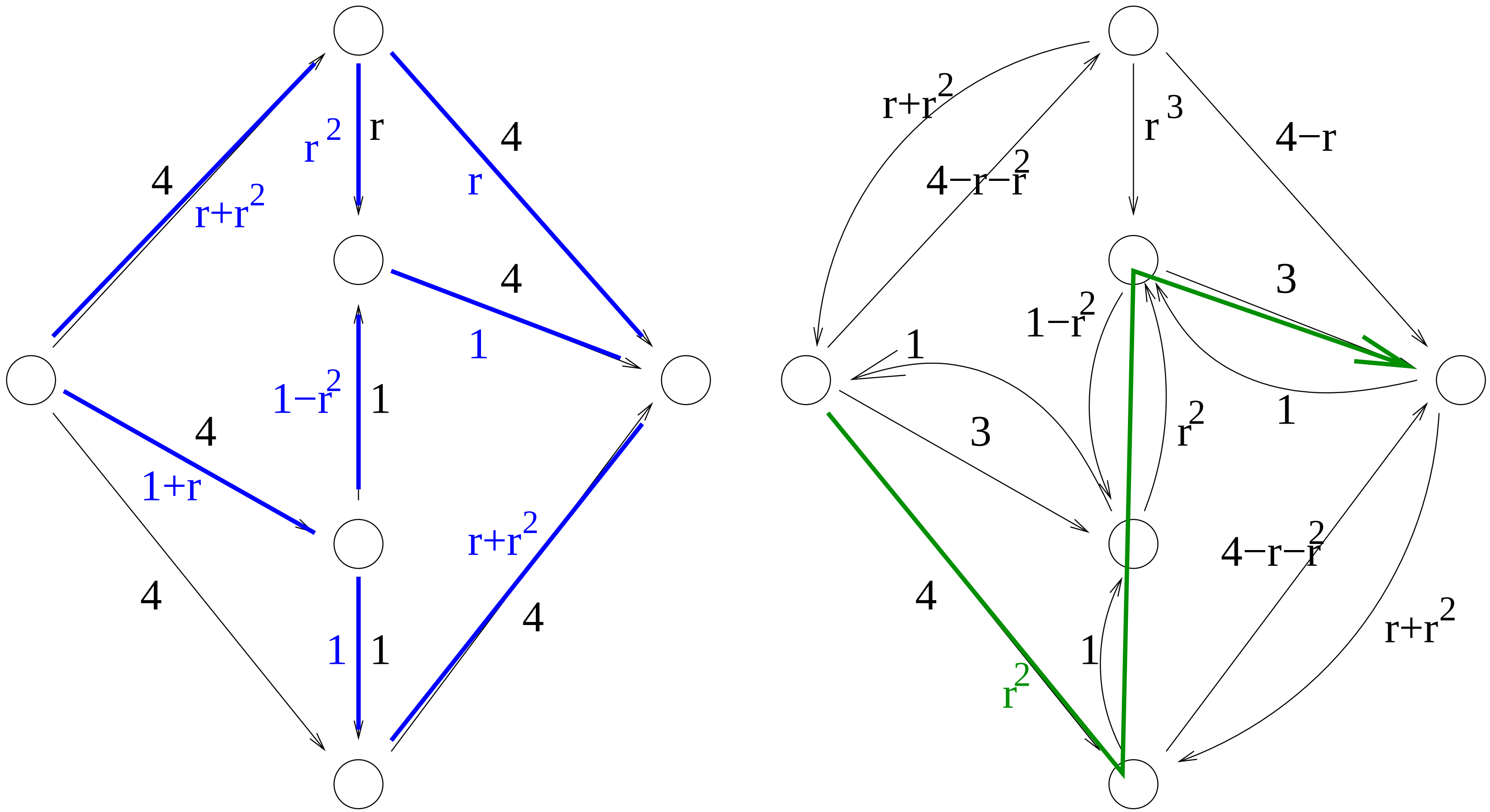
An Even Worse Example for Ford Fulkerson

[U. Zwick, TCS 148, p. 165–170, 1995]



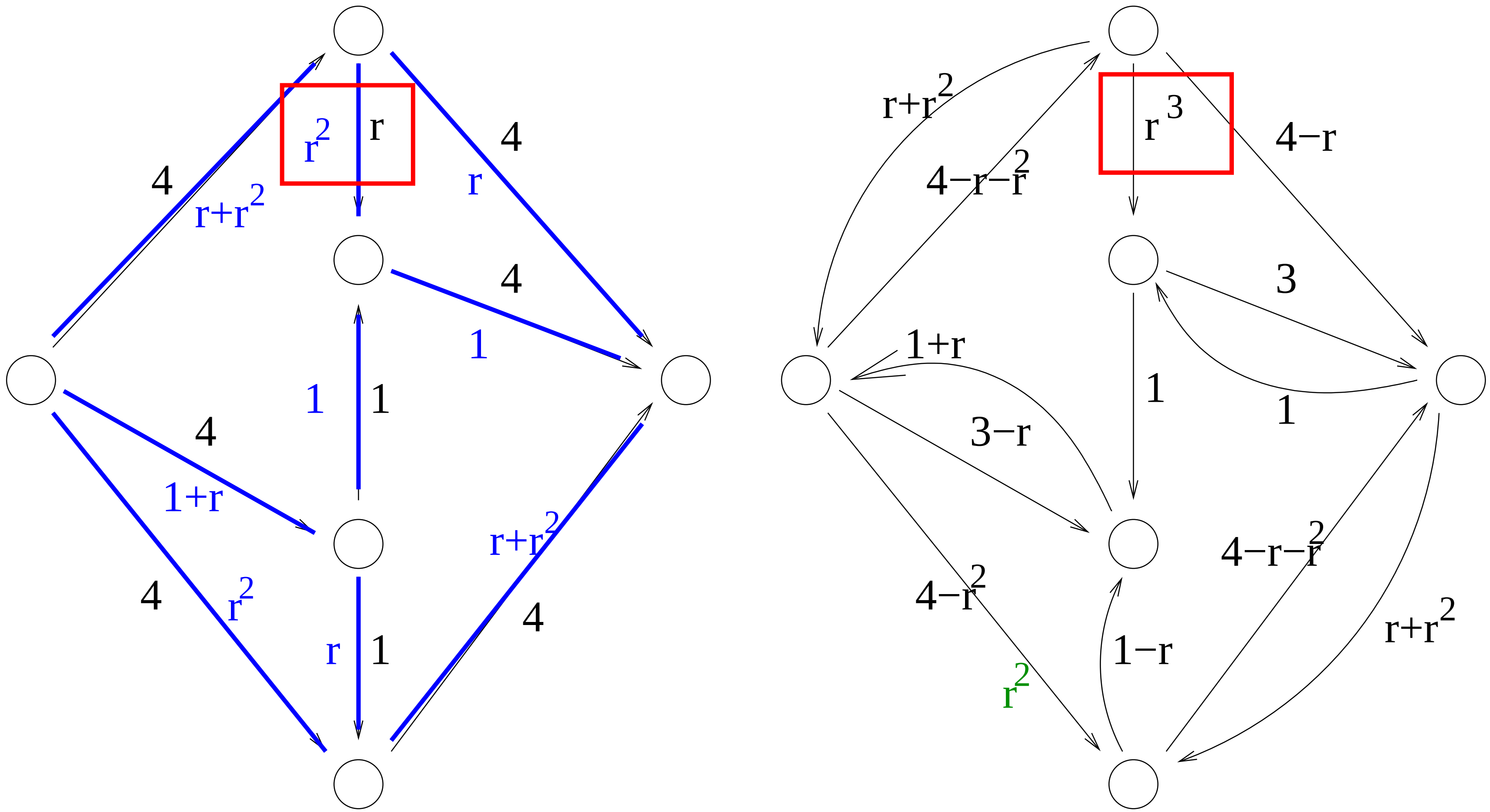
An Even Worse Example for Ford Fulkerson

[U. Zwick, TCS 148, p. 165–170, 1995]



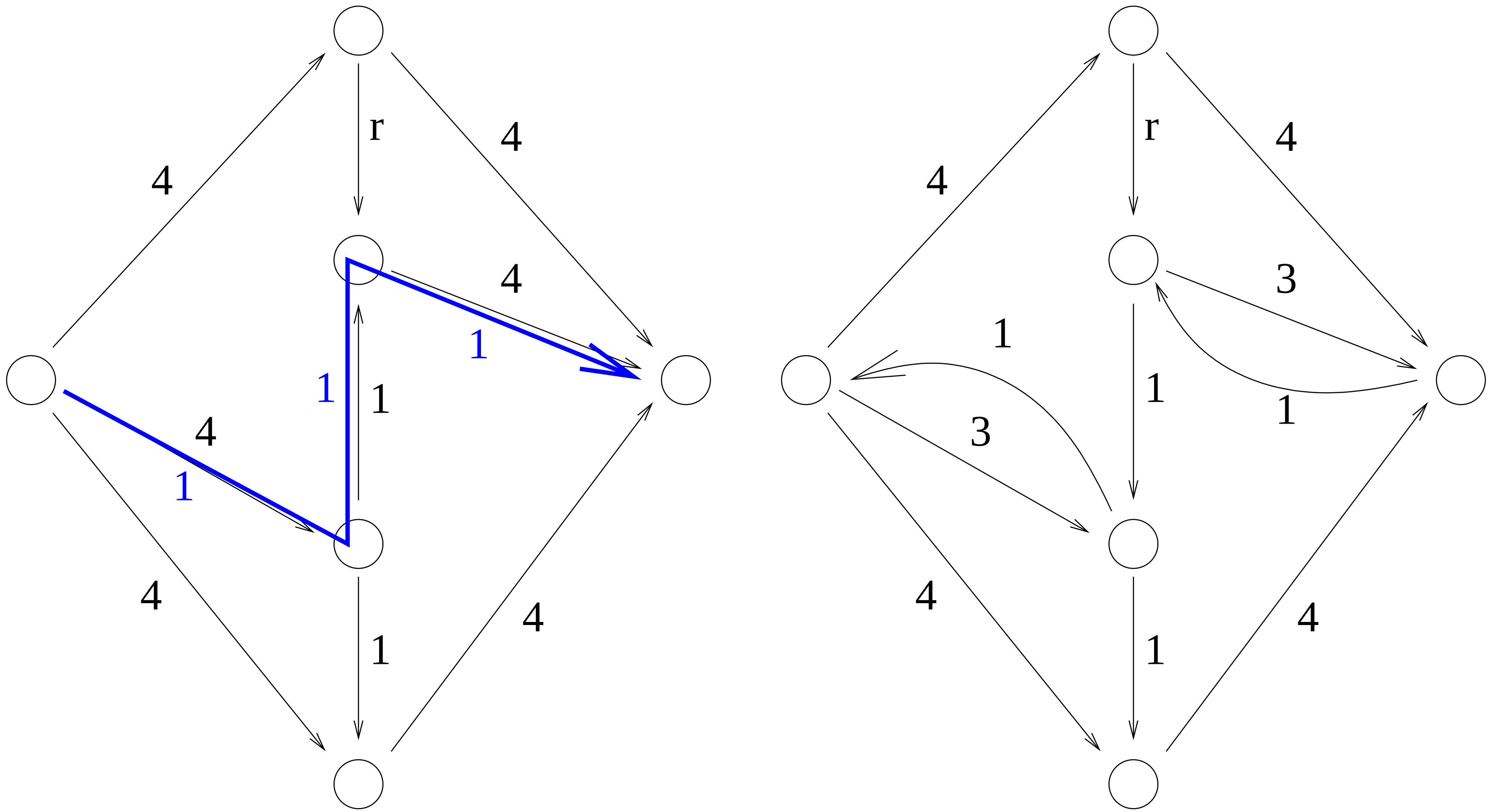
An Even Worse Example for Ford Fulkerson

[U. Zwick, TCS 148, p. 165–170, 1995]



An Even Worse Example for Ford Fulkerson

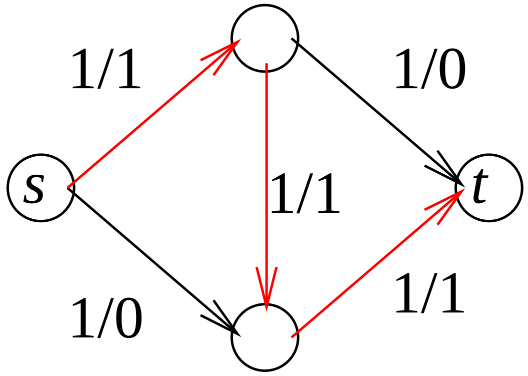
[U. Zwick, TCS 148, p. 165–170, 1995]



Blocking Flows

f_b is a blocking flow in H if

$$\forall \text{paths } p = \langle s, \ldots, t \rangle : \exists e \in p : f_b(e) = \text{cap}(e)$$



Dinitz Algorithm

Function DinitzMaxFlow($G = (V,E), s, t,$ cap $: E \to \mathbb{R}) : E \to \mathbb{R}$

$f := 0$

while $\exists$path $p = (s, \ldots, t)$ in G_f **do**

$d = G_f.reverseBFS(t) \quad : V \to \mathbb{N}$

$L_f = (V, \{(u,v) \in E_f : d(v) = d(u) - 1\})$ // layer graph

find a blocking flow f_b in L_f

augment f+=f_b

return f

Function blockingFlow$(H = (V,E)) : E \to \mathbb{R}$

p=$\langle s \rangle$: Path v=NodeRef : p.last()

f_b:= 0

loop // Round

if $v = t$ **then** // breakthrough

δ:= $\min\{\mathsf{cap}(e) - f_b(e) : e \in p\}$

foreach $e \in p$ **do**

$f_b(e)$+=δ

if $f_b(e) = \mathsf{cap}(e)$ **then** remove e from E

p:= $\langle s \rangle$

else if $\exists e = (v,w) \in E$ **then** p.pushBack(w) // extend

else if $v = s$ **then return** f_b // done

else delete the last edge from p in p and E // retreat

Blocking Flows Analysis 1

- running time is $\#_{extends} + \#_{retreats} + n \cdot \#_{breakthroughs}$
- $\#_{breakthroughs} \leq m$, since at least one edge is saturated
- $\#_{retreats} \leq m$, since one edge is removed
- $\#_{extends} \leq \#_{retreats} + n \cdot \#_{breakthroughs}$, since a retreat cancels one extend and a breakthrough cancels n extends

time is $O(m + nm) = O(nm)$

Blocking Flows Analysis 2

Unit capacities:

breakthroughs saturates all edges on p, i.e., amortized constant cost per edge.

time $O(m+n)$

Blocking Flows Analysis 3

Dynamic trees: breakthrough (!), retreat, extend in time $\mathrm{O}(\log n)$

time $O((m+n)\log n)$

Theory alert: In practice, this seems to be slower (few breakthroughs, many retreat, extend ops.)

Dinitz Analysis 1

Lemma 1. *$d(s)$ increases by at least one in each round.*

Beweis. not here □

Dinitz Analysis 2

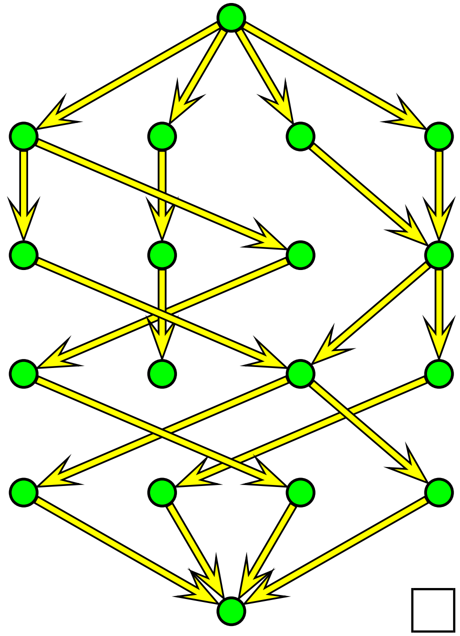
- $\leq n$ rounds
- time $\mathrm{O}(mn)$ each

time $\mathrm{O}\left(mn^2\right)$ (strongly polynomial)

time $\mathrm{O}(mn\log n)$ with dynamic trees

Dinitz Analysis 3

unit capacities



Lemma 2. *At most* $2\sqrt{m}$ *rounds:*

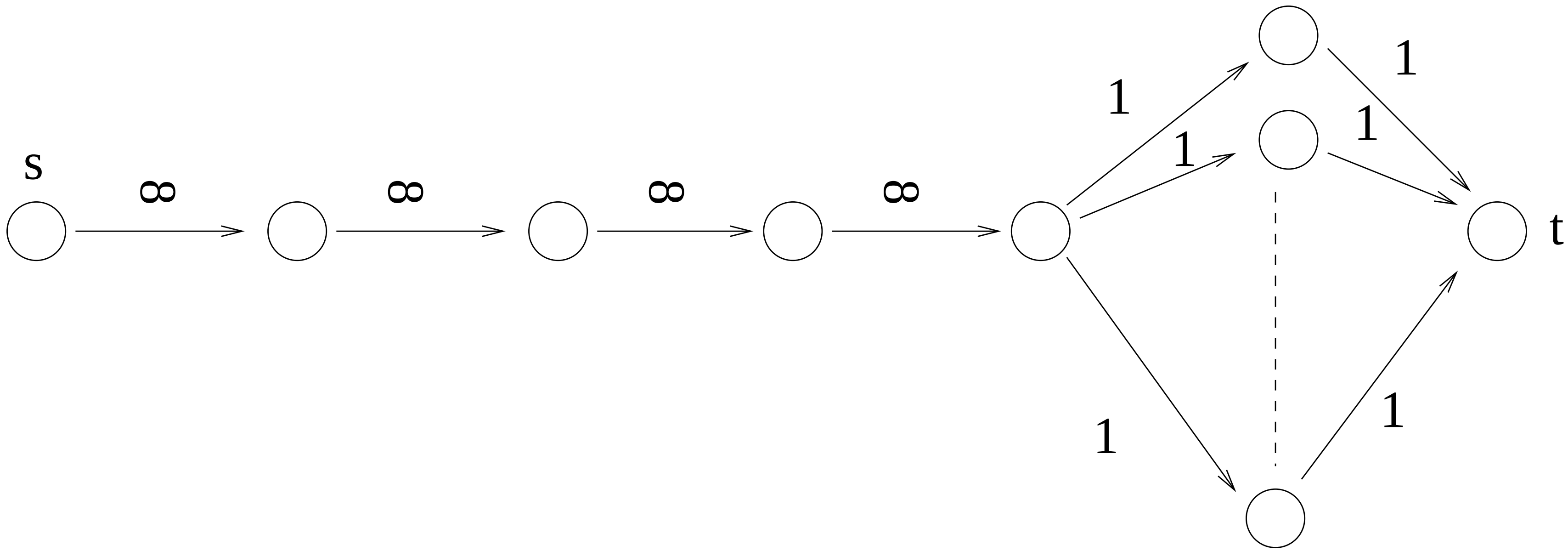
Beweis. Consider round $k = \sqrt{m}$.
Any s-t path contains $\geq k$ edges
FF can find $\leq m/k = \sqrt{m}$ augmenting paths $\square$

Total time: $\mathrm{O}((m+n)\sqrt{m})$

more detailed analysis: $\mathrm{O}\left(m \min\left\{m^{1/2}, n^{2/3}\right\}\right)$

$\forall v \in V$: $\min\{\text{indegree}(v), \text{outdegree}(v)\} = 1$: time:
$\mathrm{O}((m+n)\sqrt{n})$

Disadvantage of augmenting paths algorithms



Preflow-Push Algorithms

Preflow f: a flow where the flow conservation constraint is relaxed to $\text{excess}(v) \geq 0$.

Procedure $\text{push}(e = (v, w), \delta)$

  assert $\delta > 0$

  assert residual capacity of $e \geq \delta$

  assert $\text{excess}(v) \geq \delta$

  $\text{excess}(v) -= \delta$

  if $f(e) > 0$ **then** $f(e) += \delta$

  else $f(\text{reverse}(e)) -= \delta$

Level Function

Idea: make progress by pushing towards t

Maintain

an approximation $d(v)$ of the BFS distance from v to t in G_f.

invariant $d(t) = 0$

invariant $d(s) = n$

invariant $\forall (v,w) \in E_f : d(v) \leq d(w) + 1$ // no steep edges

Edge directions of $e = (v,w)$

steep: $d(w) < d(v) - 1$

downward: $d(w) < d(v)$

horizontal: $d(w) = d(v)$

upward: $d(w) > d(v)$

Procedure genericPreflowPush(G=(V,E), f)

forall $e=(s,v)\in E$ **do** push$(e,$cap$(e))$ // saturate

$d(s):=n$

$d(v):=0$ for all other nodes

while $\exists v\in V\setminus\{s,t\}$: excess$(v)>0$ **do** // active node

if $\exists e=(v,w)\in E_f : d(w)<d(v)$ **then** // eligible edge

choose some $\delta\leq\min\{$excess$(v),$ resCap$(e)\}$

push(e,δ) // no new steep edges

else $d(v)$++ // relabel. No new steep edges

Obvious choice for $\delta:\delta=\min\{$excess$(v),$ resCap$(e)\}$

Saturating push: $\delta=$ resCap(e)

nonsaturating push: $\delta<$ resCap(e)

To be filled in: How to select active nodes and eligible edges?

Lemma 3.

$\forall$ *active nodes* v : excess$(v) > 0 \Rightarrow \exists$ *path* $\langle v, \ldots, s\rangle \in G_f$

Intuition: what got there can always go back.

Beweis. $S := \left\{u \in V : \exists \text{ path } \langle v, \ldots u\rangle \in G_f\right\}$, $T := V \setminus S$. Then

$$\sum_{u \in S} excess(u) = \sum_{e \in E \cap (T \times S)} f(e) - \sum_{e \in E \cap (S \times T)} f(e),$$

$\forall (u,w) \in E_f : u \in S \Rightarrow w \in S$ by Def. of G_f, S

$\Rightarrow \forall e = (u,w) \in E \cap (T \times S) : f(e) = 0$ Otherwise $(w,u) \in E_f$

Hence, $\sum_{u \in S} excess(u) \leq 0$

One the negative excess of s can outweigh excess$(v) > 0$.

Hence $s \in S$. $\square$

Lemma 3.

$\forall$ *active nodes* v : excess$(v) > 0 \Rightarrow \exists$ *path* $\langle v, \ldots, s \rangle \in G_f$

Intuition: what got there can always go back.

Beweis. $S := \{u \in V : \exists \text{ path } \langle v, \ldots u \rangle \in G_f\}$, $T := V \setminus S$. Then

$$\sum_{u \in S} excess(u) = \sum_{e \in E \cap (T \times S)} f(e) - \sum_{e \in E \cap (S \times T)} f(e),$$

$\forall (u, w) \in E_f : u \in S \Rightarrow w \in S$ by Def. of G_f, S

$\Rightarrow \forall e = (u, w) \in E \cap (T \times S) : f(e) = 0$ Otherwise $(w, u) \in E_f$

Hence, $\sum_{u \in S} excess(u) \leq 0$

One the negative excess of s can outweigh excess$(v) > 0$.

Hence $s \in S$. $\square$

Lemma 4.

$\forall v \in V : d(v) < 2n$

Beweis. Suppose v is lifted to $d(v) = 2n$.

By Lemma 3, there is a (simple) path p to s in G_f.

p has at most $n-1$ nodes

$d(s) = n$.

Hence $d(v) < 2n$. Contradiction. $\square$

Partial Correctness

Lemma 5. *When* genericPreflowPush *terminates f is a maximal flow.*

Beweis.
f is a flow since $\forall v \in V \setminus \{s,t\} : \text{excess}(v) = 0$.

To show that f is maximal, it suffices to show that
$\nexists$ path $p = \langle s,\ldots,t\rangle \in G_f$ (Max-Flow Min-Cut Theorem):
Since $d(s) = n$, $d(t) = 0$, p would have to contain steep edges.
That would be a contradiction. $\square$

Lemma 6. *# Relabel operations* $\leq 2n^2$

Beweis. $d(v) \leq 2n$, i.e., v is relabeled at most $2n$ time.
Hence, at most $|V| \cdot 2n = 2n^2$ relabel operations. $\square$

Lemma 7. *# saturating pushes $\leq nm$*

Beweis.

We show that there are at most n sat. pushes over any edge $e = (v, w)$.

A saturating push(e, δ) removes e from E_f.

Only a push on (w, v) can reinsert e into E_f.

For this to happen, w must be lifted at least two levels.

Hence, at most $2n/2 = n$ saturating pushes over (v, w)

$\square$

Lemma 8. *# nonsaturating pushes* $= \mathrm{O}\left(n^2 m\right)$
if $\delta = \min\left\{\mathsf{excess}(v), \mathsf{resCap}(e)\right\}$
for arbitrary node and edge selection rules.
(arbitrary-preflow-push)

Beweis. $\Phi := \sum_{\{v : v \text{ is active}\}} d(v).$ (Potential)

$\Phi = 0$ initially and at the end (no active nodes left!)

Operation	$\Delta(\Phi)$	How many times?	Total effect
relabel	1	$\leq 2n^2$	$\leq 2n^2$
saturating push	$\leq 2n$	$\leq nm$	$\leq 2n^2 m$
nonsaturating push	≤ -1		

$\Phi \geq 0$ always. $\square$

Searching for Eligible Edges

Every node v maintains a currentEdge pointer to its sequence of outgoing edges in G_f.

invariant no edge $e = (v, w)$ to the left of currentEdge is eligible

Reset currentEdge at a relabel $(\leq 2n\times)$

Invariant cannot be violated by a push over a reverse edge $e' = (w, v)$ since this only happens when e' is downward, i.e., e is upward and hence not eligible.

Lemma 9.
Total cost for searching $\leq \sum_{v \in V} 2n \cdot \text{degree}(v) = 4nm = \mathrm{O}(nm)$

Satz 10. *Arbitrary Preflow Push finds a maximum flow in time* $O(n^2 m)$.

Beweis.

Lemma 5: partial correctness

Initialization in time $O(n+m)$.

Maintain set (e.g., stack, FIFO) of active nodes.

Use reverse edge pointers to implement push.

Lemma 6: $2n^2$ relabel operations

Lemma 7: nm saturating pushes

Lemma 8: $O(n^2 m)$ nonsaturating pushes

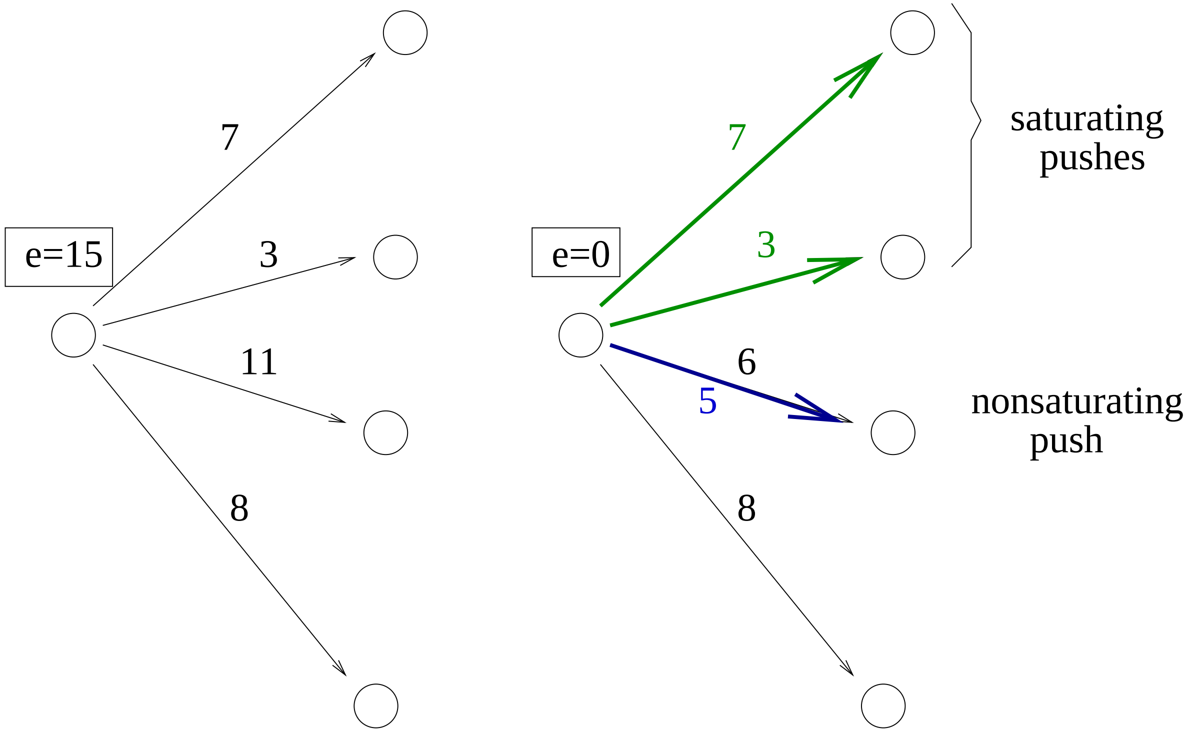
Lemma 9: $O(nm)$ search time for eligible edges

Total time $O(n^2 m)$ □

FIFO Preflow push

Examine active nodes in first-in, first-out order

Node examination = sequence of saturating pushes followed by nonsaturating push or relabel



FIFO Preflow push

Examine active nodes in first-in, first-out order

Node examination = sequence of saturating pushes followed by nonsaturating push or relabel

Partition sequence of examinations into phases

Phase 1 = examination of nodes that became active in preprocessing

Phase 2 = examination of nodes that became active in phase 1

...

Phase i = examination of nodes that became active in phase $i-1$

At most n nonsaturating pushes per phase. But how many phases?

FIFO Preflow push

Node examination = sequence of saturating pushes followed by nonsaturating push or relabel

Phase i = examination of nodes that became active in phase $i-1$

$\Phi := \max_{\{v:v \text{ is active}\}} d(v).$ (Potential)

$\Phi = 0$ initially and at the end (no active nodes left!)

$\Phi \geq 0$ always.

$\Phi = n$ after preprocessing (pushing flow out of s): $d(s) = n$

How does Φ change in a phase?

FIFO Preflow push

$$\Phi := \max_{\{v: v \text{ is active}\}} d(v). \qquad \text{(Potential)}$$

- At least one relabel operation in a phase:

  $\Delta(\Phi) \leq$ maximum increase of any distance label

  Total increase in Φ over all phases $\leq 2n^2$

- No relabel operation:

  all excess moves to nodes with smaller distance labels

  Φ decreases by at least 1

There cannot be more than $2n^2 + n$ phases before $\Phi = 0$

No active nodes left $\Rightarrow$ FIFO-PP runs in $\mathrm{O}\left(n^3\right)$

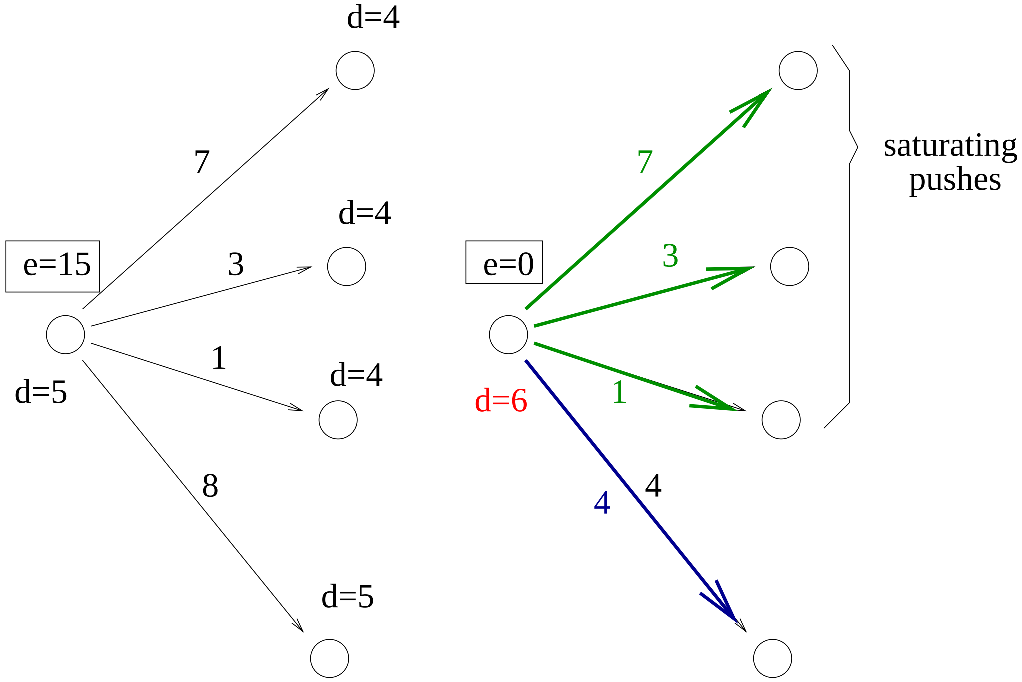
Modified FIFO preflow push

FIFO: examine nodes in FIFO order

MFIFO: when a node is relabeled, put it first in the list

MFIFO does not leave a node until all excess is pushed out of it

(FIFO leaves a node when it is relabeled)



Bucket-Queues

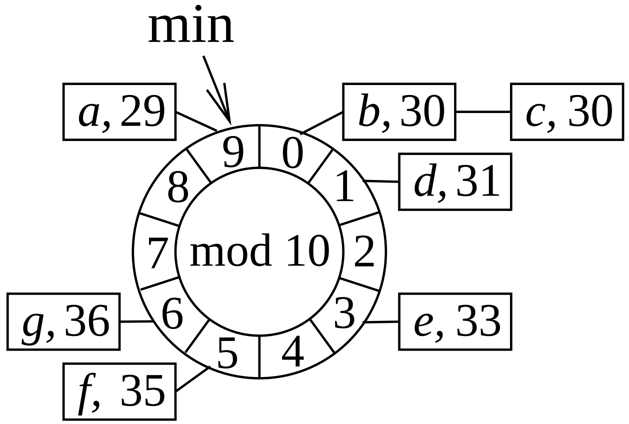
Eine Bucket-Queue ist ein kreisförmiges Array B von $C+1$ doppelt gelinkten Listen

Ein Knoten mit aktuelle Distanz $d[v]$ wird gespeichert bei Index

$$d[v] \quad \bmod (C+1)$$

Alle Knoten im gleichen Bucket haben die gleiche Distanz $d[v]$!

Bucket queue with C = 9



Content=
<(a,29), (b,30), (c,30), (d,31)
(e,33), (f,35), (g,36)>

Highest Level Preflow Push

Always select active nodes that maximize $d(v)$

Use bucket priority queue (insert, increaseKey, deleteMax)

not monotone (!) but relabels “pay” for scan operations

Lemma 11. *At most $n^2\sqrt{m}$ nonsaturating pushes.*

Beweis. later □

Satz 12. *Highest Level Preflow Push finds a maximum flow in time* $\mathrm{O}\left(n^2\sqrt{m}\right)$.

Proof of Lemma 11

$K := \sqrt{m}$ tuning parameter

$d'(v) := \frac{|\{w : d(w) \leq d(v)\}|}{K}$ scaled number of dominated nodes

$\Phi := \sum_{\{v:v \text{ is active}\}} d'(v).$ (Potential)

$d^* := \max\{d(v) : v \text{ is active}\}$ (highest level)

phase:= all pushes between two consecutive changes of d^*

expensive phase: more than K pushes

cheap phase: otherwise

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together
2. $\Phi \geq 0$ always, $\Phi \leq n^2/K$ initially (obvious)
3. a relabel or saturating push increases Φ by at most n/K.
4. a nonsaturating push does not increase Φ.
5. an expensive phase with $Q \geq K$ nonsaturating pushes decreases Φ by at least Q.

Operation	Amount
Relabel	$2n^2$
Sat.push	nm

Lemma 6+Lemma 7+2.+3.+4.:$\Rightarrow$

total possible decrease $\leq (2n^2 + nm)\frac{n}{K} + \frac{n^2}{K}$

This $+5.$ $:\leq \frac{2n^3+n^2+mn^2}{K}$ nonsaturating pushes in expensive phases

This $+1.$ $:\leq \frac{2n^3+n^2+mn^2}{K} + 4n^2K = \mathrm{O}\left(n^2\sqrt{m}\right)$ nonsaturating pushes overall for $K = \sqrt{m}$ $\square$

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together

We first show that there are at most $4n^2$ phases

(changes of $d^* = \max\{d(v) : v \text{ is active}\}$).

$d^* = 0$ initially, $d^* \geq 0$ always.

Only relabel operations increase d^*, i.e.,

$\leq 2n^2$ increases by Lemma 6 and hence

$\leq 2n^2$ decreases

$\leq 4n^2$ changes overall

By definition of a cheap phase, it has at most K pushes.

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together
2. $\Phi \geq 0$ always, $\Phi \leq n^2/K$ initially (obvious)
3. a relabel or saturating push increases Φ by at most n/K.

Let v denote the relabeled or activated node.

$$d'(v) := \frac{|\{w : d(w) \leq d(v)\}|}{K} \leq \frac{n}{K}$$

A relabel of v can increase only the d'-value of v.

A saturating push on (u,w) may activate only w.

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together
2. $\Phi \geq 0$ always, $\Phi \leq n^2/K$ initially (obvious)
3. a relabel or saturating push increases Φ by at most n/K.
4. a nonsaturating push does not increase Φ.

v is deactivated (excess(v) is now 0)

w may be activated

but $d'(w) \leq d'(v)$ (we do not push flow away from the sink)

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together
2. $\Phi \geq 0$ always, $\Phi \leq n^2/K$ initially (obvious)
3. a relabel or saturating push increases Φ by at most n/K.
4. a nonsaturating push does not increase Φ.
5. an expensive phase with $Q \geq K$ nonsaturating pushes decreases Φ by at least Q.

During a phase d^* remains constant

Each nonsat. push decreases the number of nodes at level d^*

Hence, $|\{w : d(w) = d^*\}| \geq K$ during an expensive phase

Each nonsat. push across (v, w) decreases Φ by

$\geq d'(v) - d'(w) \geq |\{w : d(w) = d^*\}|/K \geq K/K = 1$ ■

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together
2. $\Phi \geq 0$ always, $\Phi \leq n^2/K$ initially (obvious)
3. a relabel or saturating push increases Φ by at most n/K.
4. a nonsaturating push does not increase Φ.
5. an expensive phase with $Q \geq K$ nonsaturating pushes decreases Φ by at least Q.

Operation	Amount
Relabel	$2n^2$
Sat.push	nm

Lemma 6+Lemma 7+2.+3.+4.:$\Rightarrow$

total possible decrease $\leq (2n^2 + nm)\frac{n}{K} + \frac{n^2}{K}$

This $+5.$ $:\leq \frac{2n^3+n^2+mn^2}{K}$ nonsaturating pushes in expensive phases

This $+1.$ $:\leq \frac{2n^3+n^2+mn^2}{K} + 4n^2K = \mathrm{O}\left(n^2\sqrt{m}\right)$ nonsaturating pushes overall for $K = \sqrt{m}$ $\square$

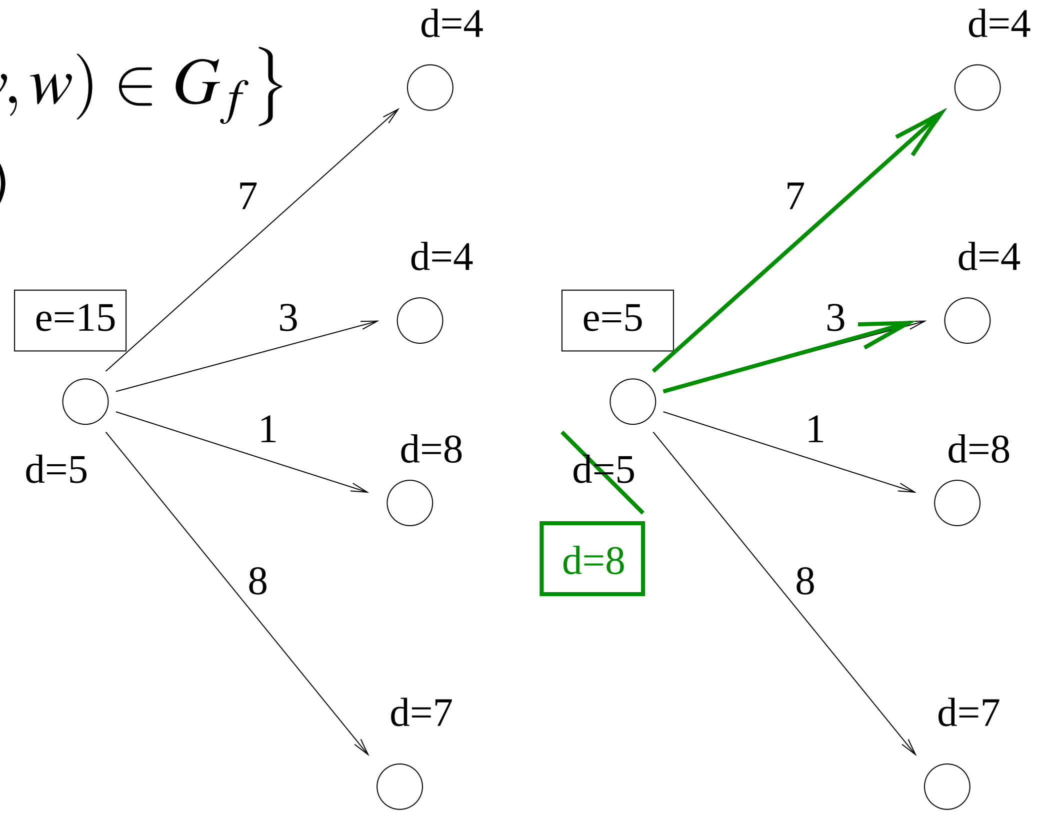
Heuristic Improvements

Naive algorithm has best case $\Omega\left(n^2\right)$. Why? We can do better.

aggressive local relabeling:

$d(v) := 1 + \min\left\{d(w) : (v,w) \in G_f\right\}$

(like a sequence of relabels)



Heuristic Improvements

Naive algorithm has best case $\Omega\left(n^2\right)$. Why?

We can do better.

aggressive local relabeling: $d(v) := 1 + \min\left\{d(w) : (v, w) \in G_f\right\}$

(like a sequence of relabels)

global relabeling: (initially and every $\mathrm{O}(m)$ edge inspections):

$d(v) := G_f$.reverseBFS(t) for nodes that can reach t in G_f.

Special treatment of nodes with $d(v) \geq n$. (Returning flow is easy)

Gap Heuristics. No node can connect to t across an empty level:

if $\{v : d(v) = i\} = \emptyset$ **then foreach** v with $d(v) > i$ **do** $d(v) := n$

Experimental results

We use four classes of graphs:

- Random: n nodes, $2n+m$ edges; all edges (s,v) and (v,t) exist
- Cherkassky and Goldberg (1997) (two graph classes)
- Ahuja, Magnanti, Orlin (1993)

Timings: Random Graphs

Rule	BASIC	HL	LRH	GRH	GAP	LEDA
FF	5.84	6.02	4.75	0.07	0.07	—
	33.32	33.88	26.63	0.16	0.17	—
HL	6.12	6.3	4.97	0.41	0.11	0.07
	27.03	27.61	22.22	1.14	0.22	0.16
MF	5.36	5.51	4.57	0.06	0.07	—
	26.35	27.16	23.65	0.19	0.16	—

$n \in \{1000, 2000\}, m = 3n$

FF=FIFO node selection, HL=hightest level, MF=modified FIFO

HL= $d(v) \geq n$ is special,

LRH=local relabeling heuristic, GRH=global relabeling heuristics

Timings: CG1

Rule	BASIC	HL	LRH	GRH	GAP	LEDA
FF	3.46	3.62	2.87	0.9	1.01	—
	15.44	16.08	12.63	3.64	4.07	—
HL	20.43	20.61	20.51	1.19	1.33	0.8
	192.8	191.5	193.7	4.87	5.34	3.28
MF	3.01	3.16	2.3	0.89	1.01	—
	12.22	12.91	9.52	3.65	4.12	—

$n \in \{1000, 2000\}, m = 3n$

FF=FIFO node selection, HL=hightest level, MF=modified FIFO

HL= $d(v) \geq n$ is special,

LRH=local relabeling heuristic, GRH=global relabeling heuristics

Timings: CG2

Rule	BASIC	HL	LRH	GRH	GAP	LEDA
FF	50.06	47.12	37.58	1.76	1.96	—
	239	222.4	177.1	7.18	8	—
HL	42.95	41.5	30.1	0.17	0.14	0.08
	173.9	167.9	120.5	0.36	0.28	0.18
MF	45.34	42.73	37.6	0.94	1.07	—
	198.2	186.8	165.7	4.11	4.55	—

$n \in \{1000, 2000\}, m = 3n$

FF$=$FIFO node selection, HL$=$hightest level, MF$=$modified FIFO

HL$= d(v) \geq n$ is special,

LRH$=$local relabeling heuristic, GRH$=$global relabeling heuristics

Timings: AMO

Rule	BASIC	HL	LRH	GRH	GAP	LEDA
FF	12.61	13.25	1.17	0.06	0.06	—
	55.74	58.31	5.01	0.1399	0.1301	—
HL	15.14	15.8	1.49	0.13	0.13	0.07
	62.15	65.3	6.99	0.26	0.26	0.14
MF	10.97	11.65	0.04999	0.06	0.06	—
	46.74	49.48	0.1099	0.1301	0.1399	—

$n \in \{1000, 2000\}, m = 3n$

FF=FIFO node selection, HL=hightest level, MF=modified FIFO

HL= $d(v) \geq n$ is special,

LRH=local relabeling heuristic, GRH=global relabeling heuristics

Asymptotics, $n \in \{5000, 10000, 20000\}$

Gen	Rule	GRH			GAP			LEDA		
rand	FF	0.16	0.41	1.16	0.15	0.42	1.05	—	—	—
	HL	1.47	4.67	18.81	0.23	0.57	1.38	0.16	0.45	1.09
	MF	0.17	0.36	1.06	0.14	0.37	0.92	—	—	—
CG1	FF	3.6	16.06	69.3	3.62	16.97	71.29	—	—	—
	HL	4.27	20.4	77.5	4.6	20.54	80.99	2.64	12.13	48.52
	MF	3.55	15.97	68.45	3.66	16.5	70.23	—	—	—
CG2	FF	6.8	29.12	125.3	7.04	29.5	127.6	—	—	—
	HL	0.33	0.65	1.36	0.26	0.52	1.05	0.15	0.3	0.63
	MF	3.86	15.96	68.42	3.9	16.14	70.07	—	—	—
AMO	FF	0.12	0.22	0.48	0.11	0.24	0.49	—	—	—
	HL	0.25	0.48	0.99	0.24	0.48	0.99	0.12	0.24	0.52
	MF	0.11	0.24	0.5	0.11	0.24	0.48	—	—	—

Minimum Cost Flows

Define $G = (V, E)$, f, excess, and cap as for maximum flows.

Let $c : E \to \mathbb{R}$ denote the edge costs.

Consider supply : $V \to \mathbb{R}$ with $\sum_{v \in V} \mathrm{supply}(v) = 0$. A negative supply is called a demand.

Objective: minimize $c(f) := \sum_{e \in E} f(e)c(e)$

subject to

$\forall v \in V : \mathrm{excess}(v) = -\mathrm{supply}(v)$ flow conservation constraints

$\forall e \in E : f(e) \leq \mathrm{cap}(e)$ capacity constraints

The Cycle Canceling Algorithm for Min-Cost Flow

Residual cost: Let $e = (v, w) \in G_f$, $e' = (w, v)$.

$c_f(e) = -c(e')$ if $e' \in E$, $f(e') > 0$, $c_f(e) = c(e)$ otherwise.

Lemma 13. *A feasible flow is optimal iff*

$\nexists$ *cycle* $C \in G_f : c_f(C) < 0$

Beweis. not here $\square$

A pseudopolynomial Algorithm:

$f :=$ any feasible flow// Exercise: solve this problem using maximum flows

invariant f is feasible

while $\exists$ cycle $C : c_f(C) < 0$ **do** augment flow around C

Korollar 14 (Integrality Property:)**.** *If all edge capacities are integral then there exists an integral minimum cost flow.*

Finding a Feasible Flow

set up a maximum flow network G^* starting with the min cost flow problem G:

- Add a vertex s
- $\forall v \in V$ with supply$(v) > 0$, add edge (s,v) with cap. supply(v)
- Add a vertex t
- $\forall v \in V$ with supply$(v) < 0$, add edge (v,t) with cap. $-$supply(v)
- find a maximum flow f in G^*

f saturates the edges leaving $s \Rightarrow f$ is feasible for G

otherwise there cannot be a feasible flow f' because f' could easily be converted into a flow in G^* with larger value.

Better Algorithms

Satz 15. *The min-cost flow problem can be solved in time* $\mathrm{O}\left(mn\log n + m^2\log\max_{e\in E}\mathsf{cap}(e)\right)$.

For details take the courses in optimization or network flows.

Special Cases of Min Cost Flows

Transportation Problem: $\forall e \in E$: cap$(e) = \infty$

Minimum Cost Bipartite Perfect Matching:

A transportation problem in a bipartite graph $G = (A \cup B, E \subseteq A \times B)$

with

supply$(v) = 1$ for $v \in A$,

supply$(v) = -1$ for $v \in B$.

An integral flow defines a matching

Reminder: $M \subseteq E$ is a matching if (V, M) has maximum degree one.

A rule of Thumb: If you have a combinatorial optimization problem. Try to formulate it as a shortest path, flow, or matching problem. If this fails its likely to be NP-hard.

Maximum Weight Matching

Generalization of maximum cardinality matching. Find a matching $M^* \subseteq E$ such that $w(M^*) := \sum_{e \in M^*} w(e)$ is maximized

Applications: Graph partitioning, selecting communication partners...

Satz 16. *A maximum weighted matching can be found in time* $\mathrm{O}(nm + n^2 \log n)$. *[Gabow 1992]*

Approximate Weighted Matching

Satz 17. *There is an* $\mathrm{O}(m)$ *time algorithm that finds a matching of weight at least* $\max_{matching M} w(M)/2$. *[Drake Hougardy 2002]*

The algorithm is a $1/2$-approximation algorithm.

Approximate Weighted Matching Algorithm

$M' := \emptyset$

invariant M' is a set of simple paths

while $E \neq \emptyset$ **do** // find heavy simple paths

  select any $v \in V$ with degree$(v) > 0$ // select a starting node

  while degree$(v) > 0$ **do** // extend path greedily

    $(v, w) :=$ heaviest edge leaving v // (*)

    $M' := M' \cup \{(v, w)\}$

    remove v from the graph

    $v := w$

return any matching $M \subseteq M'$ with $w(M) \geq w(M')/2$

// one path at a time, e.g., look at the two ways to take every other edge.

Proof of Approximation Ratio

Let M^* denote a maximum weight matching.

It suffices to show that $w(M') \geq w(M^*)$.

Assign each edge to that incident node that is deleted first.

All $e^* \in M^*$ are assigned to different nodes.

Consider any edge $e^* \in M^*$ and assume it is assigned to node v.

Since e^* is assigned to v, it was available in line **(*)**.

Hence, there is an edge $e \in M_{01}$ assigned to v with $w(e) \geq w(e^*)$.