We devote this lecture to the network flow problem and the max-flow min-cut theorem. A number of other problems can be cast into a maximum flow or minimum cut problem, and so knowing how to solve a network flow problem can come in very handy. In the next lecture we will see a number of reductions from other problems to network flow.

4.1 Problem setup

We first describe the problem informally. Imagine that we have a network of pipes each with a certain crossection. The network has a single inlet and a single outlet. Our goal is to send water through the pipes from the inlet to the outlet at as large a "rate" as possible. There are two main constraints on the rate at which we can send water through the pipes: the rate through a pipe cannot exceed the crossection of that pipe, and at any junction in the network, the amount of water coming in must exactly equal the amount of water going out. The first is called a capacity constraint, and the second a flow conservation constraint.

Formally, a network flow is defined as follows.

Definition 4.1.1 *Let $G = (V, E)$ be a directed or undirected graph with a capacity function $c : E \mapsto \Re^+$, a source s and a sink t. We say that $f : V \times V \mapsto R^+$ is a flow if the following holds:*

$$\forall v \in V, \quad v \neq s \text{ and } v \neq t \text{ implies } \sum_{u \in V} f(u, v) = \sum_{u \in V} f(v, u)$$

We say that a flow is feasible for G (or just feasible if the graph and capacity function in question are obvious) if

$$\forall u, v \in V, \quad (u, v) \in E \text{ implies } f(u, v) \leq c((u, v)) \text{ and } (u, v) \notin E \text{ implies } f(u, v) = 0$$

We define the size of a flow f (denoted $|f|$) to be the total flow outgoing from s: $|f| = \sum_{u \in V} f(s, u) - \sum_{u \in V} f(u, s)$, or, the total flow incoming to t: $|f| = \sum_{u \in V} f(u, t) - \sum_{u \in V} f(t, u)$.

The maximum flow problem then asks for a feasible flow of maximum possible size in a given capacitated graph.

We first note that we can make a few simplifying assumptions about the problem.

- The graph G contains no self-loops (edges of the form (u, u)).
- The graph G contains no multiple edges since if e_1, e_2 are two edges from u to v with capacities c_1, c_2 then we can represent them by one edge e with capacity $c_1 + c_2$.
- The graph G is directed. If the graph contains an undirected edge (u, v), we can replace the edge by two directed edges $(u \to v)$ and $(v \to u)$, each with capacity equal to $c(u, v)$. It is easy to see that any feasible flow in the original graph is still feasible for the new graph.

Furthermore, we claim that any flow feasible in the new graph can be converted to a flow feasible for the original graph. We leave the details to the reader. These together imply that our assumption is without loss of generality.

4.2 The Ford-Fulkerson Algorithm

We now describe an algorithm for finding the max flow. As a first attempt, we may try a greedy approach as follows.

Repeat until all $s-t$ paths are saturated:

- Find an $s-t$ path with some remaining (positive) capacity.
- Saturate the path
- Repeat until no such paths exist

Unfortunately, this approach does not give us an optimal flow. Consider, for example, the network in Figure 4.2.1 (where the numbers in parantheses are the capacities and the other ones are the size of the flow). In the first iteration we choose the path through the middle and saturated it with 3 units of flow. This is a reasonable path to choose, since it has the maximum capacity of all $s-t$ paths. However, this leaves us with one remaining $s-t$ path which has a capacity of 1. Saturating this path will result in the termination of the algorithm, giving a flow of value 4. However, the maximum flow in this network is of size 6 (see Figure 4.2.2).

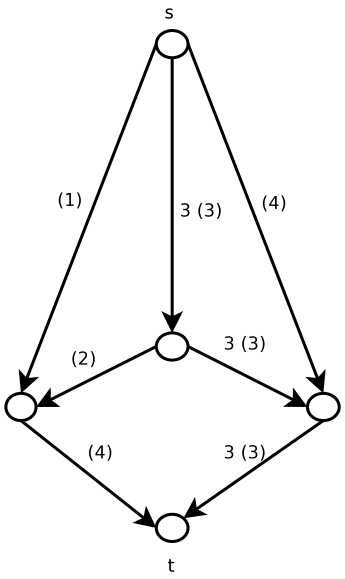

Figure 4.2.1: A greedy non-optimal flow

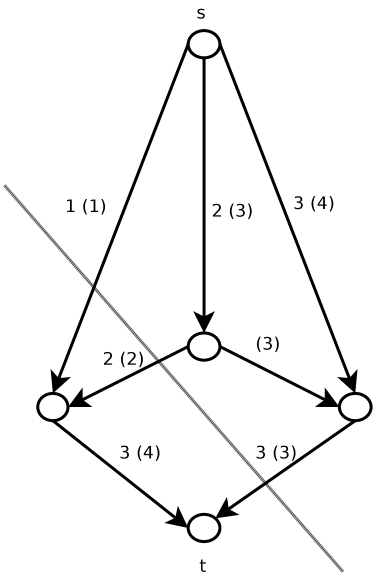

Figure 4.2.2: A min-cut and an optimal flow

The problem with the greedy approach is that it cannot revoke bad choices made in the past. The

Ford-Fulkerson algorithm remedies this issue by directly allowing the algorithm to "reverse" the direction of flow along edges, thereby correcting previous mistakes.

4.2.1 The algorithm

The general idea of the Ford-Fulkerson algorithm is similar to the greedy algorithm: at every step pick an *s-t* path through our graph saturate it. However, it finds paths in a "residual graph" which explicitly allows for the reversal of flow along previously used edges. The residual graph for a graph with a given flow is defined as follows.

Definition 4.2.1 *Let $G = (V, E)$ be an directed graph with a capacity function c. Let f be a feasible flow in this graph. Then we define the residual graph, $G_f = (V_f, E_f)$ and c_f in the following manner. $V_f = V$ and $E_f = \{(u, v) \mid u, v \in V \wedge (c(u, v) > f(u, v))\} \cup \{(u, v) \mid u, v \in V \wedge (f(v, u) > 0)\}$. We define $c_f(u, v) = c(u, v) - f(u, v) + f(v, u)$.*

This means that for every edge in our original graph, if it is saturated we remove it; then we either add a new edge in the opposite direction with capacity equal to the amount of flow on the original edge, or if there exists an edge in that direction we increase its capacity by the amount of the flow. For a non-saturated edge we decrease its capacity by the amount of the flow and add the amount of the flow to the edge going in the other direction. Figures 4.2.3 and 4.2.4 show a graph with a flow and it's corresponding residual graph.

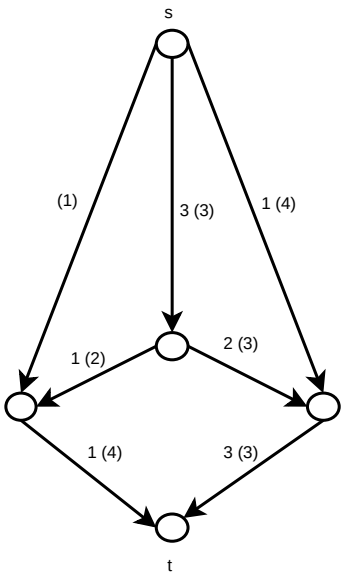

Figure 4.2.3: A graph with a feasible flow

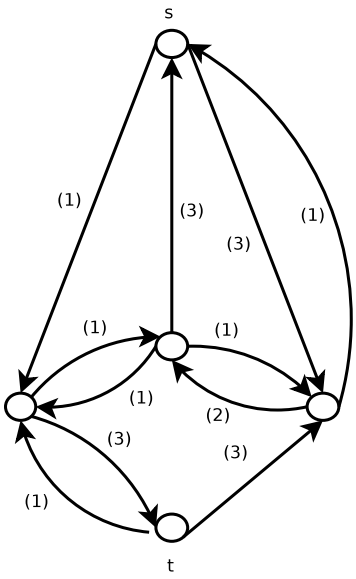

Figure 4.2.4: Residual graph of the flow in the previous figure

We also define the addition of two flows in the obvious fashion.

Definition 4.2.2 *Let $f : V \times V \mapsto \mathbb{R}^+$ and $f' : V \times V \mapsto \mathbb{R}^+$ then we define $g = f + f'$ in the following manner. $g(u, v) = max\{0, f(u, v) - f(v, u) + f'(u, v) - f'(v, u)\}$. It is easy to see that the*

conservation property holds for g.

Let us now proceed to the **Ford-Fulkerson** algorithm.

1. Start with $f : V \times V \mapsto \{0\}$ and $G = (V, E)$ an directed graph and a capacity function c.
2. Create G_f and c_f.
3. If s and t are disconnected in G_f, terminate.
4. Else, find a simple $s - t$ path P in G_f and let c' be the minimum capacity along this path. Let f' be the flow of capacity c' along P and 0 everywhere else. Let $f = f + f'$.
5. Go to Step 2.

The paths chosen in Step 4 are called "augmenting paths".

4.2.2 Flows and cuts

In order to analyze the algorithm, we first relate flows in graphs to cuts.

Definition 4.2.3 *An s-t cut is a set of edges C such that the graph $G' = (V, E \setminus C)$ contains no s-t path. The capacity of the cut is $cap(C) = \sum_{e \in C} c(e)$.*

Note that an s-t cut defines a partition on the set of edges V into $(S, \overline{S})$, such that $s \in S$ and $t \in \overline{S}$.

Lemma 4.2.4 *If we take F to be the set of all feasible flows, then for any s-t cut C, $cap(C) \geq max_{f \in F}|f|$.*

Proof: Since C is an s-t cut, any s-t path must contain at least one edge in C. So all the flow between s and t must cross C and occupy capacity along at least one edge in C. Thus any flow must have value less than $cap(C)$. ■

Corollary 4.2.5 *Let C^* be a minimum s-t cut of G and f^* be a max s-t flow of G, then $cap(C^*) \geq |f^*|$.*

Using this corollary, we see that if we can exhibit a min-cut that equals the value of our flow, it serves as a certificate of the optimality of our flow. This is illustrated in our earlier example by the cut in Figure 4.2.2 which has a capacity of 6, the same as the value of our flow.

4.2.3 Analysis

Theorem 4.2.6 *The Ford-Fulkerson algorithm described above finds the maximum flow if the capacities are integers.*

Proof: First note that at each step of the algorithm $|f|$ increases by at least a unit. So the algorithm must terminate in a finite number of steps, since we have only a finite amount of capacity leaving the source.

Next we note that the flow created at every step is actually a feasible flow in the graph G. We show this by induction. This is obvious for the empty flow — the base case of our induction. Let

f_i be the flow at iteration i, which is feasible by hypothesis, and suppose that P is the s-t path G_{f_i} selected by the algorithm in this iteration. Let $c_{f_i,P}$ be the minimum capacity along P in G_{f_i}.

Note that feasibility continues to be maintained for edges not on P. Suppose that (u, v) is any edge on the path P. We have $c_{f_i}(u, v) = c(u, v) - f_i(u, v) + f_i(v, u)$. By the definition of the addition of flows we can see that any flow that is a sum of two flows has the property that if $f(u, v) > 0$ then $f(v, u) = 0$. If $f_i(u, v) > 0$ then we have $0 < c_{f_i}(u, v) = c(u, v) - f_i(u, v)$ from feasibility of f_i and since $c_{f_i,P} \leq c_{f_i}(u, v)$ we get by simple algebra $f_{i+1} \leq c(u, v)$. If we have $f_i(v, u) > 0$ then $0 < c_{f_i}(u, v) = c(u, v)+f_i(v, u)$ and $f_{i+1}(u, v)$ is by definition either 0, or $f_{i+1}(u, v) = c_{f_i,P} - f_i(v, u)$ which then by simple algebra yields $f_{i+1}(u, v) \leq c(u, v)$.

To show that our resulting flow is a maximal flow first notice that $S = \{v \in V \mid$ there exists an s-v path in $G_f\}$ along with its complement gives us an s-t cut C in G (otherwise we would have an s-t path in G_f which is a contradiction to the fact that the algorithm terminated). We claim that the capacity of this cut is exactly equal to the size of our resulting flow f_r.

Consider some $(u, v) \in C$ with $v \notin S$. By the definition of S we must have $f_r(u, v) = c(u, v)$ since otherwise $c_{f_r}(u, v) > 0$ and we can get from s to v via u. We also have for any $(u, v) \in E$ where $v \in S$ and $u \notin S$ that $f_r(u, v) = 0$ since otherwise $c_{f_r}(v, u) = c(v, u) + f_r(u, v) > 0$ (because only one of $f_r(u, v)$, $f_r(v, u)$ can be non zero). Now this implies that the total flow exiting S is the capacity of C. That is,

$$\sum_{u \in S} \sum_{v \notin S} f_r(u, v) = \sum_{e \in C} c(e)$$

We can rewrite the total flow exiting S as follows.

$$\begin{aligned} \sum_{u \in S} \left(\sum_{v \in V} f_r(u, v) - \sum_{v \in S} f_r(u, v) \right) &= \sum_{u \in S} \left(\sum_{v \in V} f_r(u, v) - \sum_{v \in S} f_r(v, u) \right) \\ &= \sum_{u \in S} \left(\sum_{v \in V} f_r(u, v) - \sum_{v \in V} f_r(v, u) \right) \end{aligned}$$

Here the first equality follows just from regrouping, and the second follows from noting that $f_r(v, u) = 0$ for $u \in S$ and $v \notin S$.

However, the above sum is equal to $\sum_{v \in V} f_r(s, v) = |f|$, because conservation holds for all $u \in S$ other then s. So we get that the total amount of flow is exactly equal to the total flow exiting S, which is exactly equal to the capacity of C. ■

Corollary 4.2.7 Max-flow Min-cut theorem.
The size of the maximum flow is exactly equal to the capacity of the minimum cut.

Before we move on let us notice that even though we only proved that the Ford-Fulkerson algorithm works with integer capacities, it is easy to see that if we have rational capacities we can just change them by multiplying all of them by their smallest common denominator, and a maximum flow in such a graph will easily yield a maximum flow in our original graph.

We remark that when the edge capacities are all integers, the Ford-Fulkerson algorithm finds an integral flow (that is, $f_i(u, v)$ is an integer for all (u, v)). This property is important for some

applications of network flow as we will see in the following lecture.

Now let us turn to examining the running time of our algorithm. Every step in an iteration takes at most $O(m)$ time where m is the number of edges. An easy bound on the number of iterations could be for example $F = |f_r|$ the size of the maximal flow since in each iteration we increase the size of our flow by at least one. This gives us a bound of $O(mF)$. Now if we set $C = max_{e \in E} c(e)$ we get a bound of $O(m^2 C)$ or better yet if Δ_s is the number of edges exiting s we get $O(m\Delta_s c)$. This runing time is still only pseudo polynomial in our input though since we input the capacities as binary numbers and as such the size of the input is actually $\log_2(c)$.

The following example illustrates that if we get an unfortunate graph and choose our s, t paths in a bad fashion this worst case time could actually be reached. If we take the graph in Figure 4.2.5 and choose our paths to always include the middle edge of size 1 we actually have to iterate 2^{101} times.

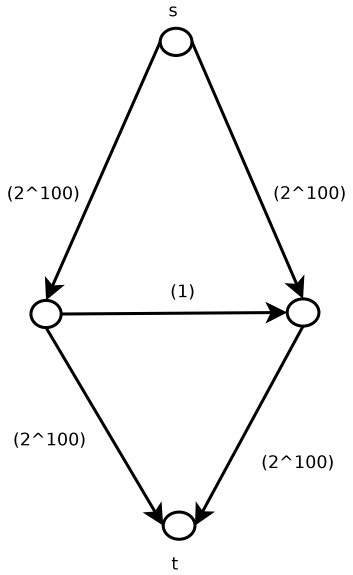

Figure 4.2.5: An example for which the Ford-Fulkerson algorithm may have a large running time

To handle this situation we can modify our algorithm in the following way. Let us define G_f^Δ to be the graph to be G_f without edges of capacity less then Δ. Then start by setting Δ to be half the maximum capacity in the graph. Let the Ford-Fulkerson algorithm run on G_f^Δ and after it's termination let $\Delta = \Delta/2$ and repeat until $\Delta = 1$. We can see that in each stage we will make at most $2m$ repetitions since each iteration increases flow by at least Δ and we have the total outgoing capacity from s at most degree of s times $\Delta * 2$ since all capacities in our graph are at most $\Delta * 2$. There will be at most $\log_2(c) + 1$ repetitions where c is the maximum capacity in our graph and we get a bound of $O(m^2 \log(c))$.

It actually turns out that if we run the Ford-Fulkerson algorithm making sure to choose the least path in every iteration we can get a bound of $O(m^2n)$, but we shall not show that here.