max flow

abhi shelat
Max flow

Min Cut
“Consider a rail network connecting two cities by way of a number of intermediate cities, where each link of the network has a number assigned to it representing its capacity. Assuming a steady state condition, find a maximal flow from one given city to the other.”
We will finally describe a more recent and more peaceful application of flow methods to railways as used by Nederlandse Spoorwegen for Timetable 2007.

NS runs an hourly train service on its route Amsterdam-Rotterdam-Roosendaal-Vlissingen and vice versa with the timetable shown above. The trains have more stops but for our purposes only those given in the table are of interest since at the stations given train sections can be coupled or separated. For each of the stages of any scheduled trains NS has estimated the number of passengers as given in the table on the next page. All data concerns weekdays and 2nd class seats.

The problem to be solved is: What is the minimum amount of train stock necessary to perform this train service in such a way that at each stage there are enough seats?

In order to answer this question one should know a number of further characteristics and constraints. In a first version of the problem considered the train stock consisted of one type of two-way train units, each consisting of three carriages. Each unit has \( x \) seats.

Each unit has at both ends an engineer’s cabin and units can be coupled together up to a certain maximum length, meaning in this case \( y \) train units. The train length can be changed by coupling or decoupling units at the terminal stations of the lines that is at Amsterdam and Vlissingen and en route at the intermediate stations Rotterdam and Roosendaal. Any train unit decoupled from a train arriving at place \( p \) at time \( t \) can be linked up to any other train departing from \( p \) at any time later than \( t \). The Amsterdam-Vlissingen schedule is such that in practice this gives enough time to make the necessary switchings.

A last condition is that for each place \( p \) \{Amsterdam, Rotterdam, Roosendaal, Vlissingen\}, the number of train units staying overnight at \( p \) should be constant during the week but may vary for different places. This requirement is made to facilitate surveying the stock and to equalize at any place the load of overnight cleaning and maintenance throughout the week. It is not required that the same train units after a night in Roosendaal for example should return to Roosendaal at the end of the day. Only the number of units is of importance.

Given these problem data and characteristics, one may ask for the minimum number of train units that should be available to perform the daily cycle of train rides required. It is assumed that if there is sufficient stock for Monday till Friday then this should also be enough for the weekend services since at the weekend a few early trains are cancelled and on the remaining trains there is a smaller expected number of passengers. Moreover, it is not taken into consideration that stock can be exchanged during the day with other lines of the network. In practice this will happen but initially this possibility is ignored.

A network model

If only one type of railway stock is used, last
FLOW NETWORKS

\[ G = (V, E) \]

SOURCE + SINK:

\[ S \quad \epsilon \]

CAPACITIES:

\[ c : E \rightarrow \mathbb{Q}^+ \quad \text{rational positive numbers} \]
FLOW

MAP FROM EDGES TO NUMBERS: \( f: E \rightarrow \mathbb{Q}^+ \)

CAPACITY CONSTRAINT: \( f(e) \leq C(e) \)

FLOW CONSTRAINT: For every node \( v \in V \): \( \sum s_i + t^3 \)

\( \text{IN}(v) = \text{OUT}(v) \Rightarrow \sum \nabla f(u,v) = \sum f(v,w) \)

\[ |f| = \text{OUT}(s) - \text{IN}(s) \]
MAX FLOW PROBLEM

given a graph G, compute

\[ \arg \max_f |f| \]
EXAMPLE of a flow

Diagram:

- Nodes: S, 1/2, 2/3, 1/3, T
- Edges with labels:
  - S → 1/2: 1/3
  - 1/2 → 2/3: 2/2
  - 2/3 → T: 2/2
  - 1/3 → S: 0/1
  - 1/2 → 2/3: 1/1
  - 2/3 → 1/2: 2/3
  - 1/1 → 1/2: 2/3

Flow capacity: 1/3
HUNDREDS OF APPLICATIONS

BIPARTITE MATCHING
EDGE-DISJOINT PATHS
NODE-DISJOINT PATHS
SCHEDULING
BASEBALL ELIMINATION
RESOURCE ALLOCATIONS

WILL DISCUSS MANY OF THESE APPLICATIONS IN L22.
ALGORITHMS FOR MAX FLOW
GREEDY FAILS

allows a flow of 30

can push 20 units along this path.
RESIDUAL GRAPHS

\[ G_f = (V, E_f) \]

given a flow \( f \), one can construct a residual graph \( G_f \) using

\[ c_f(u, v) = \]

"whenever you push \( x \) units on edge \((u, v)\), create a residual edge from \((v, u)\) with capacity \( x \)."
AUGMENTING PATHS

DEF: Any path from $s$ to $t$ in a residual graph $G_f$. 
EXAMPLE RESIDUAL GRAPH
$G = (V, E)$

- 1st augmenting path
- we can push 1 unit
- we push 1 unit of flow S → B → H → T
No more augmenting paths from $s$ to $t$!!

$\Rightarrow$ done.
\[ |f| = \| \{ S, A \}, \{ B, H, I, T, \} \| \]

By lemma, there cannot be a larger flow.

\( \{ S, A \}, \{ B, H, I, T, \} \) is a graph cut.

This cut has value 3.
OTHER CUTS ARE LARGER
For any $f, (S, T)$ it holds that $|f| \leq |S, T|$.

Example:
CUTS

DEF OF A CUT:

COST OF A CUT:

\[ ||S, T|| = \sum_{u \in S} \sum_{w \in T} c(u, w) \]
LEMMA: [MIN CUT] FOR ANY $f, (S, T)$, $|f| \leq \|S, T\|$
THM: MAX FLOW = MIN CUT

\[ \max_{f} |f| = \min_{S,T} ||S, T|| \]

If f is a max flow, then Gf has no augmenting paths.
THM: MAX FLOW = MIN CUT

\[
\max_f |f| = \min_{S,T} ||S, T||
\]
FORD-FULKERSON

\[
\begin{align*}
\text{INITIALIZE} & \quad f(u,v) \leftarrow 0 \ \forall u,v \\
\text{WHILE EXISTS AN AUGMENTING PATH } p \text{ IN } & \quad G_f \\
\text{AUGMENT } f \text{ WITH } & \quad c_f(p) = \min_{(u,v) \in p} c_f(u,v)
\end{align*}
\]
why does FF work? (high level)

We simultaneously construct a flow and cut $(S,T)$ such that $|F| = \|S \cap T\|$.
FORD-FULKERSON

**INITIALIZE**

\[ f(u, v) \leftarrow 0 \ \forall u, v \]

**WHILE EXISTS AN AUGMENTING PATH \( p \) IN \( G_f \)**

**AUGMENT \( f \) WITH**

\[ c_f(p) = \min_{(u,v) \in p} c_f(u, v) \]

**TIME TO FIND AN AUGMENTING PATH:**

**NUMBER OF ITERATIONS OF WHILE LOOP:**
ROOT OF THE PROBLEM

picky bad paths.
EDMONDS-KARP 2

CHOOSE PATH WITH FEWEST EDGES FIRST.

$$\delta_f(s, v) : \# \text{ of hops from } s \text{ to } v \text{ along the shortest path in residual graph } G_f.$$
LEMMA: $\delta_f(s, v)$ INCREASES MONOTONICALLY THRU EXEC

$\delta_{i+1}(v) \geq \delta_i(v)$

[Diagram: A graph with nodes labeled S and v, and edges showing shortest path at i]
For every augmenting path, some edge is critical.
Critical edges are removed in next residual graph.
KEY IDEA: HOW MANY TIMES CAN AN EDGE BE CRITICAL?

\[
\frac{V}{2} \text{ times}
\]
Outline of the argument
first time \((u,v)\) is critical:
time $i+1$: $(u,v)$ is critical:

$$\delta_{i+1}(s,v) \geq \delta_i(s,u) + 1$$

(time $i+1$: $(u,v)$ STRIKES BACK)

(time $j$: Edge $(u,v)$ STRIKES BACK)
time $i+1$: (u,v) is critical:
\[ \delta_{i+1}(s,v) \geq \delta_i(s,u) + 1 \]

\[ \delta_{i+1}(s,v) = \delta_i(s,u) + 1 \]

diagram:

- Time $i+1$: (u,v) is critical
- Edge (u,v) STRIKES BACK

Time $j$: Edge (u,v) STRIKES BACK

\[ \delta_j(s,u) = \delta_j(s,v) + 1 \]
time $j$: Edge $(u,v)$ STRIKES BACK

\[
\begin{align*}
\delta_{i+1}(s, v) & \geq \delta_i(s, u) + 1 \\
\delta_j(s, u) & = \delta_j(s, v) + 1
\end{align*}
\]
time $k$: RETURN OF THE $(u,v)$ critical

$$\delta_k(s, u) \geq \delta_i(s, u) + 2$$

**QUESTION**: How many times can $(u,v)$ be critical?
- edge critical only \( \frac{V}{2} \) times.
- there are only \( E \) edges.

Ergo, total # of augmenting paths: \( \frac{EV}{2} \)

time to find an augmenting path: \( \Theta(E^2V) \) (BFS)

total running time of E-K algorithm: \( \Theta(E^2V) \)
\( O(E|f^*|) \)

\( \Theta(E^2V) \)

Tarjan push-relabel

faster push-relabel \( \Theta(V^3) \)
APPLICATIONS OF MAX FLOW
MAXIMUM BIPARTITE MATCHING
MAXIMUM BIPARTITE MATCHING
BIPARTITE MATCHING

PROBLEM:
Chapter 7
Network Flow

The Problem

One of our original goals in developing the Maximum-Flow Problem was to be able to solve the Bipartite Matching Problem. We now show how to do this.

Recall that a bipartite graph $G=(V,E)$ is an undirected graph whose node set can be partitioned as $V=X \cup Y$, with the property that every edge $e \in E$ has one end in $X$ and the other end in $Y$.

A matching $M$ in $G$ is a subset of the edges $M \subseteq E$ such that each node appears in at most one edge in $M$.

The Bipartite Matching Problem is that of finding a matching in $G$ of largest possible size.

Designing the Algorithm

The graph defining a matching problem is undirected, while flow networks are directed; but it is actually not difficult to use an algorithm for the Maximum-Flow Problem to find a maximum matching.

Beginning with the graph $G$ in an instance of the Bipartite Matching Problem, we construct a flow network $G'$ as shown in Figure 7.9. First we direct all edges in $G$ from $X$ to $Y$. We then add a node $s$, and an edge $(s,x)$ from $s$ to each node in $X$. We add a node $t$, and an edge $(y,t)$ from each node in $Y$ to $t$. Finally, we give each edge in $G'$ a capacity of 1.

We now compute a maximum $s$-$t$ flow in this network $G'$. We will discover that the value of this maximum is equal to the size of the maximum matching in $G$. Moreover, our analysis will show how one can use the flow itself to recover the matching.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure7.9.png}
\caption{(a) A bipartite graph. (b) The corresponding flow network, with all capacities equal to 1.}
\end{figure}
EDGE-DISJOINT PATHS
ALGORITHM
VERTEX-DISJOINT PATHS
## BASEBALL ELIMINATION

<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>L</th>
<th>Left</th>
<th>A</th>
<th>P</th>
<th>N</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATL</td>
<td>83</td>
<td>71</td>
<td>8</td>
<td>-</td>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>PHL</td>
<td>80</td>
<td>79</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>NY</td>
<td>78</td>
<td>78</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>MONT</td>
<td>77</td>
<td>82</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>
### BASEBALL ELIMINATION

<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>L</th>
<th>Left</th>
<th>N</th>
<th>B</th>
<th>Bo</th>
<th>T</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>NY</td>
<td>75</td>
<td>59</td>
<td>28</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>BAL</td>
<td>71</td>
<td>63</td>
<td>28</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>BOS</td>
<td>69</td>
<td>66</td>
<td>27</td>
<td>8</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOR</td>
<td>63</td>
<td>72</td>
<td>27</td>
<td>7</td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DET</td>
<td>49</td>
<td>86</td>
<td>27</td>
<td>3</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Against
<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>L</th>
<th>Left</th>
<th>N</th>
<th>B</th>
<th>Bo</th>
<th>T</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>NY</td>
<td>75</td>
<td>59</td>
<td>28</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>BAL</td>
<td>71</td>
<td>63</td>
<td>28</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>BOS</td>
<td>69</td>
<td>66</td>
<td>27</td>
<td>8</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOR</td>
<td>63</td>
<td>72</td>
<td>27</td>
<td>7</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DET</td>
<td>49</td>
<td>86</td>
<td>27</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ALGORITHMS FOR MAX FLOW
CUTS

DEF OF A CUT:

COST OF A CUT:

\[ ||S, T|| = \]
LEMMA: [MIN CUT] FOR ANY $f, (S, T)$
FOR ANY $f, (S, T)$ IT HOLDS THAT $|f| \leq ||S, T||$

EXAMPLE: