Flow Networks

Setting: a directed graph where

- each edge e has positive capacity $C(e)$
- one node is source (often called s)
- another node is sink (often called t)

A flow assigns to each edge e a number $F(e)$ with

$$0 \leq F(e) \leq C(e)$$

which satisfies flow conservation:

> *for all nodes except source and sink,*
> *sum of incoming flow equals sum of outgoing flow*

Value of flow: sum of outgoing flows from source

- minus sum of incoming flow, but typically zero
- will equal sum of incoming flow to sink

Goal: find flow with maximum value

Finding Maximal Flow

An augmenting path is an

acyclic path from source to sink

Naive attempt: if no augmenting path then zero flow; otherwise

1. find augmenting path π
2. let m be minimum capacity of edges in π
3. for each edge in π, subtract m from its capacity (remove edge if capacity is now 0)
4. recursively build flow F for resulting network (with smaller total capacity so recursion will terminate)
5. return F, after adding m to all edges in π

This approach

- often works
- but often does not work

Ford-Fulkerson Method

To allow us to (partially) undo flow assignments, we augment step 3:

- for each edge in π, subtract m from its capacity (remove edge if capacity is now 0)

so that for each edge in π it also

- adds an edge with capacity m in the reverse direction (or adds m to an already existing edge)

This is the Ford-Fulkerson method which

- is highly non-deterministic
- returns a maximum flow when (if!) it terminates
- always terminates when capacities are integers
- never decreases flow of edge from source or into sink

Running time for integer network:

- to find augmenting path: $O(|E|)$ (may use DFS)
- number of iterations: $O(M)$ with M the max flow

Thus the total running time is in $O(M|E|)$

The Edmonds-Karp Algorithm

Recall Ford-Fulkerson: we repeatedly

1. select arbitrary augmenting path
2. update network so as to allow us to (partially) undo flow assignments.

With M the maximum flow, this will

- terminate after at most M iterations
- but M iterations may actually be needed!

To reduce the risk of many iterations,

choose an augmenting path with fewest edges.

This is the Edmonds-Karp algorithm:

- we then need at most $O(|V||E|)$ iterations
- and inherit the bound of M iterations
- with each iteration taking time in $O(|E|)$

The total running time is thus in

$$O(\min \left\{ \begin{array}{l} |V| \cdot |E|^2 \\ M \cdot |E| \end{array} \right\})$$

Cuts

A cut in flow network (V, E) is a subset U of V where

- the source is in U
- but the sink is not in U.

The capacity of a cut U is given by

$$C(U) = \sum_{u \in U,\, w \notin U} C(u \to w)$$

A cut with smallest capacity is a "bottleneck"; indeed:

- with M the maximum (value of a) flow
- and C the minimum (capacity of a) cut

we have $M = C$ as we shall now argue.

Flows Cannot Exceed Cuts

$$\begin{aligned}
V(F) &= \sum_{v\in V} F(s,v) - \sum_{v\in V} F(v,s) + 0 \\
&= \sum_{v\in V} (F(s,v) - F(v,s)) + \\
&\quad \sum_{u\in U\setminus\{s\}, v\in V} (F(u,v) - F(v,u)) \\
&= \sum_{u\in U, v\notin U} F(u,v) + \sum_{u\in U, v\in U} F(u,v) \\
&\quad - \sum_{u\in U, v\in U} F(v,u) - \sum_{u\in U, v\notin U} F(v,u) \\
&= \sum_{u\in U, v\notin U} F(u,v) - \sum_{u\in U, v\notin U} F(v,u) \\
&\leq \sum_{u\in U, v\notin U} F(u,v) \leq \sum_{u\in U, v\notin U} C(u,v) = C(U)
\end{aligned}$$

Finding Bottlenecks

Theorem: with M the maximal flow ,

there exists a cut U with capacity M.

Proof for $M = 0$: then there is no augmenting path, so

$$U = \{v \mid \text{ path from source to } v\}$$

- is a cut since it contains the source but not the sink
- has capacity zero since there is no edge from a node $\in U$ to a node $\notin U$.

Proof for $M > 0$: apply Ford-Fulkerson until no augmenting path; there is (cf above) a cut U that

- has capacity zero in the resulting network
- but then U has capacity M in the original network.

We see that we can find a minimum cut without (exponential) brute-force search.

Bipartite Matching

A Bipartite Graph is an undirected graph (V, E) where there exists V_1, V_2 such that

- $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$
- each edge $\in E$ is:
  between a node $\in V_1$ and a node $\in V_2$

A Matching is a subset E' of E such that for all $v \in V$:
v is the end point of at most one edge in E'

Goal: find a matching that is maximal
in that no other matching has more edges

Construct Maximal Matching

From bipartite graph $(V_1 \cup V_2, E)$ we construct a flow network (V', E'):

- $V' = V_1 \cup V_2 \cup \{s, t\}$ where s and t are fresh
- s is the source; t is the sink
- each edge has capacity 1
- E' is given by

$$\begin{array}{rl} & \{(s \rightarrow v_1) \mid v_1 \in V_1\} \\ \cup & \{(v_1 \rightarrow v_2) \mid (v_1, v_2) \in E\} \\ \cup & \{(v_2 \rightarrow t) \mid v_2 \in V_2\} \end{array}$$

Fact: the network

has a flow with value M

iff the bipartite graph has a matching with M edges.

Thus a maximal matching can be

- constructed by the Ford-Fulkerson method
- where a node once matched will remain matched (though may switch partner)