	6.854 Advanced Algorithms	
Lecture 6: 9/24/2003		Lecturer: Erik Demaine, David Karger
	Scribes: Alexandr Andoni	

Network flows. Maximum flow.

6.1 General definitions.

Definition 1 *Network is a directed graph $G = (V, E)$, in which there is a source vertex $s \in V$, a sink vertex $t \in V$, and on each edge (v, w) there is defined a capacity of the edge, $u(v, w) > 0$. (It is useful also to define the capacity for any pair of verteces (v, w), so that for any $(v, w) \notin E$, $u(v, w) = 0$.)*

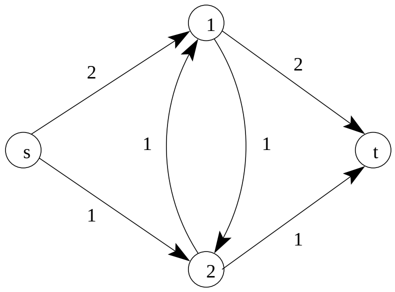

Figure 6.1: An example of a network with 4 vertices and 6 edges. The capacities of the edges are shown on the edges.

There are two ways to define a *flow*: raw flow and net flow.

Definition 2 *Raw flow is a function $r(v, w) \geq 0$ for each edge (v, w). Function r satisfies the following properties:*

- *Conservation:* $\underbrace{\sum_{w \in V} r(w, v)}_{\textit{incoming flow}} - \underbrace{\sum_{w \in V} r(v, w)}_{\textit{outgoing flow}} = 0$, *for all* $v \in V \setminus \{s, t\}$.
- *Capacity:* $r(v, w) \leq u(v, w)$.

Definition 3 *Net flow is a function $f(v, w) : V^2 \to \mathbb{R}$ that satisfies the following conditions:*

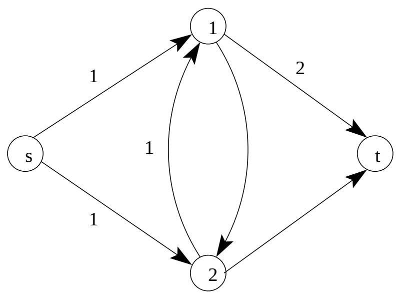

Figure 6.2: An example of a raw flow for the network above. The flow has a value of 2.

- *Skew symmetry:* $f(v,w) = -f(w,v)$.
- *Conservation:* $\sum_{w \in V} f(v,w) = 0$, *for all* $v \in V \setminus \{s,t\}$.
- *Capacity (feasibility condition):* $f(v,w) \le u(v,w)$ *for all* $v, w \in V$.

To ease some notations later in the lecture, we will denote $\sum_{w \in S} f(v,w)$ by $f(v,S)$ or $-f(S,v)$.

Definition 4 *Value of a flow* f *is defined as* $|f| = \sum_{v \in V} f(s,v)$.

Definition 5 *Maximum flow problem: given a network* G*, find the flow with maximum value.*

The maximum flow problem is the problem that we will try to solve in this lecture.

Definition 6 *Residual network* G_f *of a network* G *and flow* f *is a network that contains the same vertices as the network* G*, but the capacities of the edges are* $u'(v,w) = u(v,w) - f(v,w)$*. (From feasibility conditions it results that* $u'(v,w) \ge 0$ *and* $u'(v,w) \le u(v,w) + u(w,v)$*.)* G_f *has edges wherever* $u'(v,w) > 0$.

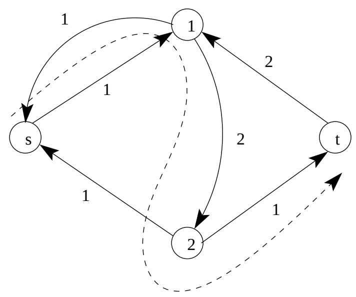

Figure 6.3: An example of a residual network. This residual network corresponds to the network and flow depicted in the previous figures. The dashed line corresponds to a possible augmenting path.

Definition 7 *Augmenting path is a directed path from the node s to node t in the residual network.*

Note that we can push additional unit of flow along such a path in the original network G. The maximum value that we can push on a path $(s, v_1, v_2, \ldots v_k, t)$ is $\min\{u'(s, v_1), u'(v_1, v_2), u'(v_2, v_3), \ldots u'(v_{k-1}, v_k), u'(v_k, t)\}$. If, for a given network G and flow f, there exists an augmenting path in G_f, then the flow f is not maximum. More generally, we can add and subtract a flow in G_f to the flow f (in network G). Thus, if we have a flow f' in G_f, then the flow $f + f'$ is a valid flow in G. Conversely, if we have a flow f' in G, then the flow $f - f'$ is a valid flow G_f.

Definition 8 *A cut is a partition of vertices into 2 groups S and $\bar{S} = V \setminus S$.*

Definition 9 *An s-t cut is a cut, such that $s \in S$ and $t \in \bar{S}$.*

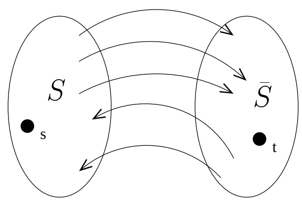

Figure 6.4: An illustration of the s-t cut. $s \in S$ and $t \in \bar{S}$. There might be both edges from S to $\bar{S}$ and from $\bar{S}$ to S.

Definition 10 *Net flow along cut S and $\bar{S}$ is defined as $f(S) = \sum_{v \in S} \sum_{w \in \bar{S}} f(v, w)$.*

Definition 11 *Value (capacity) of a cut is defined as $u(s) = \sum_{v \in S} \sum_{w \in \bar{S}} u(v, w)$.*

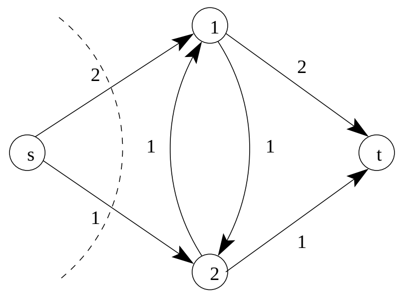

Figure 6.5: An example of a cut in a network. The s-t cut is represented by a dashed line. The value (capacity) of the cut is equal to 3. This is one of the minimum s-t cuts.

Lemma 1 *All s-t cuts carry the same flow, that is for any cut S, $\bar{S}$, it holds that $f(S) = |f| =$ value of flow f.*

Proof: We will prove by induction on the size of the sets S that $f(S) = |f|$. For $S = s$, the claim is true. Now, we move one vertex from $\bar{S}$ to S. Let this vertex be v. The value $f(S)$ changes in the following way:

- $f(S)$ increases by $f(v, \bar{S})$.
- $f(S)$ decreases by $f(S, v) = -f(v, S)$.

Concluding, as a result of moving the vertex v from S to $\bar{S}$, the total change in the value of $f(S)$ is equal to $f(v, \bar{S}) + f(v, S) = f(v, V) = 0$ (by conservation of flow). ■

A conclusion of the above lemma is that $|f| = f(S) \leq u(S)$, and, therefore, any flow, including the maximum flow, has a value equal or less than the capacity of the minimum cut.

Theorem 1 *(Max-flow - min cut theorem). The value of the max flow is equal to the value of the minimum s-t cut:* $\max_f |f| = \min_S u(S)$*, where f is a flow and S is a s-t cut.*

Proof: Consider a max flow f in a network G. Since the flow f is a maximum flow, the residual network G_f has no augmenting paths and, therefore, is disconnected.

Let $S = \{\text{vertices reachable from } s \text{ in } G_f\}$. Since t is not reachable, the set S describes a s-t cut.

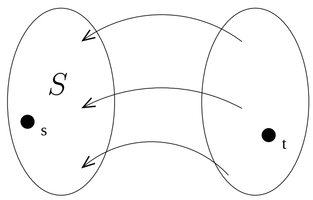

Figure 6.6: Network G_f is disconnected. The set S contains all the nodes that are reachable from s.

Since all edges (v, w) stradding the cut have residual capacity 0, if means that for these edges $f(v, w) = u(v, w)$. Therefore, $|f| = f(S) = u(S)$. In other order of ideas, for a max flow f with value $|f|$, we found a s-t cut that has the capacity $|f|$. With the conclusion from the lemma above, we can conclude that the value of the max flow is equal to the value of the minimum s-t cut. ■

6.2 Ford-Fulkerson Algorithm

Ford-Fulkerson algorithm solves the problem of finding a maximum flow for a given network. The description of the algorithm is as follows:

1. Start with $f(v, w) = 0$.

2. Find an augmenting path from s to t (using, for example, depth first search or similar algorithms).
3. Augment the path found in the previous step.
4. Repeat until there are no augmenting paths.

The running time is $O(m \cdot |f|)$, given that we have integral capacities (the capacities are natural numbers).

In general, if we have integral capacities, then there exists an integral maximal flow. This property is called *integrality property*.

Since the running time is directly proportional to the value of the maximal flow, this particular algorithm is only good for cases when the value $|f|$ is small (for example, when all capacities are at most 1, $|f|$ is at most n). In general, the algorithm may be as bad as linear in unary representation of the input (pseudo polynomial).

If the capacities are rational, then the algorithm will finish, however, it might require more time. If the capacities are real, the algorithm might never finish, converging even to a non-optimal value.

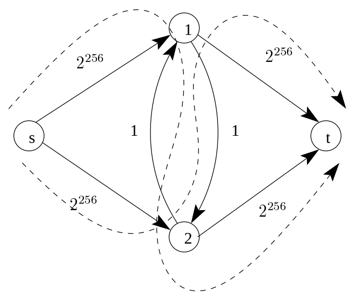

Figure 6.7: An example for which the Ford-Fulkerson, in the stated form, might perform very badly, specifically, if at each step, the augmentation path is either $s-1-2-t$ or $s-2-1-t$ (shown with dashed lines). At an augmentation, the flow will increase with at most 2.

However, if we setup some better rules for selecting the augmentation paths, we might get better results. Before showing some improvements to the Ford-Fulkerson algorithm, we will introduce some new notions on the running time of algorithms.

Definition 12 *An algorithm is psuedo-polynomial if it is polynomial in the unary representation of the input.*

Definition 13 *An algorithm is weakly polynomial if it is polynomial in the binary representation of the input.*

Definition 14 *An algorithm is strongly polynomial if it is polynomial in combinatorial complexity of input. (For example, in the case of max-flow problem, the algorithm would have to be polynomial in n and m.)*

6.2.1 Improvements to the Ford-Fulkerson Algorithm

The are at least two possible ideas for improving the Ford-Fulkerson algorithm. Both of the improvements relay on a better choice of an augmenting path (rather than a random selection of an augmenting path).

1. Using breadth-first search, we can choose shortest-length augmenting path. With this path-selection rule, the number of augmentation is bounded by $n \cdot m$, and thus the running time of the algorithm gets down to $O(nm^2)$ time.
2. We could also choose among all the possible augmenting paths, the augmenting path that can increase the flow the most (max-capacity augmenting path). It is possible to find such a path in $O(m \log n)$ time using a modified Dijkstra's algorithm (ignoring the cycles). The number of augmentations will be at most $m \ln |f| \leq m \ln(nU)$, where $U = \max\{u(v, w)\}$ (for integral capacities).

In this lecture we will prove the time bound for the second improvement. Before this, we will need to show the flow decomposition lemma.

Lemma 2 *(Flow decomposition). We can decompose any flow into at most m cycles and s-t paths.*

Proof: The algorithm for extracting the m paths and cycles is the following.

1. Find a path with positive flow from the node s to node t. (If the flow is non-zero, there exists at least one such path.)
2. Anti-augment the flow on this path, that is reduce the flow in the path until the flow on some edge becomes 0.
3. Add this path as an element of the flow decomposition.
4. Continue these operations until there are no more paths from s to t with positive flow.
5. If there are still some edges with non-zero flow, the remaining flow can be decomposed into cycles. Find a cycle in the following way: take any edge with non-zero flow and follow an outgoing edge with non-zero flow until a cycle is found.
6. Anti-augment on the cycle found.
7. Add the cycle as an element of the flow decomposition.
8. Repeat finding cycles until there are no more edges with non-zero flow.

Since each time we anti-augment a path or a cycle, we zero out the flow on some edge. There are at most m anti-augmentations, and, consequently, paths/cycles in the flow decomposition. ■

Now we can proceed with proving the $m \ln |f|$ bound on the number of augmentation in the second improvement stated earlier (augmenting along a path with max-capacity).

Consider the maximum flow f in the current residual network. We can apply the flow-decomposition lemma (we can discard the cycles since they do not modify $|f|$). There are at most m paths carrying all the flow, and, therefore, there must be at least one path with at least $|f|/m$ flow. Therefore, the augmenting path with maximum capacity increases the flow in the original network by at least $|f|/m$, decreasing the maximum possible flow in the residual graph from $|f|$ to $(1 - 1/m) \cdot |f|$ (remember, the smaller is the maximum possible flow in the residual graph, the greater is the corresponding flow in the original graph). We need to decrease the flow $|f|$ by a factor of $(1-1/m)$ about $m \ln |f|$ times before we decrease the max flow in the residual graph to 1: $|f| \cdot (1-1/m)^{m \ln |f|} \approx |f| \cdot (1/e)^{\ln |f|} \approx 1$. In one more step, the residual graph will have a maximum flow of 0, meaning that the corresponding flow in the original graph is maximal. Thus, we need $O(m \ln |f|)$ augmentations, and, since one augmentation step takes about $O(m \log n)$ time, the total running time is $O(m^2 \cdot \ln |f| \cdot \ln n)$ (this algorithm is weakly polynomial, but not stronlgy polynomial).