18.433 Combinatorial Optimization

Flow Duality and algorithms

Sept 25, 30 *Lecturer: Santosh Vempala*

1 Introduction

A *directed graph* is a graph in which every edge has a direction. A *capacity* is the maximum flow allowed on an edge, and is represented by $c_{i,j}$, where the edge connects the vertices i and j in the direction from i to j.

Given a directed graph $G = (V, E)$, a *flow* is a collection of paths from a source $s \in V$ to a sink $t \in V$. The set of *edge disjoint paths* is the collection of all paths from s to t such that no edge appears in more than one path.

Suppose S is a set of vertices containing s but not containing t. Then $\bar{S} = V - S$ is the compliment of set S. The *size* of a cut $(S, \bar{S})$ is the number of edges from S to $\bar{S}$.

Theorem 1. *(Menger) A graph G=(V,E) has k edge disjoint paths from s to t $\iff$ k is the size of the minimum directed $s - t$ cut.*

Proof.
($\Rightarrow$) This direction is trivial.

($\Leftarrow$) Assume this is false. Take the smallest counter example G. So G has minimum cut size k but does not contain k edge disjoint paths from s to t. Since G is minimal the removal of any edge will induce a graph with a smaller minimum cut. In particular, G contains no edges into the source s or out of the sink t as these arcs are not present in any $s - t$ cut. We can divide our problem into two cases:

(i) $\exists e \in E$ that is not incident to s or t. Now, by the minimality of G, e is contained in some minimum cut $(S, \bar{S})$. The contraction of the set $\bar{S}$ gives a graph G' with a minimum cut of at least k. Similarly the contraction of the set S gives a graph G'' with a minimum cut of at least k. Since G was the smallest counterexample, G' has k disjoint paths from s to the contracted vertex $\bar{S}$ whilst G'' has k disjoint paths from the contracted vertex S to t. These two sets of k edge disjoint paths only coincide on edges within the cut $(S, \bar{S})$. Hence they may be merged to form k edge disjoint paths from s to t in G. A contradiction.

(ii) Every edge is incident to s or t. Arrange the middle vertices (all vertices except s and t) into the following two groups:

1. All vertices who have more edges going in than out, and s.
2. All vertices who have more edges going out than in, and t.

Observe that, by definition, the set of edges that go from the first group to the second group must contain k edges. It then follows easily that we may find a collection of k edge disjoint paths from s to t. $\square$

2 The Maximum Flow-Minimum Cut Theorem

The capacity of a cut $(S, \bar{S})$, denoted $c(S, \bar{S})$, is equal to the sum of the capacities of each edge in the cut whose direction is from S. Let f be a flow from s to t. We will abuse our notation and also denote by f the value of the flow.

Lemma 2. *Let $(S, \bar{S})$ be any $s - t$ cut in the graph G. Then $f \leq c(S, \bar{S})$.*

Proof. We know that $f(s, V) - f(V, s) = f$. We also know that $f(x, V) - f(V, x) = 0$ for all $x \in V - \{s, t\}$. Using this, we can see that $f(S, V) - f(V, S) = f$. We also know that V is equal to the union of S and $\bar{S}$. Thus we have $f(S, \bar{S}) - f(\bar{S}, S) = f$. In conclusion

$$f = f(S, \bar{S}) - f(\bar{S}, S) \leq f(S, \bar{S}) \leq c(S, \bar{S}) \quad \square$$

In a finite graph there is always a maximum possible flow. Finding this maximum value and the flow that attains it can be a very important part of many graph and network problems. Suppose we have a graph $G = (V, E)$, where $s, t \in V$ are the source and sink, respectively. Take a flow f from s to t. Is it the maximum flow? If we look again at Lemma 2, we can see that the value of the maximum flow is at most the value of the minimum capacity cut. So one way to see if f is maximum is to look for the minimum cut, find it's capacity, and compare the values. A better way is to attempt to find an augmenting path for f. Given our graph, with source s and sink t, an *augmenting path* for f is a path $\{u_0, u_1, \ldots, u_r\}$ where:

1. $u_0 = s$.
2. If (u_i, u_{i+1}) is an edge then $f_{i,i+1} < c_{i,i+1}$.

3. If (u_{i+1}, u_i) is an edge then $f_{i+1,i} > 0$.

We can see that for each vertex in the path, with the exception of s and t, the net flow must equal 0. If $u_r = t$, then our augmenting path is a *flow augmenting path* or *f-augmenting path*, and can be used to increase flow value.

This leads us to an algorithm for finding max flow that is very similar to the one we used to find maximum matching in a graph.

```
ALGORITHM I
{
1) Find an $f$-augmenting path.

2) Augment the original flow.

3) Repeat.
}
```

This algorithm leaves us with a few questions. What is the best way to find an augmenting path? Is this process bounded? Is f maximum when an augmenting path does not exist? We will answer these questions is reverse order.

Theorem 3. *A flow f is maximum $\iff$ there are no flow augmenting paths.*

Proof.
$(\Rightarrow)$ Clearly if there is a flow augmenting path then f can not be a maximum flow.

$(\Leftarrow)$ Take the set of vertices $A_f = \{u \in V : \exists$ an augmenting path from s to $u.\}$. Note that $t \notin A_f$ as we have no flow augmenting path. Consider the cut $(A_f, \bar{A}_f)$. Take an edge (i, j), where $i \in A_f$ and $j \in \bar{A}_f$. Note that for this edge $f_{i,j} = c_{i,j}$ otherwise we could grow A_f. Similarly $f_{j,i} = 0$. This gives us the the flow across the cut is $\sum c_{i,j} - 0$. We can't send anymore than the capacity of the cut, so therefore the flow is maximum if $t \notin A_f$. $\square$

Theorem 4. *(Max Flow-Min Cut) The maximum flow is equal to the minimum capacity cut.*

Proof. Suppose f is the maximum flow value. Therefore the flow f has no augmenting paths. Since it has no augmenting paths, the graph contains a cut, given by A_f, of capacity f. Since no cut can have a capacity less than f the result follows. $\square$.

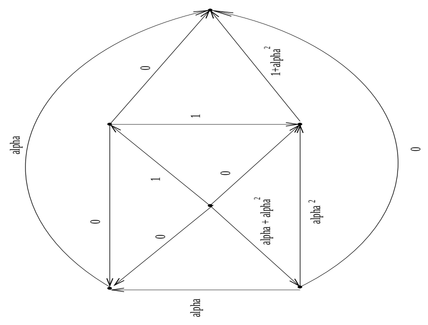

Figure 1: The initial flow.

3 Algorithmic Efficiency

The second question to be answered is whether or not the algorithm for finding an augmenting path is bounded. The answer is, not necessarily. If we look at the Figure 1, finding the wrong set of augmenting paths will lead to an infinite number of augmentation whose augmentation leads to a flow with value less than the maximum flow. The edges have extremely large capacities and the initial flow, of value $1 + \alpha + \alpha^2$ where α is the root of $1 - \alpha - \alpha^3 = 0$, is shown in Figure 1.

To find an augmenting path in this graph, pick a path that starts at s and proceeds to t. Find the lowest current flow value on this path, counting only the edges whose direction goes against the direction of your path. We do not worry about the edges going in the direction of our path, since the capacities of all edges are extremely large. Augment by this value.

For an example, see Figure 2. We may augment this flow by a value α. In the subsequent step the situation is similar except that we may find an augmenting path with capacity α^2. In the next step we may augment along a path of capacity α^3. Observe that we have an infinite process. In addition, the limit of the value of the flow obtained is bounded $(1 + \alpha + \alpha^2 + \sum_{r\geq 1} \alpha^r)$ whereas the maximum flow can be any value.

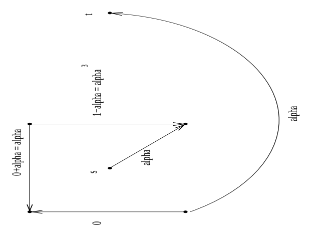

Figure 2: Augmentation by alpha

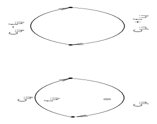

Figure 3: Residual Graph

4 A Weakly Polynomial Algorithm

There are better ways to find an augmenting path than by picking a random flow and trying to augment it. We can try making a residual graph. A *residual graph* takes each edge and makes it two different edges. Take the edge (i, j), with capacity $c_{i,j}$ and flow $f_{i,j}$. The residual graph $\mathrm{Res}(G)$ would have two edges between i and j, each with opposite directions. Edge (i, j) has a capacity on it equal to $c_{i,j} - f_{i,j}$, while the capacity on edge (j, i) is $f_{i,j}$. If either of the capacities is zero the we will omit the edge from $\mathrm{Res}(G)$. If there are already two edges of opposite direction connecting the two vertices, then forming the residual graph becomes a little more complicated (see figure 3).

Claim 1. An augmenting path in G corresponds to a directed s - t path in $\mathrm{Res}(G)$, and therefore and augmenting path in G corresponds to a directed path from s to t in $\mathrm{Res}(G)$. $\square$

This leads us to a new algorithm, similar to the one we were using before.

Algorithm II

```
{
1) Find an $s - t$ path in Res($G$)
2) Augment the flow.
3) Repeat.
}
```

The problem with this, as with before, is that it depends on picking a good original path to augment. How do we make a good choice of path? A good place to start would be to pick the path with the maximum capacity. This is a bounded search, a proof of this is asked for in the homework.

Claim 2. A flow f can be decomposed into at most m paths from s to t, excluding cycles.

Proof of Claim. Every time you create a path, remove the restraining edge in that path from the graph. The restraining edge is that whose flow capacity is filled, and thereby holds you back from sending any more flow down this path. We can easily induce that there can be no more than m paths. $\square$

Theorem 5. *There is a path P with $c(P) \geq \frac{f^*-f}{m}$, where f is the current flow value and f^* is the optimal flow value.*

Proof. Take a graph G, and let f^* be the max flow, and f be the current flow. In Res(G) you should be able to send $f^* - f$ as your max flow. Therefore there exists a path P in Res(G) where $c(P) \geq \frac{f^*-f}{m}$. $\square$.

Now let us examine the running time of the algorithm. We can find a maximum capacity path in $O(m)$ time. Consider the set of augmentations, each with a capacity of at least $\frac{f^*-f}{2m}$. There are at most $2m$ such augmentations. After these augmentations, the remaining flow is at most $\frac{f^*-f}{2}$, since after the augmentations the current maximum capacity path has capacity below $\frac{f^*-f}{2m}$. We can see that the flow remaining halves at most every $2m$ steps, and from this get that the total number of augmentations is at most $2m\log(nU)$. We then have a weakly polynomial running time of $O(m^2\log(nU))$.

5 A Strongly Polynomial Algorithm

Another way to pick which path to start with when looking for an augmenting path is to look for the shortest augmenting path. Start at the source, and look at everything you can get to in one step. That is to say, every vertex that is only one edge away. We will call these depth 1. Now find depths $2, 3, \ldots$ in similar fashion. Find the lowest depth to touch the sink, and send on it as much as you can send. Now take $d(i)$ to be the level of depth of node i. Note that on an edge (i, j) in shortest path from s to t we have $d(j) = d(i) + 1$. The following observation is left as an exercise.

Observation. The shortest path lengths are non-decreasing in the course of this algorithm.

Lemma 6. *The total number of times an edge can be the minimum capacity edge is $O(n)$.*

Proof. Suppose the edge (i, j) with capacity β is the minimum capacity edge along the augmenting path. Augment by β. Now $\mathrm{Res}(G)$ no longer contains the edge (i, j). Before the augmentation, at time τ say, we had $d(j) = d(i) + 1$ since we used a shortest path. Now, the next time this edge is used the edge (i, j) must again be in $\mathrm{Res}(G)$. For this to be the case we must, in the meantime have augmented along some path containing the edge (j, i). At this point, say at time τ', we have $d'(i) = d'(j) + 1$. Hence $d'(i) \geq d(i) + 2$. The maximum value of $d(i)$ is n and the theorem follows. Observe also that the total increase of $d(i)$'s over all vertices is less than n^2, and therefore the total number of augmentations is $O(n^2)$. $\square$

Since it takes $O(m)$ time to find a shortest path the running time of this algorithm is $O(mn^2)$, a strongly polynomial bound.