Network Flows 1:
- definitions
- residual graphs & augmenting paths
- Ford-Fulkerson algorithm

Flow Network definition
- $G = (V, E)$ is a directed weighted graph.
  - Source vertex $s$ in $V$ has no incoming edges.
  - Terminal vertex $t$ in $V$ has no outgoing edges.
- For each edge $e$ in $E$, the edge weight is an integer valued capacity $c(e) \geq 0$.

Network Flow definition
- $f : E \rightarrow \{0, 1, 2, \ldots \}$
- Capacity Constraint: for any $e \in E$, $0 \leq f(e) \leq c(e)$
- Conservation Constraint: for any $v \in V \backslash \{s, t\}$,
  $$\sum_{u \in V} f(u, v) = \sum_{u \in V} f(v, u)$$
\[ f^{\text{out}}(s) \equiv \sum_{v \in V} f(s, v) \]
\[ f^{\text{in}}(t) \equiv \sum_{v \in V} f(v, t) \]

Exercise:
Show \( f^{\text{out}}(s) = f^{\text{in}}(t) \), which is called the value of the flow.

Given a flow network \((G, s, t, c)\), there may be many “admissible,” i.e., allowable network flows. The maximum flow is the flow that has the largest value.

How can we find the maximum flow?
(Note there may be two flows have the same value. That’s ok.)

Algorithm 1: how to find maximum flow from \(s\) to \(t\)?

Initialize \( f = 0 \)
while true {
  if there is a path \(P\) from \(s\) to \(t\), such that all edges on that path have a flow that is strictly less than the capacity then increase the flow on that path by as much as possible
  else break
}

Example (flow)

\[
\begin{array}{cccc}
  & 2 & \rightarrow & 1 \\
S \rightarrow & 3 & \rightarrow & 4 \\
& t \\
\end{array}
\]

\[ f^{\text{out}}(s) = 5 \]
\[ f^{\text{in}}(t) = 5 \]

Note conservation conditions for \(V \setminus \{s, t\}\)

Applications

The formulation of the maximum flow problem suggests obvious applications. e.g., transportation networks.

But there are also many applications that have nothing to do with physical flows. e.g., matching problems.

Kleinberg Tardos textbook has several sections on applications which are typically covered in Comp 360. e.g.,

speaks first 3½ weeks on network flows.


Algorithm 1: how to find maximum flow from \(s\) to \(t\)?

Initialize \( f = 0 \)
while true {
  if there is a path \(P\) from \(s\) to \(t\), such that for all edges \(e\) in \(P\), \(f(e) < c(e)\) then 
    \( \beta = \min \{c(e) - f(e) : e \in P\} \)
    For all \(e \in P\)
      \( f(e) = f(e) + \beta \)
  else break
}

\( \beta \) stands for “bottleneck”
Example where Algorithm 1 works:

Flow network

\[ |f| = 2 \quad |f| = 4 \quad |f| = 5 \]

How to choose paths so that we don’t get stuck
- are guaranteed to find the maximum flow
- are efficient (will be covered in COMP 340)

Example where Algorithm 1 fails:

Algorithm 2 Motivation: if we could subtract flow, then we could redirect it.

Flow network

\[ |f| = 3 \quad \text{and algorithm terminates} \]

Using negative numbers on directed edges is possible, but I will present an alternative representation which is based on a “residual graph” and use that in the second algorithm (later).

The residual graph is a weighted graph whose edge weights represent how we can change the flow \( f \).

Residual Graph

Given a flow network \( G = (V,E) \) with edge capacities \( C \), and given a flow \( f \), define the residual graph \( G_f \):
- \( G_f \) has the same vertices as \( G \)
- The edges \( E_f \) have capacities \( C_f \) (called ‘residual capacities’) that allow us to change the flow \( f \), either by:
  1) adding flow to an edge \( e \) in \( E_f \)
  2) subtracting flow from an edge \( e \) in \( E_f \)

Algorithm 1

Negative value on edge doesn’t satisfy definition of flow.

\[ |f| = 2 \]

Algorithm 1 terminates here

\[ |f| = 2 \]
For each edge \( e = (u, v) \) in \( E \):

- If \( f(e) < c(e) \)
  - then put a 'forward edge' \((u, v)\) in \( E_f \) with residual capacity \( c_f(e) = c(e) - f(e) \)
- If \( f(e) > 0 \)
  - then put a 'backwards edge' \((v, u)\) in \( E_f \) with residual capacity \( c_f(e) = f(e) \)

**Example 1**

<table>
<thead>
<tr>
<th>Flow Network</th>
<th>Flow</th>
<th>Residual Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>G, c</strong></td>
<td><strong>f</strong></td>
<td><strong>G_f, c_f</strong></td>
</tr>
<tr>
<td>( s )</td>
<td>( t )</td>
<td>( s )</td>
</tr>
<tr>
<td>1 [\rightarrow] 2</td>
<td>1</td>
<td>1 [\rightarrow] 1</td>
</tr>
<tr>
<td>2 [\rightarrow] 3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3 [\rightarrow] 1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example 2**

<table>
<thead>
<tr>
<th>Flow Network</th>
<th>Flow</th>
<th>Residual Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>G, c</strong></td>
<td><strong>f</strong></td>
<td><strong>G_f, c_f</strong></td>
</tr>
<tr>
<td>( s )</td>
<td>( t )</td>
<td>( s )</td>
</tr>
<tr>
<td>1 [\rightarrow] 2</td>
<td>1</td>
<td>1 [\rightarrow] 1</td>
</tr>
<tr>
<td>2 [\rightarrow] 3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3 [\rightarrow] 1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example 3**

<table>
<thead>
<tr>
<th>Flow Network</th>
<th>Flow</th>
<th>Residual Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>G, c</strong></td>
<td><strong>f</strong></td>
<td><strong>G_f, c_f</strong></td>
</tr>
<tr>
<td>( s )</td>
<td>( t )</td>
<td>( s )</td>
</tr>
<tr>
<td>1 [\rightarrow] 2</td>
<td>1</td>
<td>1 [\rightarrow] 1</td>
</tr>
<tr>
<td>2 [\rightarrow] 3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3 [\rightarrow] 1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

An augmenting path is an "s-t path" (a path from \( s \) to \( t \)) in the residual graph \( G_f \) that allows us to increase the flow.

**Q:** In the example here, by how much can we increase the flow by using this path?

**A:** 2

Flow Network | Flow in \( G \) | Flow in \( G_f \) | Flow in \( G \)
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>G, c</strong></td>
<td><strong>f</strong></td>
<td><strong>G_f, c_f</strong></td>
<td></td>
</tr>
<tr>
<td>( s )</td>
<td>( t )</td>
<td>( s )</td>
<td></td>
</tr>
<tr>
<td>3 [\rightarrow] 2</td>
<td>0 [\rightarrow] 2</td>
<td>0 [\rightarrow] 2</td>
<td></td>
</tr>
<tr>
<td>0 [\rightarrow] 3</td>
<td>[\rightarrow] 3</td>
<td>[\rightarrow] 3</td>
<td></td>
</tr>
<tr>
<td>3 [\rightarrow] 0</td>
<td>2 [\rightarrow] 2</td>
<td>2 [\rightarrow] 2</td>
<td></td>
</tr>
<tr>
<td>value of flow</td>
<td>(</td>
<td>f</td>
<td>= 3 )</td>
</tr>
</tbody>
</table>

Bottleneck value of flow
Algorithm 2: computing maximum flow (Ford - Fulkerson 1954)

\[ f = 0 \]
\[ G_f = G \]

while there is an s-t path \( P \) in \( G_f \) \{ 
  f. augment (\( P \))
  update \( G_f \) based on new \( f \)
\}

Claim: The Ford-Fulkerson algorithm terminates.

Proof:
The capacities and flows are integers \( \geq 0 \).
The sum of capacities leaving \( S \) is finite.
Bottleneck values \( \beta \) are positive integers.
The flow increases by the bottleneck \( \beta \) in each pass through main loop.
\( \Rightarrow \) Flow is an increasing sequence of integers, bounded above.

How long does F-F take?
Let \( C = \sum c(e) \) outgoing from \( s \).
Finding a path from \( s \) to \( t \) in \( G_f \) takes \( O(1 \cdot |E|) \) e.g. DFS or BFS.
Since the flow increases by at least 1 in each pass, the algorithm is \( O(C \cdot |E|) \)

Worst case of F-F

\[ G_f = G \]
find any path \( f \)
Worst case of F-F (continued)

\[ G_f \] choose augmenting path \[ f \]

Repeating this will take \( 1000 + 2 \) iterations to find a flow of 2000. A better choice would have found that flow in 2 iterations.

| ASIDE: | many implementations of F-F (such as Sedgewick's) don't construct a residual graph \( G_f \). Instead, they find a "path" in the original graph \( G \) (where some edges may be "backward") and they subtract flow on these edges.

Next lecture
network flows 2

- max flow = min cut

COMP 360
- efficient network flow (how to choose a good augmenting path)
- applications

Announcements
- midterm 1 grading - see updated solutions PDF
- A1 grading: late penalties and fairness
- A3 will be posted end of next week (or later) and due in early/mid March (now is the time to catch up)