7. NETWORK FLOW I

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

Flow network

A flow network is a tuple \( G = (V, E, s, t, c) \).
- Digraph \((V, E)\) with source \( s \in V \) and sink \( t \in V \).
- Non-negative capacity \( c(e) \) for each \( e \in E \).

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

Minimum-cut problem

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

**Def.** Its capacity is the sum of the capacities of the edges from \( A \) to \( B \).

\[
\text{cap}(A, B) = \sum_{e \text{ out of } A} c(e)
\]
**Minimum-cut problem**

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

**Def.** Its capacity is the sum of the capacities of the edges from \(A\) to \(B\).

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

---

**Minimum-cut problem**

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

**Def.** Its capacity is the sum of the capacities of the edges from \(A\) to \(B\).

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

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

---

**Maximum-flow problem**

**Def.** An *s*-*t* flow (flow) \(f\) is a function that satisfies:

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

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

---

**Maximum-flow problem**

**Def.** An *s*-*t* flow (flow) \(f\) is a function that satisfies:

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

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

Def. An \textit{st-flow (flow)} \( f \) is a function that satisfies:
- For each \( e \in E \):
  \[ 0 \leq f(e) \leq c(e) \] \hspace{1cm} \text{[capacity]}
- For each \( v \in V - \{s, t\} \):
  \[ \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e) \] \hspace{1cm} \text{[flow conservation]}

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

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

Towards a max-flow algorithm

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

7. Network Flow I

\begin{itemize}
  \item max-flow and min-cut problems
  \item Ford-Fulkerson algorithm
  \item max-flow min-cut theorem
  \item capacity-scaling algorithm
  \item shortest augmenting paths
  \item blocking-flow algorithm
  \item simple unit-capacity networks
\end{itemize}

Towards a max-flow algorithm

Greedy algorithm.
- Start with \( f(e) = 0 \) for each edge \( e \in E \).
- Find an \textit{s-t} path \( P \) where each edge has \( f(e) < c(e) \).
- Augment flow along path \( P \).
- Repeat until you get stuck.
Towards a max-flow algorithm

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

\[
\text{flow network G and flow f}
\]

\[
\begin{array}{c}
\text{ending flow value = 16} \\
\text{flow network G and flow f}
\end{array}
\]

Towards a max-flow algorithm

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

\[
\text{but max-flow value = 19}
\]

\[
\text{flow network G and flow f}
\]
Why the greedy algorithm fails

Q. Why does the greedy algorithm fail?
A. Once greedy algorithm increases flow on an edge, it never decreases it.

Ex.
- The max flow is unique; flow on edge \((v, w)\) is zero.
- Greedy algorithm could choose \(s \rightarrow v \rightarrow w \rightarrow t\) for first augmenting path.

\[
\text{flow network } G
\]

Bottom line. Need some mechanism to “undo” bad decision.

Augmenting path

Def. An augmenting path is a simple \(s \rightarrow t\) path in the residual network \(G_f\).

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

Key property. Let \(f\) be a flow and let \(P\) be an augmenting path in \(G_f\). Then, after calling \textsc{Augment}, the resulting \(f'\) is a flow and 
\[
\text{val}(f') = \text{val}(f) + \text{bottleneck}(G_f, P).
\]

\[
\textsc{Augment} (f, c, P)
\]

\[
b \leftarrow \text{bottleneck capacity of path } P.
\]

\[
\text{FOREACH } e \in P
\]

\[
\text{IF } (e \in E) \ f[e] \leftarrow f[e] + b.
\]

\[
\text{ELSE } f[e_\text{reverse}] \leftarrow f[e_\text{reverse}] - b.
\]

\[
\text{RETURN } f.
\]

Residual network

Original edge. \(e = (u, v) \in E\).
- Flow \(f(e)\).
- Capacity \(c(e)\).

Reverse edge. \(e_\text{reverse} = (v, u)\).
- “Undo” flow sent.

Residual capacity.
\[
c_f(e) = \begin{cases} 
  c(e) - f(e) & \text{if } e \in E \\
  f(e) & \text{if } e_\text{reverse} \in E
\end{cases}
\]

Residual network. \(G_f = (V, E_f, s, t, c_f)\).
- \(E_f = \{e : f(e) < c(e)\} \cup \{e_\text{reverse} : f(e) > 0\}\).
- Key property: \(f'\) is a flow in \(G_f\) iff \(f + f'\) is a flow in \(G\).

Ford–Fulkerson algorithm

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

\[
\text{FORD–FULKERSON} (G)
\]

\[
\text{FOREACH } e \in E : f[e] \leftarrow 0.
\]

\[
G_f \leftarrow \text{residual network of } G \text{ with respect to } f.
\]

\[
\text{WHILE } (\text{there exists an } s \rightarrow t \text{ path } P \text{ in } G_f)
\]

\[
f \leftarrow \text{AUGMENT} (f, c, P).
\]

\[
\text{Update } G_f.
\]

\[
\text{RETURN } f.
\]
### 7. Network Flow I

- Max-flow and min-cut problems
- Ford–Fulkerson algorithm
- Max-flow min-cut theorem
- Capacity-scaling algorithm
- Shortest augmenting paths
- Blocking-flow algorithm
- Simple unit-capacity networks

#### Relationship between flows and cuts

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

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

Net flow across cut = 10 + 5 + 10 = 25
**Relationship between flows and cuts**

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

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

**Pf.**

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

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

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

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

---

**Certificate of optimality**

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

**Pf.**

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

---

**Max-flow min-cut theorem**

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

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

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

i. There exists a cut $(A, B)$ such that $\text{cap}(A, B) = \text{val}(f)$.
ii. $f$ is a max flow.
iii. There is no augmenting path with respect to $f$.

$[i \Rightarrow ii]$

- Suppose that $(A, B)$ is a cut such that $\text{cap}(A, B) = \text{val}(f)$.
- Then, for any flow $f'$: $\text{val}(f') \leq \text{cap}(A, B) = \text{val}(f)$.
- Thus, $f$ is a max flow. $lacklozenge$

---

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

**Pf.**

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

$$\leq \sum_{e \text{ out of } A} f(e)$$

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

---

**Strong duality**

If Ford-Fulkerson terminates, then $f$ is max flow.

---

**Certificate of optimality**

**Max-flow min-cut theorem**

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

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

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

i. There exists a cut $(A, B)$ such that $\text{cap}(A, B) = \text{val}(f)$.
ii. $f$ is a max flow.
iii. There is no augmenting path with respect to $f$.

$[i \Rightarrow ii]$

- Suppose that $(A, B)$ is a cut such that $\text{cap}(A, B) = \text{val}(f)$.
- Then, for any flow $f'$: $\text{val}(f') \leq \text{cap}(A, B) = \text{val}(f)$.
- Thus, $f$ is a max flow. $lacklozenge$
Max-flow min-cut theorem

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

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

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

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

ii. $f$ is a max flow.

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

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

$\cdot$ Suppose that there is an augmenting path with respect to $f$.

$\cdot$ Can improve flow $f$ by sending flow along this path.

$\cdot$ Thus, $f$ is not a max flow. •

Analysis of Ford–Fulkerson algorithm (when capacities are integral)

Assumption. Capacities are integers between 1 and $C$.

Integrality invariant. Throughout the algorithm, the flows $f(e)$ and the residual capacities $c_f(e)$ are integers.

Theorem. The algorithm terminates in at most $\text{val}(f^*) \leq nC$ iterations, where $f^*$ is a max flow.

Pf. Each augmentation increases the value of the flow by at least 1. •

Corollary. The running time of Ford–Fulkerson is $O(mnC)$.

Corollary. If $C = 1$, the running time of Ford–Fulkerson is $O(mn)$.

Integrality theorem. Then exists a max flow $f^*$ for which every flow $f^*(e)$ is an integer.

Pf. Since algorithm terminates, theorem follows from invariant. •
Bad case for Ford–Fulkerson

Q. Is generic Ford–Fulkerson algorithm poly-time in input size?

A. No. If max capacity is $C$, then algorithm can take $\geq C$ iterations.

- $s \rightarrow v \rightarrow w \rightarrow f$
- $s \rightarrow w \rightarrow y \rightarrow f$
- $s \rightarrow y \rightarrow w \rightarrow f$
- $s \rightarrow y \rightarrow v \rightarrow f$
- $s \rightarrow t \rightarrow y \rightarrow f$
- $s \rightarrow w \rightarrow t \rightarrow f$
- $s \rightarrow t \rightarrow w \rightarrow f$
- $s \rightarrow v \rightarrow t \rightarrow f$
- $s \rightarrow t \rightarrow v \rightarrow f$

Each augmenting path sends only 1 unit of flow (# augmenting paths = 2C)

Choosing good augmenting paths

**Use care when selecting augmenting paths.**
- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.

**Pathology.** If capacities are irrational, algorithm not guaranteed to terminate (or converge to correct answer).

**Goal.** Choose augmenting paths so that:
- Can find augmenting paths efficiently.
- Few iterations.

Capacity-scaling algorithm

**Intuition.** Choose augmenting path with highest bottleneck capacity: it increases flow by max possible amount in given iteration.
- Don’t worry about finding exact highest bottleneck path.
- Maintain scaling parameter $\Delta$.
- Let $G_f(\Delta)$ be the part of the residual network consisting of only those arcs with capacity $\geq \Delta$.
Capacity-scaling algorithm

**CAPACITY-SCALING** (G)

**FOREACH** edge \( e \in E : f[e] \leftarrow 0. \)
\( \Delta \leftarrow \) largest power of 2 \( \leq C. \)

**WHILE** (\( \Delta \geq 1 \))
\( G_\ell(\Delta) \leftarrow \Delta\)-residual network of \( G \) with respect to flow \( f. \)
**WHILE** (there exists an \( s \to t \) path \( P \) in \( G_\ell(\Delta) \))
\( f \leftarrow \text{AUGMENT} (f, c, P). \)
Update \( G_\ell(\Delta). \)
\( \Delta \leftarrow \Delta / 2. \)

**RETURN** \( f. \)

---

**Capacity-scaling algorithm: analysis of running time**

**Lemma 1.** The outer while loop repeats \( 1 + \lceil \log_2 C \rceil \) times.

**Pf.** Initially \( C/2 < \Delta \leq C; \) \( \Delta \) decreases by a factor of 2 in each iteration. •

**Lemma 2.** Let \( f \) be the flow at the end of a \( \Delta \)-scaling phase.

Then, the max-flow value \( \leq \text{val}(f) + m \Delta. \)  \( \longrightarrow \) proof on next slide

**Lemma 3.** There are at most \( 2m \) augmentations per scaling phase.

**Pf.**
- Let \( f \) be the flow at the end of the previous scaling phase.
- **Lemma 2 \( \Rightarrow \)** max-flow value \( \leq \text{val}(f) + 2m \Delta. \)
- Each augmentation in a \( \Delta \)-phase increases \( \text{val}(f) \) by at least \( \Delta. \) •

**Theorem.** The scaling max-flow algorithm finds a max flow in \( O(m \log C) \)

 augmentations. It can be implemented to run in \( O(m^2 \log C) \) time.

**Pf.** Follows from Lemma 1 and Lemma 3. •

---

**Capacity-scaling algorithm: proof of correctness**

**Assumption.** All edge capacities are integers between 1 and \( C. \)

**Integrality invariant.** All flows and residual capacities are integral.

**Theorem.** If capacity-scaling algorithm terminates, then \( f \) is a max flow.

**Pf.**
- By integrality invariant, when \( \Delta = 1 \) \( \Rightarrow G_\ell(\Delta) = G_f. \)
- Upon termination of \( \Delta = 1 \) phase, there are no augmenting paths. •

---

**Capacity-scaling algorithm: analysis of running time**

**Lemma 2.** Let \( f \) be the flow at the end of a \( \Delta \)-scaling phase.

Then, the max-flow value \( \leq \text{val}(f) + m \Delta. \)

**Pf.**
- We show there exists a cut \( (A, B) \) such that \( \text{cap}(A, B) \leq \text{val}(f) + m \Delta. \)
- Choose \( \Delta \) to be the set of nodes reachable from \( s \) in \( G_f(\Delta). \)
- By definition of cut \( A: s \in A. \)
- By definition of flow \( f: t \notin A. \)

\[
\text{val}(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)
\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta
\geq \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta
\geq \text{cap}(A, B) - m\Delta. \]
Shortest augmenting path: overview of analysis

**Lemma 1.** Throughout the algorithm, the length of a shortest augmenting path never decreases.

**Lemma 2.** After at most \( m \) shortest-path augmentations, the length of a shortest augmenting path strictly increases.

**Theorem.** The shortest-augmenting-path algorithm runs in \( O(m^2n) \) time.

**Pf.**
- \( O(m + n) \) time to find shortest augmenting path via BFS.
- \( O(m) \) augmentations for paths of length \( k \).
- If there is an augmenting path, there is a simple one.
  \[ 1 \leq k < n \]
  \[ \Rightarrow \, O(mn) \text{ augmentations.} \]

---

Shortest augmenting path: analysis

**Def.** Given a digraph \( G = (V, E) \) with source \( s \), its **level graph** is defined by:
- \( \ell(v) = \) number of edges in shortest path from \( s \) to \( v \).
- \( L_v = (V, E_v) \) is the subgraph of \( G \) that contains only those edges \( (v, w) \in E \) with \( \ell(w) = \ell(v) + 1 \).

---

**Shortest-augmenting-path algorithm**

\[
\begin{align*}
\text{FOREACH } e \in E : f(e) & \leftarrow 0. \\
G_f & \leftarrow \text{residual network of } G \text{ with respect to flow } f. \\
\text{WHILE (there exists an } s \rightarrow t \text{ path in } G_f) : \\
& P \leftarrow \text{BREADTH-FIRST-SEARCH} \ (G_f). \\
f & \leftarrow \text{AUGMENT} \ (f, c, P). \\
& \text{Update } G_f. \\
\text{RETURN } f.
\end{align*}
\]
Def. Given a digraph $G = (V, E)$ with source $s$, its level graph is defined by:
- $\ell(v) =$ number of edges in shortest path from $s$ to $v$.
- $L_C = (V, E_C)$ is the subgraph of $G$ that contains only those edges $(v, w) \in E$ with $\ell(w) = \ell(v) + 1$.

Property. Can compute level graph in $O(m + n)$ time.

PF. Run BFS; delete back and side edges.

Key property. $P$ is a shortest $s \to v$ path in $G$ iff $P$ is an $s \to v$ path $L_C$.

Shortest augmenting path: analysis

Lemma 2. After at most $m$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.
- The bottleneck edge(s) is deleted from $L$ after each augmentation.
- No new edge added to $L$ until length of shortest path strictly increases. •

Theorem. The shortest-augmenting-path algorithm runs in $O(m^2 n)$ time.

PF.
- $O(m + n)$ time to find shortest augmenting path via BFS.
- $O(m)$ augmentations for paths of length $k$.
- If there is an augmenting path, there is a simple one.
  $\Rightarrow 1 \leq k < n$
  $\Rightarrow O(m n)$ augmentations. •
Shortest augmenting path: improving the running time

Note. \( \Theta(m n) \) augmentations necessary on some flow networks.
- Try to decrease time per augmentation instead.
- Simple idea \( \Rightarrow O(m n^2) \) \cite{Dinitz 1970}
- Dynamic trees \( \Rightarrow O(m n \log n) \) \cite{Sleator–Tarjan 1983}

### Blocking-flow algorithm

**Two types of augmentations.**
- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

**Phase of normal augmentations.**
- Explicitly maintain level graph \( L_G \).
- Start at \( s \), advance along an edge in \( L_G \) until reach \( r \) or get stuck.
- If reach \( r \), augment and and update \( L_G \).
- If get stuck, delete node from \( L_G \) and go to previous node.

### Blocking-flow algorithm

**Two types of augmentations.**
- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

**Phase of normal augmentations.**
- Explicitly maintain level graph \( L_G \).
- Start at \( s \), advance along an edge in \( L_G \) until reach \( r \) or get stuck.
- If reach \( r \), augment and and update \( L_G \).
- If get stuck, delete node from \( L_G \) and go to previous node.
Blocking-flow algorithm

Two types of augmentations.
- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.
- Explicitly maintain level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment and update $L_G$.
- If get stuck, delete node from $L_G$ and go to previous node.

Blocking-flow algorithm

Two types of augmentations.
- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.
- Explicitly maintain level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment and update $L_G$.
- If get stuck, delete node from $L_G$ and go to previous node.
Blocking-flow algorithm

Two types of augmentations.
- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.
- Explicitly maintain level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment and update $L_G$.
- If get stuck, delete node from $L_G$ and go to previous node.

Blocking-flow algorithm

Two types of augmentations.
- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.
- Explicitly maintain level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment and update $L_G$.
- If get stuck, delete node from $L_G$ and go to previous node.
Blocking-flow algorithm

Two types of augmentations.
- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.
- Explicitly maintain level graph \(L_G\).
- Start at \(s\), advance along an edge in \(L_G\) until reach \(t\) or get stuck.
- If reach \(t\), augment and and update \(L_G\).
- If get stuck, delete node from \(L_G\) and go to previous node.

end of phase

Choosing good augmenting paths: summary

<table>
<thead>
<tr>
<th>year</th>
<th>method</th>
<th># augmentations</th>
<th>running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1955</td>
<td>augmenting path</td>
<td>(n \cdot C)</td>
<td>(O(mn))</td>
</tr>
<tr>
<td>1970</td>
<td>fastest augmenting path</td>
<td>(m \log(mn))</td>
<td>(O(m^2 \log n \log (mn)))</td>
</tr>
<tr>
<td>1972</td>
<td>capacity scaling</td>
<td>(m \log C)</td>
<td>(O(m^2 \log C))</td>
</tr>
<tr>
<td>1985</td>
<td>improved capacity scaling</td>
<td>(m \log C)</td>
<td>(O(mn \log C))</td>
</tr>
<tr>
<td>1970</td>
<td>shortest augmenting path</td>
<td>(mn)</td>
<td>(O(m^2 n)</td>
</tr>
<tr>
<td>1970</td>
<td>blocking flow</td>
<td>(mn)</td>
<td>(O(mn^2))</td>
</tr>
<tr>
<td>1983</td>
<td>dynamic trees</td>
<td>(mn)</td>
<td>(O(mn \log n))</td>
</tr>
</tbody>
</table>

*augmenting path algorithms with \(m\) edges, \(n\) nodes and integer capacities between 1 and \(C\)*
Maximum-flow algorithms: theory

<table>
<thead>
<tr>
<th>year</th>
<th>method</th>
<th>worst case</th>
<th>discovered by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1951</td>
<td>simplex</td>
<td>$O(m^3 C)$</td>
<td>Dantzig</td>
</tr>
<tr>
<td>1955</td>
<td>augmenting path</td>
<td>$O(m^2 C)$</td>
<td>Ford-Fulkerson</td>
</tr>
<tr>
<td>1970</td>
<td>shortest augmenting path</td>
<td>$O(m^3)$</td>
<td>Dinitz, Edmonds-Karp</td>
</tr>
<tr>
<td>1970</td>
<td>fattest augmenting path</td>
<td>$O(m^2 \log m \log (mC))$</td>
<td>Dinitz, Edmonds-Karp</td>
</tr>
<tr>
<td>1977</td>
<td>blocking flow</td>
<td>$O(m^{1.5})$</td>
<td>Cherkassky</td>
</tr>
<tr>
<td>1978</td>
<td>blocking flow</td>
<td>$O(m^{2/3})$</td>
<td>Galil</td>
</tr>
<tr>
<td>1983</td>
<td>dynamic trees</td>
<td>$O(n \log m)$</td>
<td>Sleator-Tarjan</td>
</tr>
<tr>
<td>1985</td>
<td>improved capacity scaling</td>
<td>$O(m \log C)$</td>
<td>Gabow</td>
</tr>
<tr>
<td>1997</td>
<td>length function</td>
<td>$O(m^{3/2} \log m \log C)$</td>
<td>Goldberg–Rao</td>
</tr>
<tr>
<td>2012</td>
<td>compact network</td>
<td>$O(m^2 / \log m)$</td>
<td>Orlin</td>
</tr>
</tbody>
</table>

max-flow algorithms for sparse digraphs with $m$ edges, integer capacities between 1 and $C$.

Maximum-flow algorithms: practice

Warning. Worst-case running time is generally not useful for predicting or comparing max-flow algorithm performance in practice.


On Implementing Push-Relabel Method for the Maximum Flow Problem

Bryan V. Cherkassky* and Andrew V. Goldberg


Abstract. We study efficient implementations of the push-relabel method for maximum flow problems. The resulting codes are faster than the fastest known implementations of other methods. The codes also exhibit a variety of features that we believe are also important for the writing of high-performance algorithms.

A New Approach to the Maximum-Flow Problem

Andrew V. Goldberg

Massachusetts Institute of Technology, Cambridge, Massachusetts

And

Robert E. Tarjan

Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the push-relabel concept of Karp and Tardos is introduced. A push-relabel algorithm is fast, except that the total amount flowing into a vertex is allowed to exceed the total amount flowing out. The central motivation is that the original network and pushed local flow exist near the edge along which we estimate to be the augmenting path. The algorithm is in essence simple and intuitive, yet the algorithm runs as fast as any other known method on dense graphs, achieving an $O(m)$ bound on a $m$-vertex graph. By incorporating the dynamic tree data structure of Sleator and Tarjan, we obtain an $O(n \log m)$ bound on an $m$-vertex, $n$-edge graph. This is as fast as any known method for any graph density and faster on graphs of moderate density. The algorithm also yields efficient distributed and parallel implementations. A parallel implementation running in $O(\log m)$ time using $m$ processors and $O(m)$ space is obtained. This time bound matches that of the Ford-Fulkerson algorithms, which also uses $m$ processors but requires $O(m)$ space.

An Experimental Comparison of Max-Flow/Min-Cut Algorithms for Density Minimization in Vision

Yiwei Hecker and Vladlen Koltun

Abstract. We compare several max-flow/min-cut algorithms on modern problems. Our problem instances arise in the contexts of image restoration, stereo, and segmentation. We present empirical results and analysis of several max-flow/min-cut algorithms.

MaxFlow Revisited: An Empirical Comparison of Maxflow Algorithms for Dense Vision Problems

Yiwei Hecker

Abstract. We compare several max-flow/min-cut algorithms on modern problems. Our problem instances arise in the contexts of image restoration, stereo, and segmentation. We present empirical results and analysis of several max-flow/min-cut algorithms.

Computer vision. Different algorithms work better for some dense problems that arise in applications to computer vision.
7. **NETWORK FLOW I**

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

**Simple unit-capacity networks**

**Def.** A flow network is a simple unit-capacity network if:
- Every edge has capacity 1.
- Every node (other than s or t) has either (i) exactly one entering edge or (ii) exactly one leaving edge (or both).

**Property.** Let \( G \) be a simple unit-capacity network and let \( f \) be a 0–1 flow, then \( G_f \) is a simple unit-capacity network.

**Ex.** Bipartite matching.

**Bipartite matching**

**Q.** Which max-flow algorithm to use for bipartite matching?
- Generic augmenting path: \( O(m \text{val}(f^*)) = O(mn) \).
- Capacity scaling: \( O(m^2 \log C) = O(m^2) \).
- Blocking flow: \( O(mn^2) \).

**Q.** Suggests more sophisticated algorithms are not so fast when \( C = 1 \).
**A.** No, just need more clever analysis!

**Next.** We prove that shortest-augmenting-path algorithm can be implemented to run in \( O(mn^{1/2}) \) time.

---

**NETWORK FLOW AND TESTING GRAPH CONNECTIVITY*\(^{1}\)**

**SHIMON EVEN** and R. ENDRE TARJAN

**Abstract.** An algorithm of Dinic for finding the maximum flow in a network is described. It is then shown that if the vertex capacities are all equal to one, the algorithm requires at most \( O(|V|^{1/2} |E|) \) time, and if the edge capacities are all equal to one, the algorithm requires at most \( O(|V|^{1/2} |E|) \) time. Also, these bounds are tight for Dinic’s algorithm.

These results are used to test the vertex connectivity of a graph in \( O(|V|^{1/2} |E|) \) time and the edge connectivity in \( O(|V|^{1/2} |E|) \) time.

---

**Simple unit-capacity networks**

**Shortest-augmenting-path algorithm.**
- Normal augmentation: length of shortest path does not change.
- Special augmentation: length of shortest path strictly increases.

**Theorem.** [Even–Tarjan 1975] In simple unit-capacity networks, the shortest-augmenting-path algorithm computes a maximum flow in \( O(mn^{1/2}) \) time.

**Pf.**
- Lemma 1. Each phase of normal augmentations takes \( O(m) \) time.
- Lemma 2. After at most \( n^{1/2} \) phases, \( \text{val}(f) \geq \text{val}(f^*) - n^{1/2} \).
- Lemma 3. After at most \( n^{1/2} \) additional augmentations, flow is optimal.

**Lemma 3.** After at most \( n^{1/2} \) additional augmentations, flow is optimal.

**Pf.** Each augmentation increases flow value by at least 1.
Simple unit-capacity networks

Phase of normal augmentations.
- Explicitly maintain level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment and and update $L_G$. delete all edges in augmenting path from $L_G$.
- If get stuck, delete node from $L_G$ and go to previous node.
Simple unit-capacity networks

Phase of normal augmentations.
- Explicitly maintain level graph \(L_G\).
- Start at \(s\), advance along an edge in \(L_G\) until reach \(t\) or get stuck.
- If reach \(t\), augment and and update \(L_G\) \(\rightarrow\) delete all edges in augmenting path from \(L_G\).
- If get stuck, delete node from \(L_G\) and go to previous node.

detail image

Simple unit-capacity networks: analysis

Phase of normal augmentations.
- Explicitly maintain level graph \(L_G\).
- Start at \(s\), advance along an edge in \(L_G\) until reach \(t\) or get stuck.
- If reach \(t\), augment and and update \(L_G\) \(\rightarrow\) delete all edges in augmenting path from \(L_G\).
- If get stuck, delete node from \(L_G\) and go to previous node.

Lemma 1. A phase of normal augmentations takes \(O(m)\) time.

Pf. 
- \(O(m)\) to create level graph \(L_G\).
- \(O(1)\) per edge since each edge traversed and deleted at most once.
- \(O(1)\) per node since each node deleted at most once.

end of phase
detail image
**Simple unit-capacity networks: analysis**

**Lemma 2.** After at most $n^{1/2}$ phases, $\text{val}(f) \geq \text{val}(f^*) - n^{1/2}$.

- After $n^{1/2}$ phases, length of shortest augmenting path is $> n^{1/2}$.
- Level graph has more than $n^{1/2}$ levels.
- Let $1 \leq h \leq n^{1/2}$ be layer with min number of nodes: $|V_h| \leq n^{1/2}$.

\[
\text{level graph } L_f \text{ for flow } f
\]

\[
V_0 \quad V_1 \quad V_h \quad V_{n^{1/2}}
\]

**residual network** $G_f$

\[
\text{residual edges}
\]

\[
V_0 \quad V_1 \quad V_h \quad V_{n^{1/2}}
\]