Maximum Network Flow

Network: a directed graph, with two
distinguished vertices, a source $s$ and
a sink $t$, and a positive capacity $u(v,w)$
on each edge $(v,w)$.

A flow on a network: a nonnegative function $f$
on edges, bounded above by the capacities,
such that the total flow into any vertex
other than $s$ and $t$ equals the total flow out
Maximum flow: a flow that maximizes the net flow into the sink (which equals the net flow out of the source).

Problem: Find a maximum flow in a given network, as fast as possible.

\[ n = \# \text{vertices} \]

\[ m = \# \text{edges} \]

\[ U = \text{maximum edge capacity} \] (if capacities are integers)
Ford-Fulkerson Method

Residual edge: a pair \((v, w)\) such that

(i) \(f(v, w) < u(v, w)\): \(u_f(v, w) = u(v, w) - f(v, w)\)

or

(ii) \(f(w, v) > 0\): \(u_f(v, w) = f(w, v)\)

Residual network: the network of residual edges

Thm. A flow is maximum iff there is no path from \(s\) to \(t\) in the residual network (such a path is an augmenting path).
Ford-Fulkerson method:

repeat \{ 
  find an augmenting path
  augment flow
\}

Time: $O(nmU)$ (not polynomial, need not terminate if capacities are irrational)
(Bad) Example
Maximum Flow Problem

Network \( G = (V, E) \), source \( s \), sink \( t \)

edge capacities \( u(v,w) \) for \( (v,w) \in E \)

\(|V| = n \quad |E| = m \quad U = \max |u(v,w)|\)

Assume network is symmetric:

\( (v,w) \in E \quad \text{iff} \quad (w,v) \in E \)

Flow \( f : E \to \mathbb{R} \)

\( f(v,w) \leq u(v,w) \)

\( f(v,w) = -f(w,v) \)

\( e(w) = \sum_{w} f(v,w) = 0 \quad \forall \ W \in \{s,t\} \)

Objective: maximize \( e(t) \) \( (= -e(s)) \)
Edmonds & Karp: augment along shortest (fewest edges) paths: \( O(nm^2) \)

Dinitz: build shortest path subnetwork of residual network, find all augmenting paths of a given length at once: \( O(n^2m) \)

An edge \((v,w)\) is saturated if \( f(v,w) = u(v,w) \)

A blocking flow is a flow such that every path from \( s \) to \( t \) contains a saturated edge

Dinitz reduced the maximum flow problem to \( n \) blocking flow problems, each on an acyclic network.

Finding a blocking flow is easier than finding a maximum flow, at least on an acyclic network.
Edmonds & Karp: always augment along a shortest (fewest edges) path:

\[ O(m) \text{ time per path} \times O(m) \text{ paths per layer} \times O(n) \text{ path lengths} = O(nm^2) \text{ time} \]

Dinic: find all augmenting paths of a given length at once, in a phase:

\[ O(n) \text{ time per path} \times O(nm) \text{ paths} + O(m) \text{ time per phase} \times O(n) \text{ phases} = O(n^2m) \text{ time} \]
### Classical Algorithms

<table>
<thead>
<tr>
<th>Date</th>
<th>Discoverer</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1956</td>
<td>Ford &amp; Fulkerson</td>
<td>$O(nmW)$</td>
</tr>
<tr>
<td>1969</td>
<td>Edmonds &amp; Karp</td>
<td>$O(nm^3)$</td>
</tr>
<tr>
<td>1970</td>
<td>Dinic</td>
<td>$O(n^2m)$</td>
</tr>
<tr>
<td>1974</td>
<td>Karzanov</td>
<td>$O(n^3)^*$</td>
</tr>
<tr>
<td></td>
<td>(same bound by several others later)</td>
<td></td>
</tr>
<tr>
<td>1977</td>
<td>Cherkasky</td>
<td>$O(n^2m^{1.5})^{**}$</td>
</tr>
<tr>
<td>1978</td>
<td>Galil</td>
<td>$O(n^{5/3}m^{1/3})^*$</td>
</tr>
<tr>
<td>1978</td>
<td>Galil &amp; Naamad; Shiloach</td>
<td>$O(nm(\log n)^2)$</td>
</tr>
<tr>
<td>1980</td>
<td>Sleator &amp; Tarjan</td>
<td>$O(nm \log n)$</td>
</tr>
<tr>
<td>1983</td>
<td>Gabow</td>
<td>$O(nm \log W)$</td>
</tr>
</tbody>
</table>

*Forerunners of preflow push method*
Techniques

Iterative Improvement:
locally modify the current solution
to improve it

Successive Approximation:
solve successively closer approximations
of the original problem, using each
solution as a starting point for the
next problem

Data Structures:
represent relevant information
about the current flow in an
appropriate way
Preflow Push Approach (Goldberg)

Two ideas:

Make the basic steps in the computation smaller
(relax the flow conservation requirement)

Use a less global, more distributed approach to do the preprocessing associated with each phase

Main effect: simpler algorithms
Preflow (Karzanov): like a flow except that the total flow into a vertex can exceed the total flow out.

A vertex $v$ with extra incoming flow is active. The net incoming flow $e(v)$ is the excess of vertex $v$.

Idea: move flow excess toward sink along estimated shortest paths. Move excess that cannot reach the sink back to the source, also along estimated shortest paths.

To estimate path lengths: a valid labeling is an integer function $d$ on vertices such that:

(i) $d(t) = 0$
(ii) $d(s) = n$
(iii) $d(v) \leq d(w) + 1$ if $u_f(yw) > 0$

$d(v)$ is a lower bound on the minimum of distance to $t$, $n +$ distance to $s$. 

1. Saturate all edges leaving $s$. Choose initial $d$.
2. Repeat push and relabel steps in any order until no vertex is active.

push $(v, w)$:
- if $v$ is active, $u_f (v, w) > 0$, and $d(v) = d(w) + 1$
- then move $\min \{ e(v), u_f (v, w) \}$ units of flow from $v$ to $w$ (the push is saturating if $u_f (v, w)$ units are moved)

relabel $(v)$:
- if $v$ is active and for all $(w)$, $u_f (v, w) = 0$ or $d(v) \leq d(w)$
- then let $d(v) = \min \{ d(w) + 1 | u_f (v, w) > 0 \}$
Bounds

Every active vertex has a label of at most $2n-1$; there is always a residual path to $s$.

$\Rightarrow O(n^2)$ relabelings, taking $O(nm)$ time.

Between saturating pushes through the same edge, both ends of edge must be relabeled

$\Rightarrow O(nm)$ saturating pushes.

The heart of the analysis is in bounding the number of nonsaturating pushes.
Generic Bound: $O(n^2m)$

Pf. Define $\Phi = \sum_{v \text{ active}} d(v)$.

$0 \leq \Phi \leq 2n^2$. A nonsaturating push decreases $\Phi$ by one.

Increases to $\Phi$: $O(n^2)$ in total due to relabelings.

$O(n^3m)$ due to saturating pushes:

$O(n)$ per saturating push.

$\Rightarrow O(n^2m)$ nonsaturating pushes.
FIFO Method

Maintain a queue of active vertices.
Always push from the vertex on the front of the queue.
Add newly active vertices to the rear of the queue.

Analysis

Phases: phase 1 = processing of vertices originally on queue.
        phase i+1 = processing of vertices added to queue during phase i.

Only one nonsaturating push per vertex per phase
such a push reduces the excess to zero and removes the vertex from the queue.
$O(n^2)$ bound on # phases

Define $\Phi = \max_{v \text{ active}} d(v)$. $0 \leq \Phi \leq 2n$.

A phase reduces $\Phi$ by one unless a relabeling occurs.

All increase in $\Phi$ is due to relabelings, totals $O(n^2)$.

The number of phases in which $\Phi$ doesn't change is also $O(n^2)$.

$\Rightarrow O(n^2)$ total phases.

$\Rightarrow O(n^3)$ nonsaturating pushes.
Ahuja-Orlin Excess Scaling

Maintain $\Delta$, an upper bound on max excess

Maintain integrality of flow.

After each phase, replace $\Delta$ by $\Delta/2$.

Stop when $\Delta < 1$.

Push from a vertex $v$ of smallest $d(v)$ with $e(v) > \Delta/2$.

When pushing from $v$ to $w+\epsilon$, move

$$\min \{ e(v), u_x(v,w), \Delta - e(w) \}$$
Analysis

Each nonsaturating push moves at least $\Delta/2$ units of flow.

Let $\Phi = \sum_{v \text{ active}} e(v) d(v)/\Delta$

$0 \leq \Phi \leq 2n^2$

Each nonsaturating push decreases $\Phi$ by $\pi/2$.

Increases in $\Phi$: $O(n^2)$ associated with relabeling.

$O(n^3)$ per phase from change in $\Phi$.

$O(\log U)$ phases $\Rightarrow$

$O(n^2 \log U)$ nonsaturating pushes
saturating pushes = \( O(nm) \)

nonsaturating pushes = \( O(n^2 \log U) \)

Can these estimates be balanced?

Yes: change algorithm: make all pushes large enough by retaining enough excess to immediately saturate very-small-capacity edges.

\[ \# \text{ pushes} = O\left(n^{3/2} m^{1/2} (\log U)^{1/2}\right) \]

Cheriyan - Mehlhorn

What about relabeling time??
<table>
<thead>
<tr>
<th>Year</th>
<th>Discoverer</th>
<th>Time</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1985</td>
<td>Goldberg</td>
<td>$O(n^3)$</td>
<td>FIFO</td>
</tr>
<tr>
<td>1987</td>
<td>Cheriyan &amp; Maheshwari</td>
<td>$O(n^2 m^{1/2})$</td>
<td>Max Distance</td>
</tr>
<tr>
<td>1986</td>
<td>Goldberg &amp; Tarjan</td>
<td>$O(n m \log(n^2/m))$</td>
<td>FIFO + Trees</td>
</tr>
<tr>
<td>1986</td>
<td>Ahuja &amp; Orlin</td>
<td>$O(n m + n^2 \log U)$</td>
<td>Excess Scolling</td>
</tr>
<tr>
<td>1987</td>
<td>Ahuja &amp; Orlin</td>
<td>$O(n m + n^2 \log U)$</td>
<td></td>
</tr>
<tr>
<td>1987</td>
<td>Ahuja, Orlin, &amp; Tarjan</td>
<td>$O(n m \log (\frac{n (\log U)^{1/2}}{m} + 2))$</td>
<td>Excess Scolling + Trees</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(n m)$</td>
<td></td>
</tr>
<tr>
<td>1989</td>
<td>Cheriyan &amp; Hagerup (improved)</td>
<td>$O(n m + n^2 (\log n)^2)$</td>
<td>Excess Scolling + Randomization</td>
</tr>
<tr>
<td>1989</td>
<td>Cheriyan &amp; Hagerup + Mehlhorn</td>
<td>$O(n^3/\log n)$</td>
<td></td>
</tr>
</tbody>
</table>
Practice

Appropriate versions of the preflow push method are easy to implement and very fast in practice: 4-14 times faster than Dinic on reasonable classes of graphs.

Important heuristic: periodically compute tight distance labels using breadth-first search. (Otherwise the relabeling time is too high.)

The FIFO algorithm can be parallelized: push from all active vertices at once. It seems to give drastic speedups in practice.

Whether dynamic trees help on very large graphs has not yet been studied.