CSCI270 Week 9

Jacob Ma

November 3, 2022

1 Max-Flow / Min-Cut

Problem 1: Intuitive question for Max-Flow

Given a graph with edge capacities (think: traffic capacities, or widths of pipes), how much "flow" can we route from s to t at "steady state"?

Definition 1.1: $s-t$ flow

An $s-t$ flow f in the network G (with edge capacities $c(e)$) is a function mapping edges to the amount of flow on the edge. $f : E \to \mathbb{R}^+$
f must satisfy the following:

(1) **Conservation**: for all nodes v other than the source s and the sink t, the amount of flow going in is the same as the amount of flow going out:

$$\forall v \neq s, t : \sum_{e \text{ into } v} f_e = \sum_{e \text{ out of } v} f_e$$

(2) **Non-negativity and Capacity**:

$$0 \leqslant f(e) \leqslant c(e)$$

The value of the flow is $v(f)$ = sum of the flow on all edges out of s:

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

Lemma 1.1.1

$v(f)$ is also equal to the total flow on all edges into t.

Definition 1.2: Alternative Definition of Flows

A flow is a collection of k $s-t$ paths $P_1, P_2, \cdots, P_k$ with associated values $a_1, a_2, \cdots, a_k$ such that for each edge

e, the total capacity is not violated:

$$\sum_{i:e\in P_i} a_i \leqslant c(e)$$

The value of the $s-t$ flow is then the sum of all a_i.

Lemma 1.2.1: Path Decomposition Lemma

These two definitions of flows are equivalent.

Given a flow represented in one form, we can compute the other form in polynomial time.

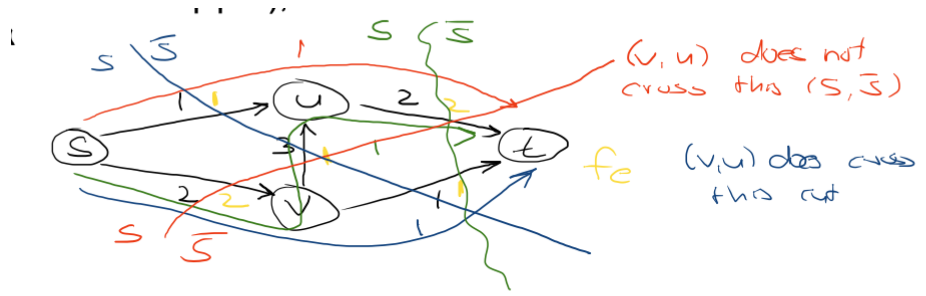
If $f(e)$ are given, then we can find paths P_i such that the number of paths $k \leqslant m$ (#edges).

Proof. Not very hard, slightly tedious, repeated BFS and induction. □

Remember: A cut of a graph is simply a partition of the vertices into $S, \overline{S}$.

An $s-t$ cut is a cut such that s is in S, and t is in $\overline{S}$.

An edge $e = (u, v)$ **crosses** the $s-t$ cut $(S, \overline{S})$ if u is in S and v is in $\overline{S}$.



S
$\bar{S}$
S
$\bar{S}$
S
$\bar{S}$
(v, u) does not cross this (S, $\bar{S}$)
(v,u) does cross this cut
fe

Example of $s-t$ cut

Problem 2: Two Natural Optimization Problems

(1) **Maximum $s-t$ flow**: Find an $s-t$ flow f maximizing $v(f)$.

(2) **Minimum $s-t$ cut**: Find an $s-t$ cut $(S, \overline{S})$ minimizing the total capacity of edges crossing the cut:

$$\sum_{\substack{e=(u,v):\\ u\in S, v\in \overline{S}}} c(e) = c(S, \overline{S})$$

Minimum cuts intuitively correspond to "bottlenecks" in the graph.

Observation(proved later): Minimum cuts are bottlenecks for flows:

$$v(f) \leqslant c(S, \overline{S})$$

for all flows f and all $s-t$ cuts $(S, \overline{S})$.

Theorem 1.3: Max-Flow Min-Cut Theorem, Ford/Fulkerson

Let G be a graph with non-negative edge capacities $c(e)$, and s, t nodes.

(1) There is an algorithm that computes a maximum $s-t$ flow f and a minimum $s-t$ cut $\left(S, \overline{S}\right)$ in polynomial time.

(2) If all edge capacities $c(e)$ are integers, then the algorithm returns a flow f in which all $f(e)$ are integers.

(3) The maximum flow and minimum cut satisfy $v(f) = c\left(S, \overline{S}\right)$.

2 Application: Maximum Bipartite Matching

Problem 3: Maximum Bipartite Matching

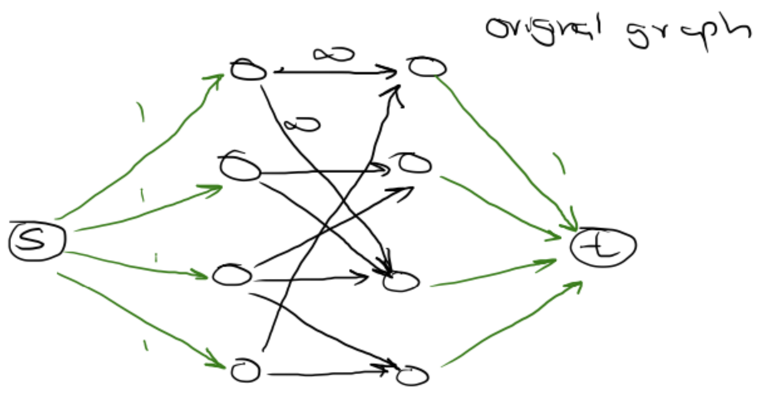
Given a bipartite graph $G = (X + Y, E)$.
Find a matching M of as many edges as possible (maximum cardinality $|M|$). Remember:

(1) Bipartite means that all edges go between X and Y (no edges inside X or inside Y).

(2) Matching M is a set of edges such that no node is incident on more than one edge of M.

2.1 Reduction to Maximum $s-t$ Flow:

- Add a new source s and a new sink t.
- Connect s to all nodes of X with new edges of capacity 1.
- Connect all of Y to t with new edges of capacity 1.
- Make each edge from X to Y directed, and give it capacity ∞ (or anything at least 1).



Construction of flow network

Algorithm:

```
    - Build the flow network as described above.
    - Compute an integer maximum s-t flow (using Ford- Fulkerson).
    - Include in M all the original edges with f(e) = 1.
```

Theorem 2.1

M will be a maximum matching.

Notice: This is a reduction from Maximum Bipartite Matching to Maximum $s-t$ Flow.
We were able to give this reduction even though we don't know the algorithm for maximum flow yet.

Correctness Proof for the reduction. The algorithm finds a maximum cardinality matching.

(1) The algorithm finds a valid matching.

Because each node on the left has at most one unit of flow coming in (by capacity constraint on the incoming edge), it has at most one unit of flow going out.

So at most one of its outgoing edges can have $f(e) = 1$.

So at most one of its incident edges in the bipartite graph is put into M.

Same argument for nodes on the right with flow going out.

$\implies M$ is a matching.

(2) The matching is of maximum size.

The matching M we output has size $|M| = v(f)$, where f is the (maximum) flow returned by Ford-Fulkerson.

This is because Ford-Fulkerson returned an *integer* max-flow, so all of the units of flow crossing the middle layer must be on edges with flow 1 each.

Assume that there is a bigger matching M' with $|M'| > |M|$.

We want to show that then, f was not a max-flow (contradiction, because Ford-Fulkerson finds a max-flow).

To do so, we will use M' to build a new flow f', and show that $v(f') > v(f)$.

To define f', for each edge $e(i) = (u(i), v(i))$ we send one unit of flow from s to $u(i)$, then from $u(i)$ to $v(i)$, then from $v(i)$ to t (along the path $s - u(i) - v(i) - t$).

Because M' was a matching, all capacity constraints out of s and into t are satisfied. (In the middle, capacities are infinite.) So f' would be a valid flow.

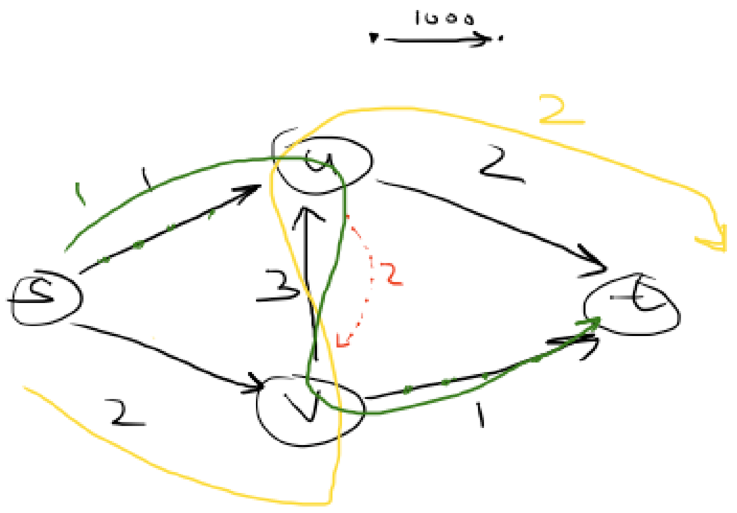
Because each $e(i)$ resulted in sending one unit of flow,

$$v(f') = |M'| > |M| = v(f)$$

So f was not a max-flow $\implies$ Contradiction.

□

3 Proving the Max-Flow/Min-Cut Theorem



3.1 Basic Greedy Algorithmic Idea

Start with the empty flow $f = 0$.
In each iteration, find a path P from s to t on which we can still add flow. Send as much flow on P as possible.
Terminate when no more path exists.

This may terminate with a flow that's not a max-flow - see example above.

To fix this problem, we want to be able to "undo" flow.
To make that formal, we define **backward edges** and **residual graphs**.

3.2 Backward Edges & Residual Graphs

Definition 3.1: Backward Edges & Residual Graphs

Given an input graph G with capacities c and an $s - t$ flow f, the **residual graph** $G(f)$ with respect to the flow f is defined as follows:

- Whenever $f(e) < c(e)$, it contains the **forward edges** e with
$$c'(e) = c(e) - f(e)$$
- Whenever $e = (u, v)$ has flow $f(e) > 0$, $G(f)$ contains the **backward edge** (v, u) with $c'(v, u) = f(e)$.

3.3 Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for all edges e.
- while the residual graph $G(f)$ still contains an $s - t$ path
  - Let P be an $s - t$ path in $G(f)$
  - Augment f along P (add as much flow as possible along P)

Augmentation of f along P:

- Let $P = (e(1), e(2), \ldots, e(k))$
- Let $\epsilon = \min_{\text{all edges } e(i) \text{ in } P} c'(e(i))$
- For each edge $e(i)$ in P:
  - If $e(i)$ is a forward edge, then set $f'(e(i)) = f(e(i)) + \epsilon$
  - Otherwise, $e(i) = (v, u)$ is a backwards edge. Set $f'(u, v) = f(u, v) - \epsilon$
- Return f'