Algorithmen II

Peter Sanders

Übungen:
Moritz Laupichler, Nikolai Maas

Institut für Theoretische Informatik

Web:
algo2.iti.kit.edu/AlgorithmenII_WS23.php
5 Maximum Flows and Matchings

[mit Kurt Mehlhorn, Rob van Stee]

Folien auf Englisch

Literatur:
http://www.mpi-inf.mpg.de/~mehlhorn/ftp/LEDAbook/Graph_alg.ps

[Ahuja, Magnanti, Orlin, Network Flows, Prentice Hall, 1993]
Definitions: Network

- 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: Flows

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

$\forall v \in V \setminus \{s, t\}$: total incoming flow = total outgoing flow

Value of a flow $\text{val}(f) =$

- total outgoing flow from $s = \text{total flow going into } t$

Goal: find a flow with maximum value

10

s

10

10

8

4

8

4

10

12

4

2

6

12

t
Definitions: (Minimum) $s$-$t$ Cuts

An $s$-$t$ cut is partition of $V$ into $S$ and $T$ with $s \in S$ and $t \in T$.

The capacity of this cut is:

$$\sum \{c(u,v) : u \in S, v \in T\}$$
Duality Between Flows and Cuts

Theorem: [Elias/Feinstein/Shannon, Ford/Fulkerson 1956]

Value of an $s$-$t$ max-flow $=$ minimum capacity of an $s$-$t$ cut.

Proof: later
Applications

- Oil pipes
- Traffic flows on highways
- **Image Processing** [http://vision.csd.uwo.ca/maxflow-data](http://vision.csd.uwo.ca/maxflow-data)
  - segmentation
  - stereo processing
  - multiview reconstruction
  - surface fitting
- disk/machine/tanker **scheduling**
- matrix **rounding**
- ...
Current Research Challenge: AI versus Optimal Algorithms

Many image processing applications are currently taken over by deep convolutional neural networks.

+ Often better results
+ No ad-hoc definitions of $s$, $t$, $c$
  – “Optimality” is thrown over board
  – Lots of training examples needed

Is there a middle way?
Learn $s$, $t$, $c$ then optimize?
Applications in our Group

- multicasting using network coding
- balanced $k$ partitioning
- disk scheduling
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

<table>
<thead>
<tr>
<th>Year</th>
<th>Author</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1956</td>
<td>Ford-Fulkerson</td>
<td>$O(mnU)$</td>
</tr>
<tr>
<td>1969</td>
<td>Edmonds-Karp</td>
<td>$O(m^2n)$</td>
</tr>
<tr>
<td>1970</td>
<td>Dinic</td>
<td>$O(mn^2)$</td>
</tr>
<tr>
<td>1973</td>
<td>Dinic-Gabow</td>
<td>$O(mn \log U)$</td>
</tr>
<tr>
<td>1974</td>
<td>Karzanov</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>1977</td>
<td>Cherkassky</td>
<td>$O(n^2 \sqrt{m})$</td>
</tr>
<tr>
<td>1980</td>
<td>Galil-Naamad</td>
<td>$O(mn \log^2 n)$</td>
</tr>
<tr>
<td>1983</td>
<td>Sleator-Tarjan</td>
<td>$O(mn \log n)$</td>
</tr>
<tr>
<td>1986</td>
<td>Goldberg-Tarjan</td>
<td>$O(mn \log(n^2/m))$</td>
</tr>
<tr>
<td>1987</td>
<td>Ahuja-Orlin</td>
<td>$O(mn + n^2 \log U)$</td>
</tr>
</tbody>
</table>

$n =$ number of nodes  
$m =$ number of arcs  
$U =$ largest capacity
<table>
<thead>
<tr>
<th>Year</th>
<th>Author</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1987</td>
<td>Ahuja-Orlin-Tarjan</td>
<td>$O(mn \log (2 + n\sqrt{\log U}/m))$</td>
</tr>
<tr>
<td>1990</td>
<td>Cheriyan-Hagerup-Mehlhorn</td>
<td>$O(n^3/\log n)$</td>
</tr>
<tr>
<td>1990</td>
<td>Alon</td>
<td>$O(mn + n^{8/3} \log n)$</td>
</tr>
<tr>
<td>1992</td>
<td>King-Rao-Tarjan</td>
<td>$O(mn + n^{2+\epsilon})$</td>
</tr>
<tr>
<td>1993</td>
<td>Philipps-Westbrook</td>
<td>$O(mn \log n/ \log \frac{m}{n} + n^2 \log^{2+\epsilon} n)$</td>
</tr>
<tr>
<td>1994</td>
<td>King-Rao-Tarjan</td>
<td>$O(mn \log n/ \log \frac{m}{n \log n})$ if $m \geq 2n \log n$</td>
</tr>
<tr>
<td>1997</td>
<td>Goldberg-Rao</td>
<td>$O(\min{m^{1/2}, n^{2/3}}m\log (n^2/m) \log U)$</td>
</tr>
<tr>
<td>2014</td>
<td>Lee-Sidford</td>
<td>$O(m\sqrt{n} \log^2 U)$</td>
</tr>
<tr>
<td>2020</td>
<td>v. d. Brand et al.</td>
<td>$O(m + n^{3/2} \log U \log^? m)$</td>
</tr>
<tr>
<td>2021</td>
<td>Gao-Liu-Peng</td>
<td>$O(m^{3/2} - \frac{1}{328} \log U \log^? m)$</td>
</tr>
<tr>
<td>2022</td>
<td>v.d. Brand et al.</td>
<td>$O(m^{3/2} - \frac{1}{58} \log U \log^? m)$</td>
</tr>
<tr>
<td>2022</td>
<td>Chen, Kyng et al.</td>
<td>$O(m^{1+o(1)} \log U)$</td>
</tr>
</tbody>
</table>
Augmenting Paths (Rough Idea)

Find a path from \( s \) to \( t \) such that each edge has some spare capacity.

On this path, saturate the edge with the smallest spare capacity.

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

A typical greedy algorithm.
Example
Example
Example

![Graph Diagram]

0 —> 0 —> 10 —> 2

6 —> 4 —> 4 —> 4 —> 0

+4
Example

```
0 -> 1
10 -> 2
4 -> 10
0 -> 0
```
Example

are we done?
Example

\[
\begin{array}{c}
0 \\
10 \\
2 \\
8 \\
0
\end{array}
\quad
\begin{array}{c}
2 \\
2 \\
8 \\
0 \\
4
\end{array}
\quad
\begin{array}{c}
2 \\
2 \\
4 \\
4 \\
0
\end{array}
\quad
\begin{array}{c}
0 \\
12 \\
6 \\
2 \\
0
\end{array}
\]
Residual Graph

Given, network $G = (V, E, c)$, flow $f$

Residual graph $G_f = (V, E_f, c^f)$. For each $e \in E$ we have

$$
\begin{cases}
  e \in E_f \text{ with } c^f_e = c_e - f(e) & \text{if } f(e) < c(e) \\
  e^{rev} \in E_f \text{ with } c^{f}_{e^{rev}} = f(e) & \text{if } f(e) > 0
\end{cases}
$$
Augmenting Paths

Find a path $p$ from $s$ to $t$ such that each edge $e$ has nonzero residual capacity $c^f_e$

$$\Delta f := \min_{e \in p} c^f_e$$

foreach $(u, v) \in p$ do

if $(u, v) \in E$ then $f_{(u,v)} + = \Delta f$

else $f_{(v,u)} - = \Delta f$
**Ford Fulkerson Algorithm**

**Function** \( \text{FFMaxFlow}(G = (V, E), s, t, c : E \to \mathbb{N}) : E \to \mathbb{N} \)

\[
f := 0
\]

\[
\text{while } \exists \text{path } p = (s, \ldots, t) \text{ in } G_f \text{ do}
\]

\[
\text{augment } f \text{ along } p
\]

\[
\text{return } f
\]

\[
\text{time } O(m\text{val}(f))
\]
Ford Fulkerson – Correctness

“Clearly” FF computes a feasible flow $f$. (Invariant)

Todo: flow value is maximal

At termination: no augmenting paths in $G_f$ left.

Consider cut $(S, T := V \setminus S)$ with

$S := \{v \in V : v \text{ reachable from } s \text{ in } G_f\}$
A Basic Observations

**Lemma 1:** For any cut \((S, T)\):

\[
\text{val}(f) = \sum_{e \in E \cap S \times T} f_e - \sum_{e \in E \cap T \times S} f_e.
\]
Ford Fulkerson – Correctness

Todo: \( \text{val}(f) \) is maximal when no augmenting paths in \( G_f \) left.

Consider cut \((S, T := V \setminus S)\) with
\[
S := \{v \in V : v \text{ reachable from } s \text{ in } G_f \}.
\]

Observation: \( \forall (u, v) \in E \cap T \times S : f(u, v) = 0 \)
otherwise \( c^f(v, u) > 0 \) contradicting the definition of \( S \).

\[
\text{val}(f) = \sum_{e \in E \cap S \times T} f_e - \sum_{e \in E \cap T \times S} f_e \quad \text{Lemma 1}
\]

\[
= \sum_{e \in E \cap S \times T} f_e \quad \text{Observation above}
\]

\[
= \sum_{e \in E \cap S \times T} c_{(u,v)} = (S, T) \text{ cut capacity}
\]

see next slide
Max-Flow-Min-Cut theorem

**Theorem:** $\text{Max-flow} = \text{min-cut}$

**Proof:**

obvious: \(\text{any-flow} \leq \text{max-flow} \leq \text{min-cut} \leq \text{any-cut}\)

previous slide:

\((S, T)\) flow = \((S, T)\) cut capacity

\(\Rightarrow\)

\((S, T)\) flow = max-flow = min-cut
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

Let \( r = \frac{\sqrt{5} - 1}{2} \).

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.
Blocking Flows

$f_b$ is a blocking flow in $H$ if

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

**Function** DinitzMaxFlow($G = (V, E), s, t, c : E \rightarrow \mathbb{N}) : E \rightarrow \mathbb{N}$

$f := 0$

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

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

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

find a blocking flow $f_b$ in $L_f$

augment $f += f_b$

**return** $f$
Dinitz – Correctness

analogous to Ford-Fulkerson
Example

Graph diagram with nodes labeled a, b, c, d, s, and t. Edges are labeled with capacities and costs.

Unused flows: 2, 4, 6, 10
Used flows: 2, 4, 6, 10
Saturation flows: 2, 4, 6, 10

Costs: 2, 4, 6, 10
Computing Blocking Flows

Idea: repeated DFS for augmenting paths
(not using DFS algorithm schema)
Function blockingFlow($L_f = (V, E)$) : $E \rightarrow \mathbb{N}$

$p = \langle s \rangle$ : Path; $f_b = 0$ : Flow

loop

\[ \nu := p.last() \]

if $\nu = t$ then

\[ \delta := \min \{ c(e) - f_b(e) : e \in p \} \]

foreach $e \in p$ do

\[ f_b(e) += \delta \]

if $f_b(e) = c(e)$ then remove $e$ from $E$

$p := \langle s \rangle$

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

else if $\nu = s$ then return $f_b$ // done

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

Graph with nodes labeled a, b, c, d, s, t. Edges include:
- Extend from s to a with weight 3 and weight 2.
- Extend from a to c with weight 4.
- Extend from c to b with weight 4.
- Extend from b to d with weight 4.
- Extend from d to c with weight 8.
- Extend from s to t with weight 0.
- Breakthrough from t to b with weight 112.
- Retreat from a to s with weight 8.
- Extend from b to a with weight 2.
Example
Blocking Flows Analysis 1

- running time $\#_{\text{extends}} + \#_{\text{retreats}} + n \cdot \#_{\text{breakthroughs}}$

- $\#_{\text{breakthroughs}} \leq m$  \quad \text{--} \quad \geq 1 \text{ edge is saturated}$

- $\#_{\text{retreats}} \leq m$  \quad \text{--} \quad \text{one edge is removed}$

- $\#_{\text{extends}} \leq \#_{\text{retreats}} + n \cdot \#_{\text{breakthroughs}}$  \quad \text{--} \quad \text{a retreat cancels 1 extend, a breakthrough cancels} \leq n \text{ extends}$

time is $O(m + nm) = O(nm)$
Blocking Flows Analysis 2

Unit capacities:

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

time $O(m + n)$
Blocking Flows Analysis 3

If we use a dynamic tree data structure:
breakthrough (!), retreat, extend is possible in time $O(\log n)$

⇒

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.

Proof. not here
Dinitz Analysis 2

- \( \leq n \) rounds
- time \( O(mn) \) each

Time \( O(mn^2) \) (strongly polynomial)

Time \( O(mn \log n) \) with dynamic trees
Dinitz Analysis 3 – Unit Capacities

Lemma 2. At most $2\sqrt{m}$ BF computations:

Proof. Consider iteration $k = \sqrt{m}$. Cut in layergraph induces cut in residual graph of capacity at most $\sqrt{m}$. At most $\sqrt{m}$ additional phases.

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

more detailed analysis: $O\left(\min\left\{m^{1/2}, n^{2/3}\right\}\right)$
Dinitz Analysis 4 – Unit Networks

Unit capacity \( + \ \forall v \in V : \min \{ \text{indegree}(v), \text{outdegree}(v) \} = 1 : \)

time: \( O((m + n)\sqrt{n}) \)
Matching

$M \subseteq E$ is a matching in the undirected graph $G = (V, E)$ iff $(V, M)$ has maximum degree $\leq 1$.

$M$ is maximal if $\nexists e \in E \setminus M : M \cup \{e\}$ is a matching.

$M$ has maximum cardinality if $\nexists$ matching $M' : |M'| > |M|$
Maximum Cardinality Bipartite Matching

in \((L \cup R, E)\). Model as a unit network maximum flow problem

\[
\left(\{s\} \cup L \cup R \cup \{t\}, \{(s, u) : u \in L\} \cup E \cup \{(v, t) : v \in R\}\right)
\]

Dinitz algorithm yields \(O((n + m)\sqrt{n})\) algorithm
Similar Performance for Weighted Graphs?

**time:** $O\left(m \min \left\{ m^{1/2}, n^{2/3} \right\} \log C \right)$ [Goldberg Rao 97]

**Problem:** Fat edges between layers ruin the argument

Idea: *scale* a parameter $\Delta$ from small to large.
Contract SCCs of fat edges (capacity $> \Delta$)

**Experiments** [Hagerup, Sanders Träff 98]:
Sometimes best algorithm usually slower than *preflow push*
Disadvantage of augmenting paths algorithms

\[ S \xrightarrow{\infty} \cdots \xrightarrow{\infty} \cdots \xrightarrow{\infty} \cdot \xrightarrow{1} t \]
Preflow-Push Algorithms

Preflow $f$: a flow where the flow conservation constraint is relaxed to

$$\text{excess}(v) := \sum_{(u,v) \in E} f_{u,v} - \sum_{(v,w) \in E} f_{v,w} \geq 0.$$

$v \in V \setminus \{s,t\}$ is active iff $\text{excess}(v) > 0$

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

assert $\delta > 0$ \land $\text{excess}(v) \geq \delta$

assert residual capacity of $e \geq \delta$

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

$\text{excess}(w) += \delta$

if $e$ is reverse edge then $f(\text{reverse}(e)) -= \delta$

else $f(e) += \delta$
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\((e, c(e))\)  
\(d(s) := n\)  
\(d(v) := 0\) for all other nodes

while \(\exists v \in V \setminus \{s, t\}: \text{excess}(v) > 0\) do  
  if \(\exists e = (v, w) \in E_f: d(w) < d(v)\) then  
    choose some \(\delta \leq \min\\{\text{excess}(v), c_e^f\}\) 
    push\((e, \delta)\)  
  else \(d(v)++\)

Obvious choice for \(\delta\): \(\delta = \min\\{\text{excess}(v), c_e^f\}\)

saturating push: \(\delta = c_e^f\)

nonsaturating push: \(\delta < c_e^f\)

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

Graph:
- Nodes: s, d, cap, f, excess, t
- Edges:
  - s to d: 6
  - d to cap: 10
  - cap to f: 10
  - f to excess: 10
  - excess to t: 12
  - s to t: 10, 10
  - d to t: 4
  - cap to t: 0
  - f to t: 8
  - excess to t: 4
Example

```
Example

Graph:

- **s** (source) to **d** (cap) with **6** capacity
- **d** to **cap** with **10** capacity
- **cap** to **f** (excess) with **10** capacity
- **s** to **10** with **10** capacity
- **1** to **10** with **4** capacity
- **12** to **t** (sink) with **10** capacity
- **t** to **0** with **8** capacity
```
Example

Graph:

- Source node: $s$
- Sink node: $t$
- Additional nodes: $d$, $cap$, $f$, $excess$

- Edges:
  - $s$ to $d$: capacity 10, excess 6
  - $s$ to $cap$: capacity 10, excess 6
  - $s$ to $f$: capacity 10, excess 6
  - $d$ to $t$: capacity 10, excess 1
  - $cap$ to $t$: capacity 10, excess 1
  - $f$ to $t$: capacity 10, excess 1
  - $d$ to $cap$: capacity 4
  - $cap$ to $f$: capacity 4
  - $f$ to $excess$: capacity 4
  - $t$ to $excess$: capacity 8
Example
Example

```
\begin{center}
\begin{tikzpicture}
\node[vertex, fill=white] (s) at (0,0) {$s$};
\node[vertex, fill=white] (d) at (2,2) {$d$};
\node[vertex, fill=white] (cap) at (3,1) {$\text{cap}$};
\node[vertex, fill=white] (f) at (1,0) {$f$};
\node[vertex, fill=white] (excess) at (4,1) {$\text{excess}$};
\node[vertex, fill=white] (t) at (5,0) {$t$};
\node[vertex, fill=white] (1) at (2,2) {};\node[vertex, fill=white] (2) at (4,2) {};\node[vertex, fill=white] (3) at (4,0) {};\node[vertex, fill=white] (4) at (0,2) {};\node[vertex, fill=white] (5) at (2,0) {};\node[vertex, fill=white] (6) at (0,0) {};\\
\draw[->, thick] (s) -- node[above] {10} (1);
\draw[->, thick] (1) -- node[above] {10} (2);
\draw[->, thick] (2) -- node[above] {4} (3);
\draw[->, thick] (3) -- node[above] {4} (4);
\draw[->, thick] (4) -- node[above] {4} (5);
\draw[->, thick] (5) -- node[above] {4} (s);
\draw[->, thick] (1) -- node[above] {6} (6);
\draw[->, thick] (6) -- node[above] {10} (f);
\draw[->, thick] (f) -- node[above] {8} (7);
\draw[->, thick] (7) -- node[above] {4} (2);
\draw[->, thick] (2) -- node[above] {4} (3);
\draw[->, thick] (3) -- node[above] {4} (4);
\draw[->, thick] (4) -- node[above] {4} (5);
\draw[->, thick] (5) -- node[above] {4} (s);
\end{tikzpicture}
\end{center}
```
Example
Example

\begin{tikzpicture}[node distance = 1.5cm, thick, main node/.style = {circle, draw, font = \sffamily\bfseries}]

\node[main node] (1) {1};
\node[main node] (2) [right of=1] {2};
\node[main node] (3) [below of=1] {7};
\node[main node] (4) [below right of=3] {1};
\node[main node] (5) [below right of=2] {1};
\node[main node] (6) [below left of=3] {d};
\node[main node] (7) [left of=6] {s};
\node[main node] (8) [right of=5] {t};

\path[
    edge node = \makebox[1cm][c]{\sf 10 10}]
    (1) edge (2);
\path[
    edge node = \makebox[1cm][c]{\sf 4}
]
    (1) edge (3);
\path[
    edge node = \makebox[1cm][c]{\sf 4}
]
    (3) edge (5);
\path[
    edge node = \makebox[1cm][c]{\sf 4}
]
    (5) edge (4);
\path[
    edge node = \makebox[1cm][c]{\sf 10 10}
]
    (7) edge (1);
\path[
    edge node = \makebox[1cm][c]{\sf 10}
]
    (7) edge (3);
\path[
    edge node = \makebox[1cm][c]{\sf 6}
]
    (7) edge (6);
\path[
    edge node = \makebox[1cm][c]{\sf 10}
]
    (6) edge (s);
\path[
    edge node = \makebox[1cm][c]{\sf 10}
]
    (4) edge (t);
\path[
    edge node = \makebox[1cm][c]{\sf 4}
]
    (4) edge (5);
\path[
    edge node = \makebox[1cm][c]{\sf 12}
]
    (5) edge (t);
\path[
    edge node = \makebox[1cm][c]{\sf 12}
]
    (2) edge (t);
\path[
    edge node = \makebox[1cm][c]{\sf 8}
]
    (t) edge (8);
\path[
    edge node = \makebox[1cm][c]{\sf 4}
]
    (8) edge (4);
\path[
    edge node = \makebox[1cm][c]{\sf 4}
]
    (4) edge (1);
\end{tikzpicture}
Example
Example

\[
\begin{array}{c}
\text{s} & \text{cap} & \text{d} & \text{f} & \text{excess} \\
6 & 10 & 10 & 8 & 4 \\
2 & 7 & 4 & 2 & 4 \\
2 & 1 & 4 & 8 & 4 \\
2 & 0 & & & \\
\end{array}
\]
Example

12 pushes in total
Partial Correctness

Lemma 3. When genericPreflowPush terminates, f is a maximal flow.

Proof.
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 there does not exist a 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 contradicts the invariant. \qed
Lemma 4. For any cut \((S, T)\),

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

Proof:

\[
\sum_{u \in S} \text{excess}(u) = \sum_{u \in S} \left( \sum_{(v, u) \in E} f((v, u)) - \sum_{(u, v) \in E} f((u, v)) \right)
\]

Contributions of edge \(e\) to sum:

- **S to T**: \(-f(e)\)
- **T to S**: \(f(e)\)
- **within S**: \(f(e) - f(e) = 0\)
- **within T**: 0

\[\blacksquare\]
Lemma 5.

∀ active nodes v : excess(v) > 0 ⇒ ∃ path ⟨v, ..., s⟩ ∈ G_f

Intuition: what got there can always go back.

Proof. S := {u ∈ V : ∃ path ⟨v, ...u⟩ ∈ G_f}, T := V \ S. Then

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

∀(u, w) ∈ E_f : u ∈ S ⇒ w ∈ S by Def. of G_f, S
⇒ ∀e = (u, w) ∈ E \ (T \times S) : f(e) = 0 Otherwise (w, u) ∈ E_f

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

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

Hence s ∈ S. □
Lemma 6.
\[ \forall v \in V : d(v) < 2n \]

Proof.
Suppose \( v \) is lifted to \( d(v) = 2n \).
By the Lemma 2, 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 (no steep edges). \( \square \)
Lemma 7. \# Relabel operations $\leq 2n^2$

Proof. $d(v) \leq 2n$, i.e., $v$ is relabeled at most $2n$ times. Hence, at most $|V| \cdot 2n = 2n^2$ relabel operations. \qed
Lemma 8. \# saturating pushes \( \leq nm \)

**Proof.**

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

A saturating push \( (e, \delta) \) 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 \( (v, w) \)
Lemma 9. \# nonsaturating pushes = O(n^2m)

if \( \delta = \min \left\{ \text{excess}(v), c^e_f \right\} \)

for arbitrary node and edge selection rules.

(arbitrary-preflow-push)

Proof. \( \Phi := \sum_{\{v:v \text{ is active}\}} d(v) \).  

(Potential)

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>( \Delta(\Phi) )</th>
<th>How many times?</th>
<th>Total effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>relabel</td>
<td>1</td>
<td>( \leq 2n^2 )</td>
<td>( \leq 2n^2 )</td>
</tr>
<tr>
<td>saturating push</td>
<td>( \leq 2n )</td>
<td>( \leq nm )</td>
<td>( \leq 2n^2m )</td>
</tr>
<tr>
<td>nonsaturating push</td>
<td>( \leq -1 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( \Phi \geq 0 \) always.
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

Invariant violations?

- □ relabel($v$)? Reset currentEdge
- □ relabel($w$)? No, no steep edges.
- □ push($w, v$)? $\Rightarrow (v, w)$ is upward

Lemma 10.

*Total cost for searching* $\leq \sum_{v \in V} 2n \cdot \text{degree}(v) = 4nm = O(nm)$
Theorem 11. Arbitrary Preflow Push finds a maximum flow in time $O(n^2m)$.

Proof.
Lemma 3: 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 7: $2n^2$ relabel operations
Lemma 8: $nm$ saturating pushes
Lemma 9: $O(n^2m)$ nonsaturating pushes
Lemma 10: $O(nm)$ search time for eligible edges

Total time $O(n^2m)$
FIFO Preflow push

Examine a node: Saturating pushes until nonsaturating push or relabel.

Examine all nodes in phases (or use FIFO queue).

Theorem: time $O(n^3)$

Proof: not here
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 12.** At most $n^2 \sqrt{m}$ nonsaturating pushes.

**Proof.** later

**Theorem 13.** Highest Level Preflow Push finds a maximum flow in time $O(n^2 \sqrt{m})$. 
Example

\[
\begin{array}{c}
\text{s} & \text{d} & \text{cap} & \text{f} & \text{excess} \\
6 & 10 & 10 & 10 & 10 \\
2 & 2 & 10 & 10 & 10 \\
10 & 10 & 10 & 10 & 10 \\
1 & 1 & 1 & 1 & 1 \\
12 & 12 & 12 & 12 & 12 \\
t & t & t & t & t \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Example

\[
\begin{array}{c}
\text{s} & \overset{10}{\rightarrow} & 2 & \overset{4}{\rightarrow} & 1 & \overset{12}{\rightarrow} & t \\
\overset{10}{\rightarrow} & \text{d} & \overset{4}{\leftarrow} & \text{f} & \overset{4}{\leftarrow} & \text{excess} & \overset{8}{\leftarrow} & 0 \\
\overset{6}{\leftarrow} & \text{cap} & \overset{10}{\leftarrow} & 2 & \overset{10}{\leftarrow} & 10 & \overset{4}{\leftarrow} & 10 \\
\end{array}
\]
Example

\[
\begin{array}{c}
\text{Example} \\
\end{array}
\]
Example

\begin{itemize}
  \item \text{source (s)}:
    \begin{itemize}
      \item \text{cap (d)}: 6
      \item excess (f): 10
    \end{itemize}
  \item \text{sink (t)}:
    \begin{itemize}
      \item \text{cap (d)}: 12
      \item excess (f): 0
    \end{itemize}
  \item \text{intermediate nodes}:
    \begin{itemize}
      \item 2: 10, 8
      \item 3: 12, 12
      \item 1: 4, 4
      \item 7: 4, 4
    \end{itemize}
\end{itemize}
Example
Example

9 pushes in total, 3 less than before
Proof of Lemma 12

\[ K := \sqrt{m} \]  
\[ d'(v) := \frac{|\{w : d(w) \leq d(v)\}|}{K} \]  
\[ \Phi := \sum_{\{v : v \text{ is active}\}} d'(v). \]  
\[ d^* := \max \{d(v) : v \text{ is active}\} \]  
\[ \text{phase} := \text{all pushes between two consecutive changes of } d^* \]  
\[ \text{expensive phase: more than } K \text{ pushes} \]  
\[ \text{cheap phase: otherwise} \]
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 \( \Phi \) by at most \( n/K \).

4. a nonsaturating push does not increase \( \Phi \).

5. an expensive phase with \( Q \geq K \) nonsaturating pushes decreases \( \Phi \) by at least \( Q \).

Lemma 7 + Lemma 8 + 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. \( \leq \frac{2n^3 + n^2 + mn^2}{K} + 4n^2K = O\left(n^2 \sqrt{m}\right) \) nonsaturating pushes overall for \( K = \sqrt{m} \)
Claims:

1. $\leq 4n^2 K$ 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 7 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. \( \leq 4n^2 K \) nonsaturating pushes in all cheap phases together

2. \( \Phi \geq 0 \) always, \( \Phi \leq n^2 / K \) initially \hspace{1cm} \text{(obvious)}

3. a relabel or saturating push increases \( \Phi \) by at most \( n / K \).

Let \( v \) denote the relabeled or activated node.

\[
d'(v) := \left| \{ w : d(w) \leq d(v) \} \right| \leq \frac{n}{K}
\]

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 \( \Phi \) by at most \( n/K \).

4. a nonsaturating push does not increase \( \Phi \).

\( v \) 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 $\Phi$ by at most $n/K$.

4. a nonsaturating push does not increase $\Phi$.

5. an expensive phase with $Q \geq K$ nonsaturating pushes decreases $\Phi$ by at least $Q$.

During a phase $d^*$ remains constant

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

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

Each nonsat. push across $(v, w)$ decreases $\Phi$ 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, $\Phi \leq \frac{n^2}{K}$ initially (obvious)

3. a relabel or saturating push increases $\Phi$ by at most $\frac{n}{K}$.

4. a nonsaturating push does not increase $\Phi$.

5. an expensive phase with $Q \geq K$ nonsaturating pushes decreases $\Phi$ by at least $Q$.

Lemma 7 + Lemma 8 + 2. + 3. + 4. $\Rightarrow$

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

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

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

pushFront after relabel.
pushBack when activated by a push
Heuristic Improvements

Naive algorithm needs $\Omega \left( n^2 \right)$ relabels even on a path graph. 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 $O(m)$ edge inspections):
$d(v) := G_f.\text{reverseBFS}(t)$ 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

<table>
<thead>
<tr>
<th>Rule</th>
<th>BASIC</th>
<th>Ln</th>
<th>LRH</th>
<th>GRH</th>
<th>GAP</th>
<th>LEDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF</td>
<td>5.84</td>
<td>6.02</td>
<td>4.75</td>
<td>0.07</td>
<td>0.07</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>33.32</td>
<td>33.88</td>
<td>26.63</td>
<td>0.16</td>
<td>0.17</td>
<td>—</td>
</tr>
<tr>
<td>HL</td>
<td>6.12</td>
<td>6.3</td>
<td>4.97</td>
<td>0.41</td>
<td>0.11</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>27.03</td>
<td>27.61</td>
<td>22.22</td>
<td>1.14</td>
<td>0.22</td>
<td>0.16</td>
</tr>
<tr>
<td>MF</td>
<td>5.36</td>
<td>5.51</td>
<td>4.57</td>
<td><strong>0.06</strong></td>
<td>0.07</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>26.35</td>
<td>27.16</td>
<td>23.65</td>
<td>0.19</td>
<td>0.16</td>
<td>—</td>
</tr>
</tbody>
</table>

\[ n \in \{1000,2000\}, m = 3n \]

FF=_FIFO node selection, HL=highest level, MF=modified FIFO

Ln= \( d(v) \geq n \) is special,

LRH=local relabeling heuristic, GRH=global relabeling heuristics
### Timings: CG1

<table>
<thead>
<tr>
<th>Rule</th>
<th>BASIC</th>
<th>Ln</th>
<th>LRH</th>
<th>GRH</th>
<th>GAP</th>
<th>LEDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF</td>
<td>3.46</td>
<td>3.62</td>
<td>2.87</td>
<td>0.9</td>
<td>1.01</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>15.44</td>
<td>16.08</td>
<td>12.63</td>
<td>3.64</td>
<td>4.07</td>
<td>—</td>
</tr>
<tr>
<td>HL</td>
<td>20.43</td>
<td>20.61</td>
<td>20.51</td>
<td>1.19</td>
<td>1.33</td>
<td><strong>0.8</strong></td>
</tr>
<tr>
<td></td>
<td>192.8</td>
<td>191.5</td>
<td>193.7</td>
<td>4.87</td>
<td>5.34</td>
<td>3.28</td>
</tr>
<tr>
<td>MF</td>
<td>3.01</td>
<td>3.16</td>
<td>2.3</td>
<td><strong>0.89</strong></td>
<td>1.01</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>12.22</td>
<td>12.91</td>
<td>9.52</td>
<td>3.65</td>
<td>4.12</td>
<td>—</td>
</tr>
</tbody>
</table>

\(n \in \{1000, 2000\}, m = 3n\)

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

Ln= \(d(v) \geq n\) is special,

LRH=local relabeling heuristic, GRH=global relabeling heuristics
## Timings: CG2

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

\( n \in \{1000, 2000\}, m = 3n \)

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

Ln= \( d(v) \geq n \) is special,

LRH=local relabeling heuristic, GRH=global relabeling heuristics
## Timings: AMO

<table>
<thead>
<tr>
<th>Rule</th>
<th>BASIC</th>
<th>Ln</th>
<th>LRH</th>
<th>GRH</th>
<th>GAP</th>
<th>LEDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF</td>
<td>12.61</td>
<td>13.25</td>
<td>1.17</td>
<td>0.06</td>
<td>0.06</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>55.74</td>
<td>58.31</td>
<td>5.01</td>
<td>0.1399</td>
<td>0.1301</td>
<td>—</td>
</tr>
<tr>
<td>HL</td>
<td>15.14</td>
<td>15.8</td>
<td>1.49</td>
<td>0.13</td>
<td>0.13</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>62.15</td>
<td>65.3</td>
<td>6.99</td>
<td>0.26</td>
<td>0.26</td>
<td>0.14</td>
</tr>
<tr>
<td>MF</td>
<td>10.97</td>
<td>11.65</td>
<td>0.04999</td>
<td>0.06</td>
<td>0.06</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>46.74</td>
<td>49.48</td>
<td>0.1099</td>
<td>0.1301</td>
<td>0.1399</td>
<td>—</td>
</tr>
</tbody>
</table>

\(n \in \{ 1000, 2000 \}, m = 3n\)

- FF=FIFO node selection, HL=highest level, MF=modified FIFO
- Ln= \(d(v) \geq n\) is special,
- LRH=local relabeling heuristic, GRH=global relabeling heuristics
Asymptotics, \( n \in \{5000, 10000, 20000\} \)

<table>
<thead>
<tr>
<th>Gen</th>
<th>Rule</th>
<th>GRH</th>
<th>GAP</th>
<th>LEDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>rand</td>
<td>FF</td>
<td>0.16</td>
<td>0.41</td>
<td>1.16</td>
</tr>
<tr>
<td></td>
<td>HL</td>
<td>1.47</td>
<td>4.67</td>
<td>18.81</td>
</tr>
<tr>
<td></td>
<td>MF</td>
<td>0.17</td>
<td>0.36</td>
<td>1.06</td>
</tr>
<tr>
<td>CG1</td>
<td>FF</td>
<td>3.6</td>
<td>16.06</td>
<td>69.3</td>
</tr>
<tr>
<td></td>
<td>HL</td>
<td>4.27</td>
<td>20.4</td>
<td>77.5</td>
</tr>
<tr>
<td></td>
<td>MF</td>
<td>3.55</td>
<td>15.97</td>
<td>68.45</td>
</tr>
<tr>
<td>CG2</td>
<td>FF</td>
<td>6.8</td>
<td>29.12</td>
<td>125.3</td>
</tr>
<tr>
<td></td>
<td>HL</td>
<td>0.33</td>
<td>0.65</td>
<td>1.36</td>
</tr>
<tr>
<td></td>
<td>MF</td>
<td>3.86</td>
<td>15.96</td>
<td>68.42</td>
</tr>
<tr>
<td>AMO</td>
<td>FF</td>
<td>0.12</td>
<td>0.22</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>HL</td>
<td>0.25</td>
<td>0.48</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>MF</td>
<td>0.11</td>
<td>0.24</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Recent AE Results on Max-Flow


- Much faster on many (relatively easy) real world instances (image processing, graph partitioning, . . . ) than preflow-push
- Worst case performance guarantee $O(mn^2)$ (as in Dinitz algorithm)
- Adaptable to dynamic scenarios
- Uses pseudoflows that allow excesses and deficits.

Open problem: close gaps between theory and practice!
Zusammenfassung Flows und Matchings I

- Natürliche Verallgemeinerung von kürzesten Wegen:
  ein Pfad \(\leadsto\) viele Pfade

- viele Anwendungen

- “schwierigste/allgemeinsten” Graph-Probleme, die sich mit kombinatorischen Algorithmen in Polynomialzeit lösen lassen

- Beispiel für nichttriviale Algorithmenanalyse

- Manchmal sind spezielle Probleminstanzfamilien beweisbar leichter (z.B. unit capacity, matchings)
Zusammenfassung Flows und Matchings II

- Entwurfstechnik: Algorithmeninvarianten relaxieren
  (augmenting paths ↛ Preflow-Push ↛ pseudoflows)

- Invarianten leiten Entwurf und Verständnis von Algorithmen

- Potentialmethode ($\neq$ Knotenpotentiale)

- Algorithm Engineering: practical case $\neq$ worst case.
  Heuristiken/Details/Eingabeeigenschaften wichtig

- Datenstrukturen: bucket queues, graph representation,
  (dynamic trees)