Network Flow II

Inge Li Gørtz

KT 7.3, 7.5, 7.6

Network Flow

- Network flow:
  - graph $G=(V,E)$.
  - Special vertices $s$ (source) and $t$ (sink).
  - Every edge $e$ has a capacity $c(e) \geq 0$.
- Flow:
  - capacity constraint: every edge $e$ has a flow $0 \leq f(e) \leq c(e)$.
  - flow conservation: for all $u \neq s, t$: flow into $u$ equals flow out of $u$.

$$\sum_{v : (v,u) \in E} f(v,u) - \sum_{v : (u,v) \in E} f(u,v)$$

- Value of flow $f$ is the sum of flows out of $s$ minus sum of flows into $s$:

$$v(f) = \sum_{e \in E} f(e) - \sum_{e \in E} f(e) = f^o(s) - f^i(s)$$

- Maximum flow problem: find $s$-$t$ flow of maximum value

Today

- Applications
  - Bipartite matching: Hospital have to schedule doctors for the holidays.
    - Doctors have constraints on how many and on which holidays they can work.
    - Hospital needs a certain amount on doctors at work.
  - Disjoint paths:
    - Finding good augmenting paths. Edmonds-Karp and scaling algorithm.

Ford-Fulkerson

- Find (any) augmenting path and use it.
- Augmenting path (definition different than in CLRS): $s$-$t$ path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

$$\min(c_1 - f_1, c_2 - f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$$

- To find augmenting path use DFS or BFS:
Ford-Fulkerson

- Integral capacities:
  - Each augmenting path increases flow with at least 1.
  - At most \( v(f) \) iterations
  - Find augmenting path via DFS/BFS: \( O(m) \)
  - Total running time: \( O(v(f) m) \)

- Lemma. If all the capacities are integers, then there is a maximum flow where the flow on every edge is an integer.

- Bad example for Ford-Fulkerson:

```
\begin{array}{c|c|c|c|c|c|c|c}
  & s & & & & t \\
  \hline
  f_1 & +5 & -5 & -5 & +5 & -5 & -5 & t \\
  f_2 & f_1 < c_1 & f_2 > 0 & f_3 < c_3 & f_4 < c_4 & f_5 > 0 & f_6 > 0 & t \\
  \hline
  f_3 & f_5 < c_5 \\
  f_4 & f_6 > 0 \\
  \hline
  f_6 & f_6 > 0 \\
  \end{array}
```

- Can add extra flow: \( \min(c_1 - f_1, f_2, c_3 - f_5, c_4 - f_6) = \delta \)

- To find augmenting path use BFS:

```
\begin{array}{c|c|c|c|c|c|c|c}
  & s & & & & t \\
  \hline
  f_1 & +5 & -5 & -5 & +5 & -5 & -5 & t \\
  f_2 & f_1 < c_1 & f_2 > 0 & f_3 < c_3 & f_4 < c_4 & f_5 > 0 & f_6 > 0 & t \\
  \hline
  f_3 & f_5 < c_5 \\
  f_4 & f_6 > 0 \\
  \hline
  f_6 & f_6 > 0 \\
  \end{array}
```

Edmonds-Karp

- Find shortest augmenting path and use it.

- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

```
\begin{array}{c|c|c|c|c|c|c|c}
  & s & & & & t \\
  \hline
  f_1 & +5 & -5 & -5 & +5 & -5 & -5 & t \\
  f_2 & f_1 < c_1 & f_2 > 0 & f_3 < c_3 & f_4 < c_4 & f_5 > 0 & f_6 > 0 & t \\
  \hline
  f_3 & f_5 < c_5 \\
  f_4 & f_6 > 0 \\
  \hline
  f_6 & f_6 > 0 \\
  \end{array}
```

- Can add extra flow: \( \min(c_1 - f_1, f_2, c_3 - f_5, c_4 - f_6) = \delta \)

- To find augmenting path use BFS:

```
\begin{array}{c|c|c|c|c|c|c|c}
  & s & & & & t \\
  \hline
  f_1 & +5 & -5 & -5 & +5 & -5 & -5 & t \\
  f_2 & f_1 < c_1 & f_2 > 0 & f_3 < c_3 & f_4 < c_4 & f_5 > 0 & f_6 > 0 & t \\
  \hline
  f_3 & f_5 < c_5 \\
  f_4 & f_6 > 0 \\
  \hline
  f_6 & f_6 > 0 \\
  \end{array}
```
Edmonds-Karp

- Find shortest augmenting path and use it.
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow
- Can add extra flow: \( \min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta \)
- To find augmenting path use BFS:

```
  S  f_1 < c_1  f_2 > 0  f_3 < c_2  f_4 > 0  f_5 > 0  f_6 > 0  t

  S  4  4  4  4  4/4  3/3  2/6  2/5  2/8
```

Find a minimum cut

- When there are no more augmenting s-t paths:
  - Find all augmenting paths from s.
  - The nodes \( S \) that can be reached by these augmenting paths form the left side of a minimum cut.
    - edges out of \( S \) have \( c_e = f_e \).
    - edges into \( S \) have \( f_e = 0 \).
    - Capacity of the cut equals the flow.

```
```

Scaling algorithm

- Scaling parameter \( \Delta \)
- Only consider edges with capacity at least \( \Delta \) in residual graph \( G_f(\Delta) \).
- Example: \( \Delta = 4 \)
Scaling algorithm

- Scaling parameter $\Delta$
- Only consider edges with capacity at least $\Delta$ in residual graph $G(\Delta)$.
- Start with $\Delta = \text{"highest power of 2} \leq \text{largest capacity out of } s$.
Scaling algorithm

• Scaling parameter $\Delta$
• Only consider edges with capacity at least $\Delta$ in residual graph $G_f(\Delta)$.
• Start with $\Delta = \text{"highest power of 2 ≤ largest capacity out of s"}$
• When no more augmenting paths in $G_f(\Delta)$: $\Delta = \Delta/2$ (new phase).

$\Delta = 8$

$\Delta = 4$

$\Delta = 4$

$\Delta = 4$
Scaling algorithm

- Scaling parameter $\Delta$
- Only consider edges with capacity at least $\Delta$ in residual graph $G(\Delta)$.
- Start with $\Delta =$ “highest power of 2 ≤ largest capacity out of s”
- When no more augmenting paths in $G(\Delta)$: $\Delta = \Delta/2$ (new phase).

$\Delta = 4$

$\Delta = 2$

$\Delta = 2$
Scaling algorithm

- Scaling parameter $\Delta$
- Only consider edges with capacity at least $\Delta$ in residual graph $G_f(\Delta)$.
- Start with $\Delta = \text{"highest power of 2 \leq largest capacity out of s"}$
- When no more augmenting paths in $G_f(\Delta): \Delta = \Delta/2$ (new phase).

\[\Delta = 2\]

\[\Delta = 1\]

\[\Delta = 1\]
Scaling algorithm

- Scaling parameter $\Delta$
- Only consider edges with capacity at least $\Delta$ in residual graph $G(\Delta)$.
- Start with $\Delta = \text{"highest power of 2 ≤ largest capacity out of s"}$
- When no more augmenting paths in $G(\Delta)$: $\Delta = \Delta/2$ (new phase).

\[ \Delta = 1 \]

Stop when no more augmenting paths in $G(1)$. 

---

Scaling algorithm

- Scaling parameter $\Delta$
- Only consider edges with capacity at least $\Delta$ in residual graph $G(\Delta)$.
- Start with $\Delta = \text{"highest power of 2 ≤ largest capacity out of s"}$
- When no more augmenting paths in $G(\Delta)$: $\Delta = \Delta/2$ (new phase).

\[ \Delta = 1 \]
Algorithms and Data Structures 2 Weekplan 7

Lecture
At the lecture we will talk more about network flow. We will consider two applications: bipartite matching and finding disjoint paths. We will also discuss how to find good augmenting paths. You should read KT 7.3, 7.5 and 7.6.

Exercises

The Edmonds-Karp algorithm and the scaling algorithm. Use both the Edmonds-Karp's algorithm and the scaling algorithm to compute a maximum flow and minimum cut on the two graphs below. For each augmenting path write the nodes on the path and the value you augment the path with.

Blood donors
At the halloween party at a well-known academic institution north of Copenhagen not all went smooth and some students had to be taken to medical emergency treatment at Rigshospitalet. In total 150 had to get a transfusion of one bag of blood. The hospital had 155 bags in stock. The distribution of blood groups in the supply and amongst the students is shown in the table below.

<table>
<thead>
<tr>
<th>Blood type</th>
<th>A</th>
<th>B</th>
<th>0</th>
<th>AB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bags in stock</td>
<td>44</td>
<td>31</td>
<td>42</td>
<td>38</td>
</tr>
<tr>
<td>Demand</td>
<td>37</td>
<td>33</td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

Type A patients can only receive blood of type A or type 0; type B patients can receive only type B or type 0; type 0 patients can receive only type 0; and type AB patients can receive any of the four types.

Model the problem as a flow network problem. Draw the corresponding network, and interpret the meaning of the nodes, and edges (edges capacities). Describe an algorithm to check whether every student can get a transfusion, otherwise how many can get one.

You do not have to solve the problem explicitly.
Scaling algorithm

- Running time: \( O(m^2 \log C) \), where \( C \) is the largest capacity out of \( s \).

- Lemma 1. Number of scaling phases: \( 1 + \lceil \log C \rceil \)

- Lemma 2. Let \( f \) be the flow when \( \Delta \)-scaling phase ends, and let \( f' \) be the maximum flow. Then \( v(f') \leq v(f) + m \Delta \).

- Lemma 3. The number of augmentations in a scaling phase is at most \( 2m \).
  - First phase: can use each edge out of \( s \) in at most one augmenting path.
  - \( f \) flow at the end of previous phase.
  - Used \( \Delta' = 2 \Delta \) in last round.
  - Lemma 2: \( v(f') \leq v(f) + m \Delta' = v(f) + 2m \Delta \).
  - “Leftover flow” to be found \( \leq 2m \Delta \).
  - Each augmentation in a \( \Delta \)-scaling phase augments flow with at least \( \Delta \).

Maximum flow algorithms

- Edmonds-Karp: \( O(m^2 n \log n) \)
- Scaling: \( O(m^2 \log C) \)
- Ford-Fulkerson \( O(m v(f)) \)
- Preflow-push \( O(n^3) \)
- Other algorithms: \( O(mn \log n) \) or \( O(m^{1/2} n^{1/3}) \) or \( O(m \log n \log U) \).

Maximum Bipartite Matching

- Bipartite graph: Can color vertices red and blue such that all edges have a red and a blue endpoint.
- Matching: Subset of edges \( M \subseteq E \) such that no edges in \( M \) share an endpoint.
- Maximum matching: matching of maximum cardinality.

- Applications:
  - planes to routes
  - jobs to workers/machines
Maximum Bipartite Matching

• Bipartite graph: Can color vertices red and blue such that all edges have a red and a blue endpoint.
• Matching: Subset of edges $M \subseteq E$ such that no edges in $M$ share an endpoint.
• Maximum matching: matching of maximum cardinality.

• Solve via flow:
  • Matching $M \Rightarrow$ flow of value $|M|
Maximum Bipartite Matching

- Bipartite graph: Can color vertices red and blue such that all edges have a red and a blue endpoint.
- Matching: Subset of edges $M \subseteq E$ such that no edges in $M$ share an endpoint.
- Maximum matching: matching of maximum cardinality.

- Solve via flow:
  - Matching $M \Rightarrow$ flow of value $|M|
  - Flow of value $v(f) \Rightarrow$ matching of size $v(f)$

Scheduling of doctors

- $X$ doctors, $Y$ holidays, each doctor should work at most 1 holiday, each doctor is available at some of the holidays.

- Same problem, but each doctor should work at most $c$ holidays?
Scheduling of doctors

- X doctors, Y holidays, each doctor should work at at most c holidays, each doctor is available at some of the holidays.
- Same problem, but each doctor should work at most one day in each vacation period?

Edge Disjoint paths

- Problem: Find maximum number of edge-disjoint paths from s to t.
- Two paths are edge-disjoint if they have no edge in common.
**Edge Disjoint Paths**
- Reduction to max flow: assign capacity 1 to each edge.
- Thm. Max number of edge-disjoint s-t paths is equal to the value of a maximum flow.
- Suppose there are k edge-disjoint paths: then there is a flow of k (let all edges on the paths have flow 1).
- Other way (graph theory course).
- Ford-Fulkerson: $v(f) \leq n$ (no multiple edges and therefore at most $n$ edges out of s) \(\Rightarrow\) running time \(O(nm)\).

**Network Connectivity**
- **Network connectivity.** Find minimum number of edges whose removal disconnects t from s (destroys all s-t paths).
- Set all capacities to 1 and find minimum cut.
- Thm. (Menger) The maximum number of edge-disjoint s-t paths is equal to the minimum number of edges whose removal disconnects t from s.

**Node capacities**
- Capacities on nodes.