CMSC 451: Lecture 15
Network Flows: The Ford-Fulkerson Algorithm
Thursday, Nov 2, 2017

Reading: Sect. 7.2–7.3 in KT.

Network Flow: We continue discussion of the network flow problem. Last time, we introduced basic concepts, such the concepts s-t networks and flows. Today, we discuss the Ford-Fulkerson Max Flow algorithm, cuts, and the relationship between flows and cuts.

Recall that a *flow network* is a directed graph $G = (V, E)$ in which each edge $(u, v) \in E$ has a nonegative *capacity* $c(u, v) \geq 0$, with a designated source vertex s and sink vertex t. We assume that there are no edges entering s or exiting t. A *flow* is a function f that maps each edge to a nonnegative real number that does not exceed the edge's capacity, and such that the total flow into any vertex other than s and t equals the total flow out of this vertex. The total *value* of a flow is equal to the sum of flows coming out of s (which, by flow conservation, is equal to the total flow entering t). The objective of the *max flow problem* is to compute a flow of maximum value. Today we present an algorithm for this problem.

Why Greedy Fails: Before considering our algorithm, we start by considering why a simple greedy scheme for computing the maximum flow does not work. The idea behind the greedy algorithm is motivated by the path-based notion of flow. (Recall this from the previous lecture.) Initially the flow on each edge is set to zero. Next, find any path P from s to t, such that the edge capacities on this path are all strictly positive. Let $c_{\min}$ be the minimum capacity of any edge on this path. This quantity is called the *bottleneck capacity* of the path. Push $c_{\min}$ units through this path. For each edge $(u, v) \in P$, set $f(u, v) \leftarrow c_{\min} + f(u, v)$, and decrease the capacity of (u, v) by $c_{\min}$. Repeat this until no s-t path (of positive capacity edges) remains in the network.

While this may seem to be a very reasonable algorithm, and will generally produce a valid flow, it may fail to compute the maximum flow. To see why, consider the network shown in Fig. 1(a). Suppose we push 5 units along the topmost path, 8 units along the middle path, and 5 units along the bottommost path. We have a flow of value 18. After adjusting the capacities (see Fig. 1(b)) we see that there is no path of positive capacity from s to t. Thus, greedy gets stuck.

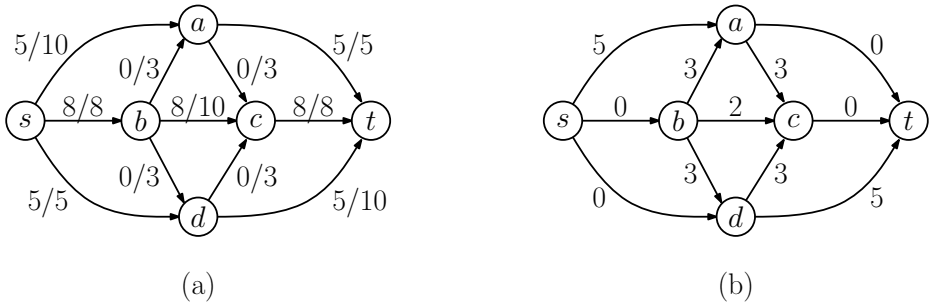


Fig. 1: The greedy flow algorithm can get stuck before finding the maximum flow.

Residual Network: The key insight to overcoming the problem with the greedy algorithm is to observe that, in addition to increasing flows on edges, it is possible to *decrease* flows on edges that already carry flow (as long as the flow never becomes negative). It may seem counterintuitive that this would help, but we shall see that it is exactly what is needed to obtain an optimal solution.

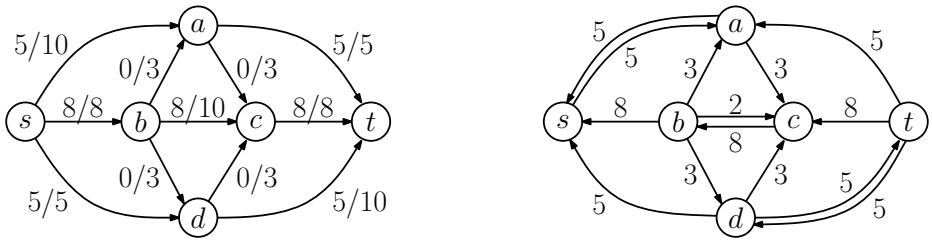
To make this idea clearer, we first need to define the notion of the residual network and augmenting paths. Given a flow network G and a flow f, define the *residual network*, denoted G_f, to be a network having the same vertex set and same source and sink, and whose edges are defined as follows:

Forward edges: For each edge (u, v) for which $f(u, v) < c(u, v)$, create an edge (u, v) in G_f and assign it the capacity $c_f(u, v) = c(u, v) - f(u, v)$. Intuitively, this edge signifies that we can *increase* the flow along this edge by up to $c_f(u, v)$ units without violating the original capacity constraint.

Backward edges: For each edge (u, v) for which $f(u, v) > 0$, create an edge (v, u) in G_f and assign it a capacity of $c_f(v, u) = f(u, v)$. Intuitively, this edge signifies that we can *decrease* the existing flow by as much as $c_f(u, v)$ units. Conceptually, by pushing positive flow along the reverse edge (v, u) we are decreasing the flow along the original edge (u, v).

Observe that every edge of the residual network has *strictly positive* capacity. (This will be important later on.) Note that each edge in the original network may result in the generation of up to two new edges in the residual network. Thus, the residual network is of the same asymptotic size as the original network.

An example of a flow and the associated residual network are shown in Fig. 2(a) and (b), respectively. For example, the edge (b, c) of capacity 2 signifies that we can add up to 2 more units of flow to edge (b, c) and the edge (c, b) of capacity 8 signifies that we can cancel up to 8 units of flow from the edge (b, c).



(a): A flow f in network G (b): Residual network G_f

Fig. 2: A flow f and the residual network G_f.

The capacity of each edge in the residual network is called its *residual capacity*. The key observation about the residual network is that if we can push flow through the residual network then we can push this additional amount of flow through the original network. This is formalized in the following lemma. Given two flows f and f', we define their *sum*, $f + f'$,

in the natural way, by summing the flows along each edge. If $f'' = f + f'$, then $f''(u, v) = f(u, v) + f'(u, v)$. Clearly, the value of $f + f'$ is equal to $|f| + |f'|$.

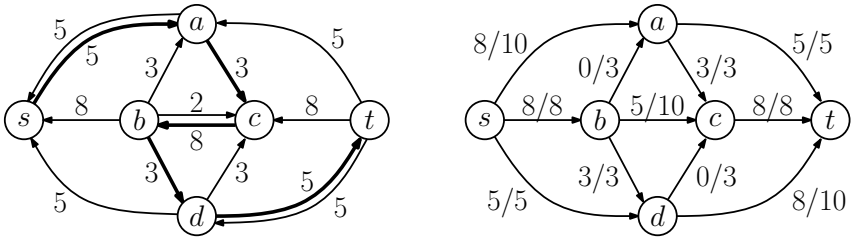
Lemma: Let f be a flow in G and let f' be a flow in G_f. Then $(f + f')$ is a flow in G.

Proof: (Sketch) To show that the resulting flow is valid, we need to show that it satisfies both the capacity constraints and flow conservation. It is easy to see that the capacities of G_f were exactly designed so that any flow along an edge of G_f when added to the flow f of G will satisfy G's capacity constraints. Also, since both flows satisfy flow conservation, it is easy to see that their sum will as well. (More generally, any linear combination $\alpha f + \beta f'$ will satisfy flow conservation.)

This lemma suggests that all we need to do to increase the flow is to find any flow in the residual network. This leads to the notion of an augmenting path.

Augmenting Paths and Ford-Fulkerson: Consider a network G, let f be a flow in G, and let G_f be the associated residual network. An *augmenting path* is a simple path P from s to t in G_f. The *residual capacity* (also called the *bottleneck capacity*) of the path is the minimum capacity of any edge on the path. It is denoted $c_f(P)$. (Recall that all the edges of G_f are of strictly positive capacity, so $c_f(P) > 0$.) By pushing $c_f(P)$ units of flow along each edge of the path, we obtain a valid flow in G_f, and by the previous lemma, adding this to f results in a valid flow in G of strictly higher value.

For example, in Fig. 3(a) we show an augmenting path of capacity 3 in the residual network for the flow given earlier in Fig. 2. In (b), we show the result of adding this flow to every edge of the augmenting path. Observe that because of the backwards edge (c, b), we have decreased the flow along edge (b, c) by 3, from 8 to 5.



(a): Augmenting path of capacity 3 (b): The flow after augmentation

Fig. 3: Augmenting path and augmentation.

How is this different from the greedy algorithm? The greedy algorithm only increases flow on edges. Since an augmenting path may increase flow on a backwards edge, it may actually *decrease* the flow on some edge of the original network.

This observation naturally suggests an algorithm for computing flows of ever larger value. Start with a flow of weight 0, and then repeatedly find an augmenting path. Repeat this until no such path exists. This, in a nutshell, is the simplest and best known algorithm for computing flows, called the *Ford-Fulkerson method*. (We do not call it an "algorithm,"

Ford-Fulkerson Network Flow

```
ford-fulkerson-flow(G = (V, E, s, t)) {
    f = 0 (all edges carry zero flow)
    while (true) {
        G' = the residual-network of G for f
        if (G' has no s-t augmenting path)
            break                                   // no augmentations left
        P = any-augmenting-path of G'               // augmenting path
        c = minimum capacity edge of P              // augmentation amount
        augment f by adding c to the flow on every edge of P
    }
    return f
}
```

since the method of selecting the augmenting path is not specified. We will discuss various strategies in future lectures.) It is summarized in the code fragment below.

There are three issues to consider before declaring this a reasonable algorithm.

- How efficiently can we perform augmentation?
- How many augmentations might be required until converging?
- If no more augmentations can be performed, have we found the max-flow?

Let us consider first the question of how to perform augmentation. First, given G and f, we need to compute the residual network, G_f. This is easy to do in $O(n+m)$ time, where $n = |V|$ and $m = |E|$. We assume that G_f contains only edges of strictly positive capacity. Next, we need to determine whether there exists an augmenting path from s to t G_f. We can do this by performing either a DFS or BFS in the residual network starting at s and terminating as soon (if ever) t is reached. Let P be the resulting path. Clearly, this can be done in $O(n+m)$ time as well. Finally, we compute the minimum cost edge along P, and increase the flow f by this amount for every edge of P.

Two questions remain: What is the best way to select the augmenting path, and is this correct in the sense of converging to the maximum flow? Next, we consider the issue of correctness. Before doing this, we will need to introduce the concept of a cut.

Cuts: In order to show that Ford-Fulkerson leads to the maximum flow, we need to formalize the notion of a "bottleneck" in the network. Intuitively, the flow cannot be increased forever, because there is some subset of edges, whose capacities eventually become saturated with flow. Every path from s to t must cross one of these saturated edges, and so the sum of capacities of these edges imposes an upper bound on size of the maximum flow. Thus, these edges form a bottleneck.

We want to make this concept mathematically formal. Since such a set of edges lie on every path from s from t, their removal defines a partition separating the vertices that s can reach from the vertices that s cannot reach. This suggests the following concept.

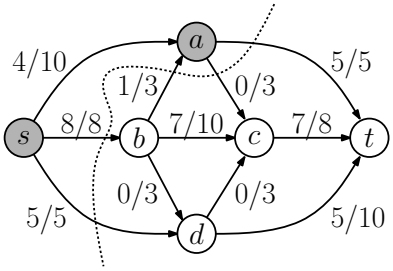
Given a network G, define a *cut* (also called an *s-t cut*) to be a partition of the vertex set into two disjoint subsets $X \subseteq V$ and $Y = V \setminus X$, where $s \in X$ and $t \in Y$. We define the *net*

flow from X to Y to be the sum of flows from X to Y minus the sum of flows from Y to X, that is,

$$f(X,Y) \;=\; \sum_{x\in X}\sum_{y\in Y} f(x,y) - \sum_{y\in Y}\sum_{x\in X} f(y,x).$$

Observe that $f(X,Y) = -f(Y,X)$.

For example, Fig. 4 shows a flow of value 17. It also shows a cut $(X,Y) = (\{s,a\},\{b,c,d,t\})$, where $f(X,Y) = 17$.



$X = \{s,a\}$

$Y = \{b,c,d,t\}$

$f(X,Y) = 5 + 0 - 1 + 8 + 5 = 17$

Fig. 4: Flow of value 17 across the cut $(\{s,a\},\{b,c,d,t\})$.

Lemma: Let (X,Y) be any s-t cut in a network. Given any flow f, the value of f is equal to the net flow across the cut, that is, $f(X,Y) = |f|$.

Proof: Recall that there are no edges leading into s, and so we have $|f| = f^{\text{out}}(s) = f^{\text{out}}(s) - f^{\text{in}}(s)$. Since all the other nodes of X must satisfy flow conservation it follows that

$$|f| \;=\; \sum_{x\in X}(f^{\text{out}}(x) - f^{\text{in}}(x))$$

Now, observe that every edge (u,v) where both u and v are in X contributes one positive term and one negative term of value $f(u,v)$ to the above sum, and so all of these cancel out. The only terms that remain are the edges that either go from X to Y (which contribute positively) and those from Y to X (which contribute negatively). Thus, it follows that the value of the sum is exactly $f(X,Y)$, and therefore $|f| = f(X,Y)$.

Define the *capacity* of the cut (X,Y) to be the sum of the capacities of the edges leading from X to Y, that is,

$$c(X,Y) \;=\; \sum_{x\in X}\sum_{y\in Y} c(x,y).$$

There is a noteworthy asymmetry between the net flow of a cut and the capacity of a cut. In the net flow, we consider both the flows from X to Y and (negated) from Y to X. In contrast, in the definition of cut capacity, we only consider capacities from X to Y. To understand why this asymmetry exists, suppose that there there are two edges (x,y) and (y,x), where $x \in X$ and $y \in Y$, where the edges have the same capacities and both carry the same flow. The flow from y to x effectively cancels out the flow from x to y (as if you have an "eddy" in the middle of a river, where the flow cycles around). Thus, this cycle does not contribute to the total flow value, and so it is important to consider both in the net flow. On the other hand,

the capacity of the edge from y to x does not contribute to nor detract from the maximum amount of flow we can push from the X-side to the Y-side of the cut. Therefore, we do not need to consider it in computing the cut's total capacity.

It is easy to see that it is not possible to push more flow through a cut than its capacity. Combining this with the above lemma we have:

Lemma: Given any s-t cut (X, Y) and any flow f we have $|f| \leq c(X, Y)$.

The optimality of the Ford-Fulkerson method is based on the following famous theorem, called the *Max-Flow/Min-Cut Theorem*. Basically, it states that in any flow network the minimum capacity cut acts like a bottleneck to limit the maximum amount of flow. The Ford-Fulkerson method terminates when it finds this bottleneck, and hence on termination, it finds both the minimum cut and the maximum flow.

Max-Flow/Min-Cut Theorem: The following three conditions are equivalent.

(i) f is a maximum flow in G,
(ii) The residual network G_f contains no augmenting paths,
(iii) $|f| = c(X, Y)$ for some cut (X, Y) of G.

Proof:

- (i) $\Rightarrow$ (ii): (by contradiction) If f is a max flow and there were an augmenting path in G_f, then by pushing flow along this path we would have a larger flow, a contradiction.
- (ii) $\Rightarrow$ (iii): If there are no augmenting paths then s and t are not connected in the residual network. Let X be those vertices reachable from s in the residual network, and let Y be the rest. Clearly, (X, Y) forms a cut. Clearly, each edge crossing the cut from X to Y must be saturated (or else we could reach one more vertex on the Y-side by adding flow along this edge), and each edge crossing the cut from Y to X must be carrying zero flow (or else we could reach one more vertex on the Y-side by reducing flow along this edge). It follows that the flow across the cut equals the capacity of the cut, thus $|f| = c(X, Y)$.
- (iii) $\Rightarrow$ (i): This follows directly from the previous lemma. No flow value can exceed any cut capacity, and so if *any* flow's value matches *any* cut's capacity, this flow must be maximum.

We have established that, on termination, Ford-Fulkerson generates the maximum flow. But, is it guaranteed to terminate and, if so, how long does it take? We will consider this question next time.