Some additional notes on Max Flow and Min Cut

1 Flows and Cuts in Networks

Recall that a *network* is a directed graph with *capacities* associated to edges, and two special nodes s and t.

Definition 1 *A network is a directed graph $G(V, E)$, in which*

- *a vertex $s \in V$ and a vertex $t \in V$ are specified as being the source node and the sink node, respectively.*
- *every directed edge $(u, v) \in E$ has a positive capacity $c(u, v) > 0$ associated to it. If both the edges (u, v) and (v, u) belong to E, we allow their capacities to be different.*

Sometimes we will write expressions that include capacities between pairs of vertices that are not connected by an edge. In that case the convention is that the capacity is zero.

A *flow* in a network is a specification of how to route "stuff" from s to t so that no link is used beyond its capacity, and so that every link, except the sender s and the receiver t, relays out "stuff" at exactly the same rate at which it receives from other vertices. In the motivating example of a communication network, if nodes where sending out less data than they receive, there would be data loss, and they cannot send out more data than they receive because they are simply forwarding data. Formally, we have the following definition.

Definition 2 *A flow in an network (G, s, t, c) is an assignment of a non-negative number $f(u, v)$ to every ordered pair (u, v) such that either (u, v) or (v, u) is an edge such that*

- *For every pair (u, v), $f(u, v) \leq c(u, v)$;*

- *For every pair* (u, v), $f(u, v) = -f(v, u)$;
- *For every vertex* $v \in V$, $\sum_{u \in V} f(u, v) = 0$

where we follow the convention that $f(u, v) = 0$ *if neither* (u, v) *nor* (v, u) *is an edge. This convention will simplify various expressions in the analysis. An algorithm for the maximum flow problem does not need to provide an output for such pairs.*

The *size*, or *value*, or *cost* of the flow (the throughput of the communication, in the communication network example) is

$$val(f) := \sum_{v} f(s, v)$$

In the *maximum flow* problem, given a network we want to find a flow of maximum value.

We also defined the capacity of a cut as follows.

Definition 3 (Cut) *A cut in a network* $(G = (V, E), s, t, c)$*, is partition* (L, R) *of the set of vertices* V *into two sets, such that* $s \in L$ *and* $t \in R$*. We will usually identify a cut with the set* L *that contains* s*. The capacity of a cut is the quantity*

$$c(L) := \sum_{u \in L, v \notin L} c(u, v)$$

The motivation for the definition is the following: let L be a cut in a network, that is, a set of vertices that contains s and does not contain t. Consider the set of edges $\{(u, v) \in E : u \in L \wedge v \notin L\}$. If we remove those edges from the network, then it is not possible to go from any vertex in L to any vertex outside L and, in particular, it is not possible to go from s to t. This means that all the flow from s to t must pass through those edges, and so the total capacity of those edges (that, is $c(L)$) is an upper bound to the cost of any feasible flow.

Even though what we just said is rather self-evident, let us give a rigorous proof, because this will help familiarize ourselves with the techniques used to prove rigorous results about flows. (Later, we will need to prove statements that are far from obvious.)

Lemma 4 *If* (G, s, t, c) *is a network, and* f *is a feasible flow, then, for every cut* (L, R)*,*

$$val(f) \leq c(L)$$

PROOF: We have

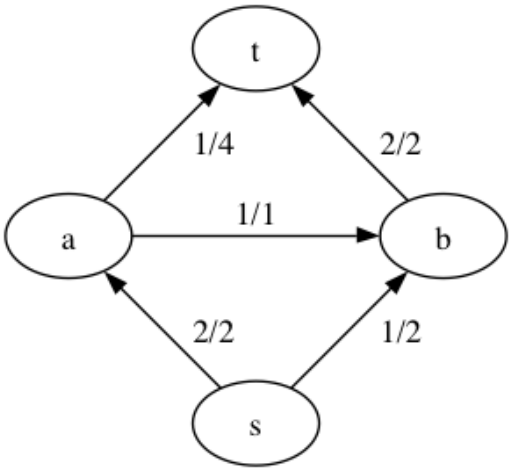
$$
\begin{aligned}
val(f) &= \sum_{v} f(s,v) && (1)\\
&= \sum_{v} f(s,v) + \sum_{u \in L - \{s\}} \sum_{v \in V} f_{u,v} && (2)\\
&= \sum_{u \in L} \sum_{v \in R} f(u,v) && (3)\\
&\leq \sum_{u \in L} \sum_{v \in R} c(u,v) && (4)\\
&= c(L) && (5)
\end{aligned}
$$

where the first line is the definition of the value of a cut, the second line is obtained by adding the conservation constraints for nodes in $L - \{s\}$, the third line collects the sums, the fourth line uses the capacity constraints, and the last line is the definition of capacity of a cut. □

2 The Ford-Fulkerson Method

So Lemma 4 gives us a way to "certify" the optimality of a flow, if we are able to find a flow and a cut such that the cost of the flow is equal to the capacity of the cut.

Consider now the complementary question: how do we see if a given flow in a network can be improved? That is, what is a clear sign that a given flow is not optimal? If we see a path from s to t such that all the edges are used at less than full capacity, then it is clear that we can push extra flow along that path and that the solution is not optimal. Can there be other cases in which a given solution is not optimal? Indeed there can.Consider the following network and feasible flow (the labels on each edge are the flow and the capacity)



The flow has size 3, which was not optimal (can you see that a flow of size 4 exists?), but if we look at the three possible paths from s to t, that is, $s \to a \to t$, $s \to a \to b \to t$, and $s \to b \to t$, they each involve an edge used at full capacity.

However, let us reason in the following way: suppose that, in addition to the edges shown in the last picture, there were also an edge of capacity 1 from b to a. Then we would have had the path $s \to b \to a \to t$ in which every edge has one unit of residual capacity, and we could have pushed an extra unit of flow along that path. In the resulting solution, a sends one unit flow to b, and b sends one unit of flow to a, a situation that is perfectly equivalent to a and b not sending each other anything, so that the extra edge from b to a is not needed after all. In general, if we are sending $f(u, v)$ units of flow from u to v, then we are effectively increasing the capacity of the edge from v to u, because we can "simulate" the effect of sending flow from v to u by simply sending less flow from u to v. These observations motivate the following definition:

Definition 5 (Residual Network) *Let $N = (G, s, t, c)$ be a network, and f be a flow. Then the residual network of N with respect to f is a network in which the edge u, v has capacity*

$$c(u, v) - f(u, v)$$

The idea is that, in the residual network, the capacity of an edge measures how much *more* flow can be pushed along that edge in addition to the flow f, without violating the capacity constraints. The edge (u, v) starts with capacity $c(u, v)$, and $f(u, v)$ units of that capacity are taken by the current flow; in addition, we have $f(v, u)$ additional units of "virtual capacity" that come from the fact that we can reduce the flow from v to u.

An *augmenting path* in a network is a path from s to t in which every edge has positive capacity in the residual network. For example, in our last picture, the path $s \to b \to a \to t$ is an augmenting path.

The *Ford-Fulkerson* algorithm is a simple greedy algorithm that starts from an empty flow and, as long as it can find an augmenting path, improves the current solution using the path.

Algorithm *Ford-Fulkerson*

- Input: network $(G = (V, E), s, t, c)$
- for all u, v: $f(u, v) = 0$
- for all u, v: $r(u, v) = c(u, v)$
- while there is a path p from s to t such that all edges (u, v) in the path p have positive residual capacity $r(u, v) > 0$:
  - let $r_{\min} = \min_{(u,v)\in p} \; r(u, v)$ be the smallest of the residual capacities of the edges of p
  - for all (u, v) in p:
    - $f(u, v) = f(u, v) + c_{\min}$
    - $f(v, u) = f(v, u) - c_{\min}$
    - $r(u, v) = r(u, v) - c_{\min}$
    - $r(v, u) = r(v, u) + c_{\min}$
- return f

At every step, the algorithm increases the cost of the current solution by a positive amount $c'_{\min}$ and, the algorithm converges in finite time to a solution that cannot be improved via an augmenting path.

The following theorem shows that the Ford-Fulkerson algorithm is optimal, and it proves the important fact that whenever a cut is optimal, its optimality can always be proved by exhibiting a cut whose capacity is equal to the cost of the flow.

Theorem 6 (Max Flow-Min Cut) *The following three conditions are equivalent for a flow f in a network:*

1. *There is a cut whose capacity is equal to the cost of f*
2. *The flow f is optimal*
3. *There is no augmenting path for the flow f*

PROOF: We have already proved that (1) implies (2), and it is clear that (2) implies (3), so the point of the theorem is to prove that (3) implies (1).

Let f be a flow such that there is no augmenting path in the residual network. Take A to be the set of vertices reachable from s (including s) via edges that have positive capacity in the residual network. Then:

- $s \in A$ by definition
- $t \notin A$ otherwise we would have an augmenting path.

So A is a cut. Now observe that for every two vertices $a \in A$ and $b \notin A$, the capacity of the edge (a, b) in the residual network must be zero, otherwise we would be able to reach b from s in the residual network via positive-capacity edges, but $b \notin A$ means that no such path can exist.

Recall that the residual capacity of the edge (a, b) is defined as $c(a, b) - f(a, b)$ so the condition of zero residual capacity is the same as the condition $c(a, b) = f(a, b)$.

Now just observe that the cost of the flow is

$$val(f) = \sum_{a \in A, b \notin A} f(a, b) = \sum_{a \in A, b \notin A} c(a, b) = c(A)$$

and so the capacity of the cut is indeed equal to the cost of the flow. □

3 The Edmonds-Karp Algorithm

The Edmonds-Karp algorithm is an implementation of the Ford-Fulkerson algorithm in which, at every step, we look for an $s \to t$ path in the residual network using BFS. This means that, if there are several possible such paths, we pick one with a minimal number of edges.

From now on, when we refer to a "shortest path" in a network, we mean a path that uses the fewest edges, and the "length" of a path is the number of edges. The "distance" of a vertex from s is the length of the shortest path from s to the vertex.

BFS can be implemented in $O(|V + E|) = O(|E|)$ time, and so to complete our analysis of the algorithm we need to find an upper bound to the number of possible iterations.

The following theorem says that, through the various iterations, the length of the shortest path from s to t in the residual network can only increase, and it does increase at the rate of at least one extra edge in the shortest path for each $|E|$ iterations.

Theorem 7 *If, at a certain iteration, the length of a shortest path from s to t in the residual network is ℓ, then at every subsequent iteration it is $\geq \ell$. Furthermore, after at most $|E|$ iterations, the distance becomes $\geq \ell + 1$.*

Clearly, as long as there is a path from s to t, the distance from s to t is at most $|V|-1$, and so Theorem 8 tells us that, after at most $|E| \cdot (|V| - 1)$ iterations, s and t must become disconnected in the residual network, at which point the algorithm terminates. Each iteration takes time $O(|E|)$, and so the total running time is $O(|V| \cdot |E|^2)$.

Let us now prove Theorem 8.

PROOF: Suppose that, after some number T of iterations, we have the residual network $R = (V, E'), s, t, c')$ and that, in the residual network, the length of the shortest path from s to t is ℓ. Construct a BFS tree starting at s, and call V_1, V_2, ..., V_k, ..., the vertices in the first, second, k-th, ..., layer of the tree, that is, the vertices whose distance from s is 1, 2, ..., and so on. Note that every edge (u, v) in the network is such that if $u \in V_i$ and $v \in V_j$ then $j \leq i + 1$, that is, nodes can go from higher-numbered layer to lower numbered layer, or stay in the same layer, or advance by at most one layer.

Let us call an edge (u, v) a *forward* edge if, for some i, $u \in V_i$ and $v \in V_{i+1}$. Then a shortest path from s to t must use a forward edge at each step and, equivalently, a path that uses a non-forward edge at some point cannot be a shortest path from s to t.

What happens at the next iteration $T + 1$? We pick one of the length-ℓ paths p from s to t and we push flow through it. In the next residual network, at least one of the edges in p disappears, because it has been saturated, and for each edge of p we see edges going in the opposite direction. Now it is still true that for every edge (u, v) of the residual network at the next step $T+1$, if $u \in V_i$ and $v \in V_j$, then $j \leq i+1$ (where $V_1, \ldots$ are the layers of the BFS tree of the network at iteration T), because all the edges that we have added actually go from higher-numbered layers to lower-numbered ones. This means that, at iteration $T + 1$ the distance of t from s is still at least ℓ, because $t \in V_\ell$ and, at every step on a path, we can advance at most by one layer.

(Note: we have proved that if the distance of t from s is ℓ at one iteration, then it is at least ℓ at the next iteration. By induction, this is enough to say that if will always be at least ℓ in all subsequent iterations.)

Furthermore, if there is a length-ℓ path from s to t in the residual network at iteration $T + 1$, then the path must be using only edges which were already present in the residual network at iteration T and which were "forward edges" at iteration T. This also means that, in all the subsequent iterations in which the distance from s to t remains ℓ, it is so because there is a length-ℓ path made entirely of edges that were forward edges at iteration T. At each iteration, however, at least one of those edges is saturated and is absent from the residual network in subsequent steps, and so there

can be at most $|E|$ iterations during which the distance from s to t stays ℓ. $\square$