CS787: Advanced Algorithms

Scribe: Blayne Field, Dan Rosendorf
Lecturer: Shuchi Chawla
Topic: Network Flows
Date: Sept. 12 2007

In this lecture, we discussed the problem of maximum networks flows, which is a general problem to which many other problems can be reduced.

4.1 Problem Definition

Given: A graph $G = (V, E)$, which can be either directed or undirected, a capacity function $c : E \mapsto \Re^+$, along with a source s and a sink t. This can be thought of as a set of pipes, with the capacities being analogous to the cross-section of the pipe. The water is inserted into the network at the source s and ends up at the sink t. Our goal will be to find out what the maximum throughput of our network can be. Where by throughput in our analogy we mean the maximum amount of water per unit of time that can flow from s to t.

Let us now explain more precisely what we mean by our notation. In our problem we obviously need not consider loops in the graph (i.e. edges from v to v) since they add nothing to the throughput abilities of the system and we can also WLOG assume there are 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 $c_1 + c_2$.

This means that if we have a directed graph we can consider it to be in the form of some set V and a set E such that $E \subset V \times V$. For an undirected graph we shall consider E to be a subset of $[V]^2$ (the two element subsets of V) by abuse of notation we shall still write (u, v) for an edge instead of the more correct $\{u, v\}$. Very soon we will see that we can actually restrict ourselves to only the case of the directed graph but first let us give a definition.

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

$$\forall v \in V (v \neq s \wedge v \neq t) \rightarrow (\sum_{u \in V} f(u, v) = \sum_{u \in V} f(v, u))$$

In words for all nodes that are not the sink or source the amount of incoming flow is equal to the amount of outgoing flow, (i.e. there are no points of acummulation of flow) this is usually called the conservation property.

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

$$\forall u, v \in V \ ((u, v) \in E) \rightarrow (f(u, v) \leq c((u, v))) \wedge ((u, v) \notin E) \rightarrow (f(u, v) = 0)$$

In words the flow along any valid edge is smaller then that edge's capacity.

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

Goal: The goal is to find a feasible flow with maximum size.

Let us now notice a few things so we can from now on only consider directed graphs. Suppose we have an undirected graph $G = (V, E)$ with a capacity function c. We then define G' to be the directed graph with vertices V and edges $E' = \{(u, v); \{u, v\} \in E\}$. In other words for each edge in our original graph we put in two edges going each way. We define $c' : E' \mapsto \mathbb{R}^+$ by $c'((u, v)) = c(\{u, v\})$, which just means we set each of our two edges we created to have the same capacity as the one original edge.

Now we claim that the maximum flow in our new oriented graph has the same size as the maximum flow in our original graph and that it is easy to get the flow in the original graph from the flow in the new graph.

Lemma 4.1.2 *Let $G = (V, E)$ be an undirected graph and define G' as above. Let f_1 be a feasible flow in G and f_2 be a feasible flow in G' then there exist feasible flows f_1' in G' and f_2' in G such that $|f_1| = |f_1'|$ and $|f_2| = |f_2'|$.*

Proof: For f_1 define f_1' to just be f_1. Then obviously the conservation properties still hold and since we made both directions of our new edge have the capacity of the original edge f_1 is still feasible. Since we have not changed anything $|f_1| = |f_1'|$.

On the other hand for f_2 define $f_2'(u, v) = max\{0, f_2(u, v) - f_2(v, u)\}$ then th conservation property is still satisfied as one easilly check. Feasibility is satisfied thanks to the fact that f_2 is nonnegative and thus $|f_2(u, v) - f_2(v, u)|$ is at most $c((u, v))$. The equality of size can be checked in a straightforward manner. ■

Definition 4.1.3 *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 $(S, \bar{S})$, such that $s \in S$ and $t \in \bar{S}$

Lemma 4.1.4 *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 a cut, any s, t path must contain at least one edge in C, thus any flow must have value less than $cap(C)$.

Corollary 4.1.5 *Let C^* be a minimum s, t cut of G and f^* be a max 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 the cut below, which has a capacity of 6, the same as the value of our flow (see Figure 4.2.2).

4.2 Ford-Fulkerson Algorithm

Since greedy approaches often lead to an optimal solution, it seems natural to start trying to solve this problem with such an approach. We know that if we can find an s, t path with some positive capacity, we can increase the flow along that edge while maintaining feasability. This leads us to the following greedy approach:

- 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. As illustrated below in Figure 4.2.1, where the numbers in parantheses are the capacities and the other ones are the size of the flow) we chose 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, as we saw before, the maximum value of this network is 6. Thus we need to improve on this approach. We do this by using the "residual graph" of G, in what is known as the Ford-Fulkerson algorithm.

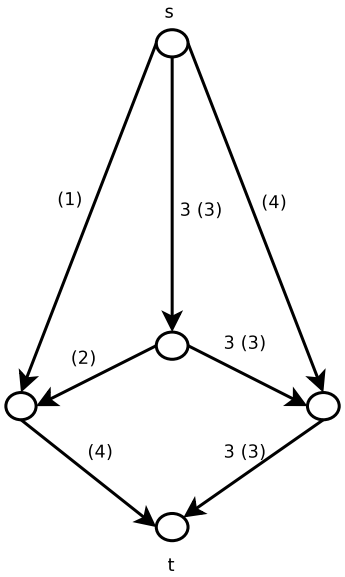

Figure 4.2.1: 3-3-3 flow

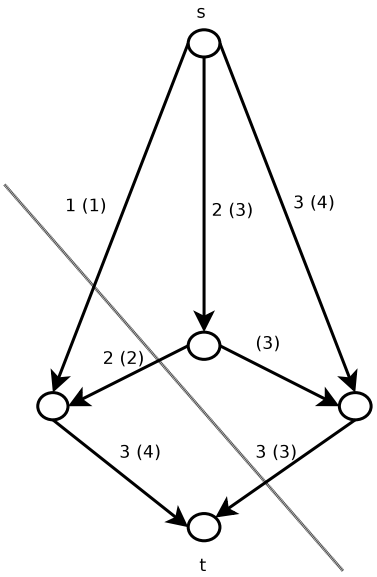

Figure 4.2.2: Cut with flow

4.2.1 Ford-Fulkerson Algorithm

The general idea of the Ford-Fulkerson algorithm is to choose an $s - t$ path through our graph saturate it, and create a new residual graph which has possibly different edges representing the flow and thus allowing us to sort of go back on our original flow if it is not maximal. We iterate this process until there is no $s - t$ path in our residual graph.

First let us define the residual graph for a graph with a given flow.

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* $E_f = \{(u,v)|u,v \in V \wedge (c(u,v) > f(u,v))\} \cup \{(u,v)|u,v \in V \wedge (f(v,u) > 0)\}$. *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 and we either add a new edge in the oposite direction with capacity equal to that of the flow or if there exists an edge in that direction we increase it's capacity by the amount of the flow. For a non saturated edge we decrease it's capacity by the amount of the flow and add the capacity 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.

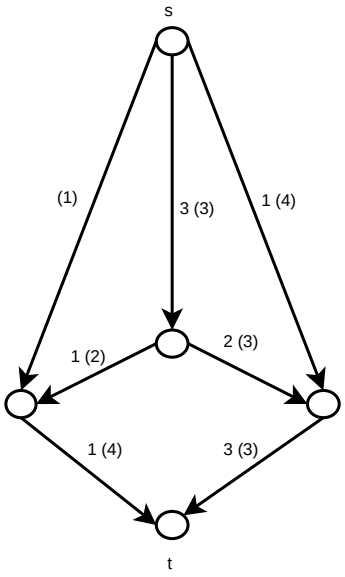

Figure 4.2.3: A graph with a feasible flow

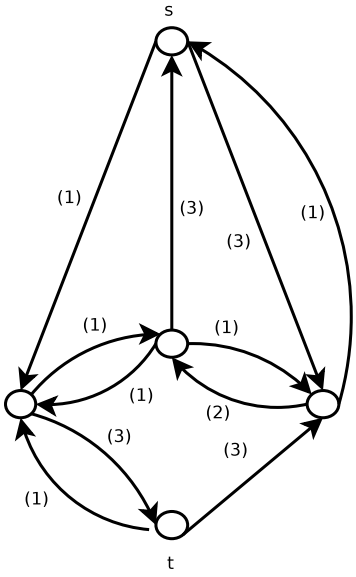

Figure 4.2.4: Residual graph of the flow in the previous figure

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 obvious 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. Find a simple $s-t$ path P in G_f and let c' be the minimum capacity along this path define let f' be the flow of capacity c' along P and 0 everywhere else. Let $f = f + f'$.
4. if no s,t path is found terminate.
5. Repeat

The paths chosen in step 3 are called "augmenting paths".

Theorem 4.2.3 *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 thus 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. It is obvious for the empty flow. Let f_i is our flow at stage i which is feasible by hypothesis and suppose P is our s,t path and $c_{f_i,P}$ is the minimum capacity along that path. Suppose (u,v) is any edge on the path P (obviously for edges outside the path feasibility is no problem).

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 $S = \{v; v \in V \wedge (\text{ exists an } s,v \text{ path in } G_f)\}$ along with it's complement gives us an s,t cut C in G using the second definition. Otherwise we would have an s,t path in G_f which is a contradiction with the fact that the algorithm terminated. Next we wish to claim that the capacity of this cut is exactly equal to the size of our resulting flow f_r.

By the definition of S we must have for any $(u,v) \in C$, $f_r(u,v) = c(u,v)$ since otherwise $c_{f_r}(u,v) > 0$ and thus we can get from s to v via some path. 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 $c_{f_r}(v,u) = c(v,u) + f_r(u,v) > 0$ (since we know that only one of $f_r(u,v)$, $f_r(v,u)$ is non zero). Now this means that we have that the total flow exiting S is the capacity of C. Now

$$\sum_{u \in S}(\sum_{v \in V} f_r(u,v) - \sum_{v \in V} f_r(v,u)) = \sum_{v \in V} f_r(s,v)$$

this follows from the fact that conservation holds for all $u \in S$ other then s, but since $f_r(v,u) = 0$ for $u \in S$ and $v \notin S$ the second sum degenerates into

$$\sum_{v \in S} f_r(v,u)$$

giving us

$$\sum_{u \in S}(\sum_{v \in V} f_r(u,v) - \sum_{v \in S} f_r(v,u))$$

and by regrouping we get $\sum_{u \in S}(\sum_{v \notin S} f_r(u,v))$ which is exactly the total flow out of S. So we see that $|f|$ is actually equal to the capacity of C. ■

Corollary 4.2.4 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.

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.

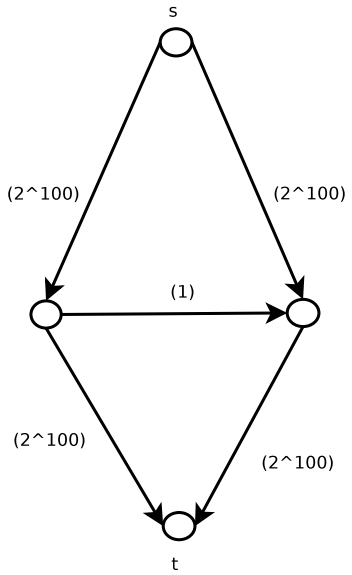

Figure 4.2.5: Bad runtime graph

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 $O(m^2 n)$, but we shall not show that here.