Algorithmen II

Peter Sanders, Christian Schulz, Simon Gog

Übungen:
Michael Axtmann

Institut für Theoretische Informatik, Algorithmik II

Web:
http://algo2.itl.kit.edu/AlgorithmenII_WS16.php
5 Maximum Flows and Matchings

[mit Kurz 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 $e = \text{capacity of } e$ (nonnegative!)
Definitions: Flows

- Flow = function $f_e$ on the edges, $0 \leq f_e \leq c_e \forall e$
  - $\forall v \in V \setminus \{s,t\}$: total incoming flow = total outgoing flow
- Value of a flow $\text{val}(f) = \text{total outgoing flow from } s = \text{total flow going into } t$
- Goal: find a flow with maximum value

![Graph with edges and capacities labeled]
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
- …
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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1956</td>
<td>Ford-Fulkerson</td>
<td>$O(mnU)$</td>
<td></td>
</tr>
<tr>
<td>1969</td>
<td>Edmonds-Karp</td>
<td>$O(m^2n)$</td>
<td></td>
</tr>
<tr>
<td>1970</td>
<td>Dinic</td>
<td>$O(mn^2)$</td>
<td></td>
</tr>
<tr>
<td>1973</td>
<td>Dinic-Gabow</td>
<td>$O(mn \log U)$</td>
<td>$n = \text{number of nodes}$</td>
</tr>
<tr>
<td>1974</td>
<td>Karzanov</td>
<td>$O(n^3)$</td>
<td>$m = \text{number of arcs}$</td>
</tr>
<tr>
<td>1977</td>
<td>Cherkassky</td>
<td>$O(n^2 \sqrt{m})$</td>
<td>$U = \text{largest capacity}$</td>
</tr>
<tr>
<td>1980</td>
<td>Galil-Naamad</td>
<td>$O(mn \log^2 n)$</td>
<td></td>
</tr>
<tr>
<td>1983</td>
<td>Sleator-Tarjan</td>
<td>$O(mn \log n)$</td>
<td></td>
</tr>
</tbody>
</table>

$n = \text{number of nodes}$

$m = \text{number of arcs}$

$U = \text{largest capacity}$
<table>
<thead>
<tr>
<th>Year</th>
<th>Author</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<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>
<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>$O(\min{m^{1/2}, n^{2/3}} m \log(n^2/m) \log \log n)$</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.
Example
Example
Example
Example
Example
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

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

Find a path \( p \) from \( s \) to \( t \) such that each edge \( e \) has nonzero residual capacity \( c_e^f \)

\[
\Delta f := \min \limits_{e \in p} c_e^f
\]

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

\[
f := 0
\]

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

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

\[\text{return } f\]

\[\text{time } O(mval(f))\]
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 \((S, V \setminus S)\) with
\[
S := \{ v \in V : v \text{ reachable from } s \text{ in } G_f \}
\]
Some Basic Observations

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

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

Lemma 2: \(\forall (u, v) \in E : c_{(u,v)}^f = 0 \Rightarrow f_{(v,u)} = 0\)
Ford Fulkerson – Correctness

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

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

Observation: \( \forall (u, v) \in E \cap S \times T : c^f_e = 0 \) and hence \( f(v,u) = 0 \)

Lemma 2.

Now, by Lemma 1,

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

\[= \sum_{e \in E \cap S \times T} f_e = \text{cut capacity} \]

\[\geq \text{max flow} \]

Corollary: 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\((G = (V, E), s, t, c : E \rightarrow \mathbb{N}) : E \rightarrow \mathbb{N}\)

\[
f := 0
\]

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

\[
d = G_f . \text{reverseBFS}(t) : V \rightarrow \mathbb{N}
\]

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

find a blocking flow \(f_b\) in \(L_f\)

augment \(f += f_b\)

\[
\text{return } f
\]
Dinitz – Correctness

analogous to Ford-Fulkerson
Example

Graph representation:

- Nodes: s, a, b, c, d, t
- Edges and weights:
  - s → b: 4
  - b → c: 2, 4
  - c → a: 10
  - a → d: 4
  - d → t: 4
  - t → d: 0
- Path weights:
  - s → t: 4 + 2 + 3 + 2 + 1 + 2 + 0 = 12
Computing Blocking Flows

Idee: wiederholte DFS nach augmentierenden Pfaden
Function blockingFlow($L_f = (V, E)$) : $E \rightarrow \mathbb{N}$

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

loop // Round

$v := p$.last()

if $v = t$ then // breakthrough

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

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 $v = s$ then return $f_b$ // done

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

\[ \text{extend} \quad 10 \quad \text{extend} \quad 10 \quad \text{extend} \quad 10 \quad \text{breakthrough} \]

\[ \text{extend} \quad 10 \quad \text{extend} \quad 10 \quad \text{extend} \quad 10 \quad \text{breakthrough} \]
Example

```
  3
  \( \ast \) s

  extend
  10

  2
  \( \ast \) b

  extend
  4

  4

  4

  2
  \( \ast \) e

  extend
  1

  8

  8

  0
  \( \ast \) t

  breakthrough
```

**Example**

- The example diagram shows the process of extending and retreating in an algorithm.
- Nodes are labeled with numbers and actions are indicated by arrows.
- The diagram illustrates the flow and decision points in the algorithmic process.

**Key Points**

- Each node represents a state or decision point.
- Arrows indicate the flow of the algorithm, with labels showing the number of steps or actions.
- The diagram highlights the concept of extending and retreating, which are crucial steps in many algorithms.

**Notes**

- Understanding the flow and decision points is essential for grasping the algorithm's logic.
- Practice with similar diagrams can enhance comprehension and problem-solving skills.

---

* Sanders: Algorithmen II - 8. September 2016

---
Example
Blocking Flows Analysis 1

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

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

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

□ $\#_{extends} \leq \#_{retreats} + n \cdot \#_{breakthroughs}$  
  - a retreat cancels 1 extend, a breakthrough cancels $\leq n$ 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 \( O(\log n) \) 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 $O(mn)$ 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:

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

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

\[
\text{time: } O((m + n)\sqrt{n})
\]
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 $\not\exists e \in E \setminus M : M \cup \{e\}$ is a matching.

$M$ has maximum cardinality if $\not\exists$ matching $M' : |M'| > |M|$
Maximum Cardinality Bipartite Matching

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

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

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

time: $O\left( m \min \left\{ m^{1/2}, n^{2/3} \right\} \log C \right)$ \cite{GoldbergRao97}

**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 \cite{HagerupSandersTraff98}:
Sometimes best algorithm usually slower than preflow push
Disadvantage of augmenting paths algorithms

\[ s \rightarrow \infty \rightarrow \infty \rightarrow \infty \rightarrow \infty \rightarrow \]
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 $e \geq \delta$

excess$(v) - = \delta$

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** \( \text{genericPreflowPush}(G=(V,E), f) \)

```
forall \( e = (s, v) \in E \) do push\( (e, c(e)) \) // saturate
\( d(s) := n \)
\( d(v) := 0 \) for all other nodes

while \( \exists v \in V \setminus \{s, t\} : \text{excess}(v) > 0 \) do // active node
    if \( \exists e = (v, w) \in E_f : d(w) < d(v) \) then // eligible edge
        choose some \( \delta \leq \min \{\text{excess}(v), c_e^f\} \)
        push\( (e, \delta) \) // no new steep edges
    else \( d(v)++ \) // relabel. No new steep edges
```

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
Example

![Graph Diagram]

- **s**
  - **d**: cap = 6
  - **excess**: 10

- **0**: 10
  - **f**: 4
  - **excess**: 0

- **t**: 12
  - **excess**: 0
  - 8
Example

![Graph Diagram](image-url)
Example

Graph showing a network flow problem with nodes labeled s, d, cap, f, excess, 1, 2, 4, 6, 8, 10, 12, and t. The edges have capacities and flow values indicated.
Example

![Diagram of a network flow problem]

- Source (s) to vertex 1 with capacity 10.
- Vertex 1 to vertex 2 with capacity 10.
- Vertex 2 to vertex 3 with capacity 10.
- Vertex 3 to vertex 4 with capacity 4.
- Vertex 4 to sink (t) with capacity 12.

Vertex labels:
- s: Source
- t: Sink
- d: Initial demand
- cap: Capacity
- f: Flow
- excess: Excess flow
Example
Example

\begin{tikzpicture}
  \node (s) at (0,0) [draw, circle] {s};
  \node (d) at (3,3) [draw, circle] {d cap f excess};
  \node (cap) at (3,0) [draw, circle] {cap f excess};
  \node (f) at (6,3) [draw, circle] {f excess};
  \node (t) at (9,0) [draw, circle] {t};
  \path[->, thick]
    (s) edge [below] node {6} (d)
    (s) edge [right] node {10} (cap)
    (d) edge [above] node {10} (cap)
    (cap) edge [right] node {10} (f)
    (cap) edge [above] node {10} (t)
    (f) edge [below] node {8} (t);
\end{tikzpicture}
Example
Example

\[
\begin{align*}
&\text{Source (s)} \quad \text{Cap (d)} \\
&\quad \quad 10 \quad 6 \\
&\text{Flow (f)} \\
&\quad \quad 10 \\
&\text{Sink (t)} \\
&\quad \quad 12 \\
&\text{Excess} \\
&\quad \quad 4 \\
\end{align*}
\]
Example

12 pushes in total
Partial Correctness

**Lemma 3.** 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 $\not\exists$ 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 4. For any cut \((S, T)\),

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

**Proof:**

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

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:** \(0\)
Lemma 5.

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

Intuition: what got there can always go back.

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

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

\( \forall (u, w) \in E_f : u \in S \Rightarrow w \in S \) \hspace{1cm} \text{by Def. of } G_f, S

\( \Rightarrow \forall e = (u, w) \in E \cap (T \times S) : f(e) = 0 \) \hspace{1cm} \text{Otherwise } (w, u) \in E_f

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

Only the negative excess of \( s \) can outweigh \( \text{excess}(v) > 0 \).

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

Beweis.
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 \( \leq 2n^2 \)

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

*Beweis.*

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

A saturating push \((e, \delta)\) 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 \(\delta = \min \{ \text{excess}(v), c^f_e \} \)

for arbitrary node and edge selection rules.

\((\text{arbitrary-preflow-push})\)

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

Reset currentEdge at a relabel

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 10.**

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

*Beweis.*

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 \(O(n^2m)\) \(\square\)
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.

Beweis. later

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

![Graph Image]
Example
Example
Example

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

\[ K := \sqrt{m} \] 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 \hspace{1cm} (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 $Q$.

\begin{tabular}{|l|l|}
\hline
Operation & Amount \\
\hline
Relabel & $2n^2$ \\
Sat.push & $nm$ \\
\hline
\end{tabular}

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 $K = \sqrt{m}$

$\square$
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 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. $\leq 4n^2K$ 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 $n/K$.

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 \((\text{obvious})\)

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 \hspace{1cm} (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 nodes at level \( d^* \)

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

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

\[ \geq d'(v) - d'(w) \geq \left| \{ w : d(w) = d^* \} \right|/K \geq K/K = 1 \]
Claims:

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

2. \( \Phi \geq 0 \) always, \( \Phi \leq n^2 / K \) initially \quad \text{(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.:\( \leq \frac{2n^3 + n^2 + mn^2}{K} + 4n^2 K = O(n^2 \sqrt{m}) \) 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:**

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

(like a sequence of relabels)
Heuristic Improvements

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

aggressive local relabeling: $d(v):= 1 + \min \{d(w) : (v,w) \in G_f\}$
(like a sequence of relabels)

global relabeling: (initially and every $O(m)$ 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

<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>0.06</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>0.8</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>0.89</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>0.15</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>HL</td>
<td>1.47</td>
<td>0.23</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>MF</td>
<td>0.17</td>
<td>0.14</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>CG1</td>
<td>3.6</td>
<td>3.62</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>MF</td>
<td>3.55</td>
<td>3.66</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>CG2</td>
<td>6.8</td>
<td>7.04</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>MF</td>
<td>3.86</td>
<td>3.9</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>AMO</td>
<td>0.12</td>
<td>0.11</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>HL</td>
<td>0.25</td>
<td>0.24</td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td>MF</td>
<td>0.11</td>
<td>0.11</td>
<td>—</td>
</tr>
</tbody>
</table>
Zusammenfassung Flows und Matchings

- Natürliche Verallgemeinerung von kürzesten Wegen: ein Pfad $\rightsquigarrow$ viele Pfade
- viele Anwendungen
- “schwierigste/allgemeinste” Graph-Probleme, die sich mit kombinatorischen Algorithmen in Polynomialzeit lösen lassen
- Beispiel für nichttriviale Algorithmenanalyse
- Potentialmethode ($\neq$ Knotenpotentiale)
- Algorithm Engineering: practical case $\neq$ worst case. Heuristiken/Details/Eingabeeigenschaften wichtig
- Datenstrukturen: bucket queues, graph representation, (dynamic trees)