CSE 711 (Fall 2018): Topics in Combinatorial Optimization and Linear Programming

Lecture 1 (8/30/2018): Network Flow

Lecturer: Prof. Shi Li *Scribe: Shi Li*

1 Network Flow Problem

In a network flow problem, we are given a flow network (G, c, s, t), where $G = (V, E)$ is a directed graph and $s, t \in V$ are called the source and the sink of the network. We assume that s does not have incoming edges and t does not have outgoing edges. $c : E \to \mathbb{R}_{>0}$ is a capacity function. Figure 1 gives an example of a flow network.

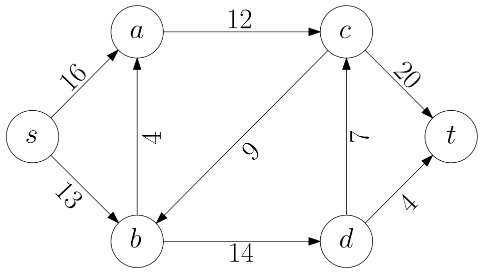

Figure 1: A flow network.

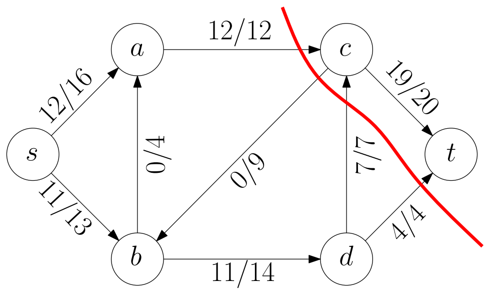

Figure 2: A maximum flow f for the network, with $\text{val}(f) = 23$. The first number on each edge e represents the flow value $f(e)$ and the second number represents the capacity $c(e)$. The red curved line gives a cut of value 23.

Definition 1. *Given a flow network $(G = (V, E), c, s, t)$, an s-t flow is a function $f : E \to \mathbb{R}$ such that*

- *for every $e \in E$: $0 \le f(e) \le c(e)$* *(capacity conditions)*
- *for every $v \in V \setminus \{s, t\}$:*

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e). \qquad \text{(conservation conditions)}$$

The value *of an s-t flow f is defined as*

$$\text{val}(f) = \sum_{e \text{ out of } s} f(e).$$

The maximum flow problem asks for a flow f with the maximum value, given the flow network (G, c, s, t). For example, the maximum flow value for the network in Figure 1 is 23, given by the flow in Figure 2.

2 First Attempt : Greedy Algorithm

We first design a simple greedy strategy for the network flow problem. The idea is that as long as we can find a path of "unsaturated" edges from s to t, we increase the flow values along the path as much as possible. Repeat the process until no such path can be found. Formally, given a flow f, for every $e \in E$, we define $c_f(e) = c(e) - f(e)$ to be the *residual* capacity of an edge e. We run Algorithm 1

Algorithm 1 Simple Greedy Algorithm for the Network Problem

```
1: $f(e) \leftarrow 0$ for every $e \in E$
2: **while** there exists a path $P$ from $s$ to $t$ in $G$ containing edges with positive residual capacities **do**
3:     $\delta \leftarrow \min_{e \in P} c_f(e)$
4:     let $f(e) \leftarrow f(e) + \delta$ for every $e \in E$
5: **return** $f$ and val($f$).
```

However, the algorithm does not always return a flow with the maximum value. Consider the flow network in Figure 3 and the flow f. The maximum flow value is 2, achieved by sending 1 unit flow along $s \to a \to t$ and 1 unit flow along $s \to b \to t$. However, if we first find the path $s \to a \to b \to t$ and send 1 unit flow along the path, then we can not send the second unit flow using the strategy in Algorithm 1.

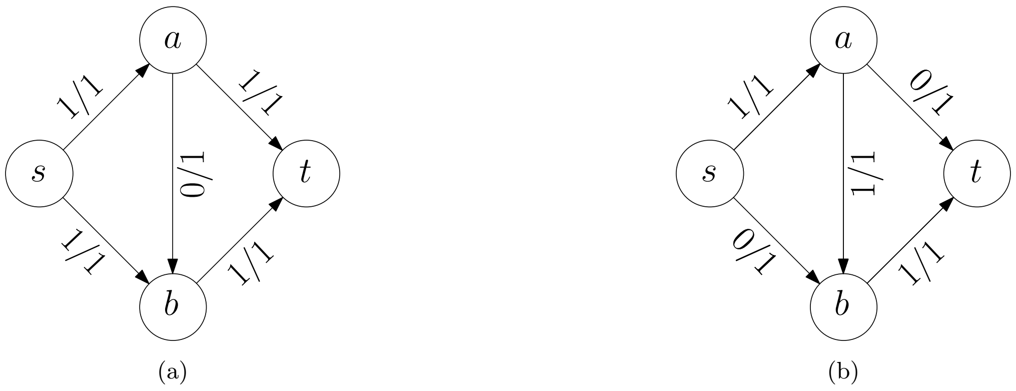

Figure 3: An example showing that Algorithm 1 is not optimal. Figure (a) shows the maximum flow of value 2 for the network, while Figure (b) shows the flow returned by Algorithm 1.

3 Fixing the issue by allowing sending flow backwards

We now consider how to fix the issue that Algorithm 1 encountered. Focus on the flow in Figure 3(b). We can indeed increase the flow value by sending an extra unit flow along the path $s \to b \to a \to t$. There is no edge (b, a) in the input flow network G. However, since the flow value from a to b is 1, we can think of sending 1 unit flow from b to a as reducing the flow sent from a to b by 1. Thus, after the update, the flow values on (s, b) and (a, t) are increased to 1, and the flow value on (a, b) is reduced to 0. This gives the maximum flow value of 2 as depicted in Figure 3(a).

The above example motivates the definition of *residual graphs*: in the residual graph for the above example, we have an edge of capacity 1 from b to a, since we are allowed to decrease the flow sent from a to b by 1. For notational convenience, let us assume that in the original network $G = (V, E)$, two anti-parallel edges (u, v) and (v, u) can not be both in E.[1] Then given a valid flow f for the flow network, we define the residual graph $G_f = (V, E_f)$ and the residual capacities c_f as follows:

- If $(u, v) \in E$ and $f((u, v)) < c((u, v))$, then we have an edge $(u, v) \in E_f$ with residual capacity $c_f((u, v)) = c((u, v)) - f((u, v))$. In this case, we call (u, v) a *forward edge* in G_f.
- If $(v, u) \in E$ and $f((v, u)) > 0$, then we have an edge $(u, v) \in E_f$ with residual capacity $c_f((u, v)) = f((v, u))$. In this case, we call (u, v) a *backward edge* in G_f.
- There are no other edges in E_f than those defined by the two rules above.

[1]This can be assumed w.l.o.g: if this happens, we can split the edge (v, u) in the middle into two edges (v, w) and (w, u).

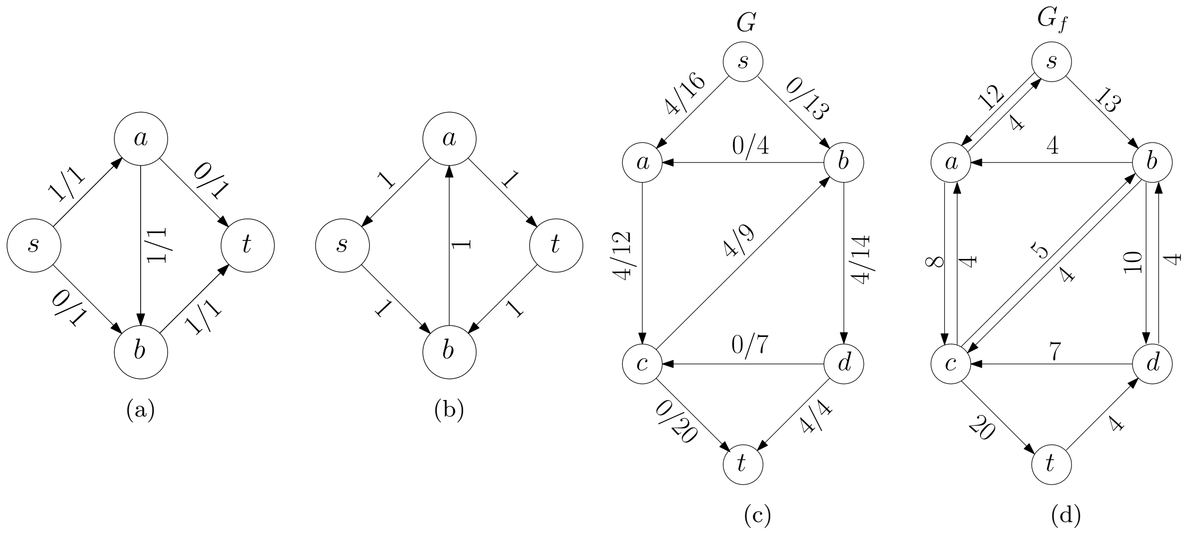

Figure 4: Examples of residual graphs. Figures 4(b) and 4(d) are respectively the residual graphs for the networks and the flows f given in Figures 4(a) and 4(c).

See Figure 4 for examples of residual graphs.

With the residual graph defined, we can now give our updated algorithm. Indeed, the algorithm is called *Ford-Fulkerson Method*, which works as follows. We start from the 0-flow f. As long as there is a path P from s to t in the residual graph G_f, we update the flow f by calling augment(P). In augment(P), we let δ be the minimum residual capacity of edges in E; this will be the increase of val(f). Then for every forward edge $(u, v) \in P$, we increase the f value of (u, v) by δ; for every backward edge $(u, v) \in P$, we decrease the f value of (v, u) by δ.

Algorithm 2 Ford-Fulkerson Method

```
1: $f(e) \leftarrow 0$ for every $e \in E$
2: while there exists a path $P$ from $s$ to $t$ in $G_f$ do
3:     augment($P$)
4: return $f$ and val($f$).
```

Algorithm 3 augment(P)

```
1: $\delta \leftarrow \min_{e \in P} c_f(e)$
2: for every $(u, v) \in P$ do
3:     if $(u, v)$ is a forward edge then
4:         $f((u, v)) \leftarrow f((u, v)) + \delta$
5:     else            \\ $(u, v)$ is a backward edge
6:         $f((v, u)) \leftarrow f((v, u)) - \delta$
```

3.1 Analysis of Ford-Fulkerson Method

The next 3 lemmas establish the correctness of the Ford-Fulkerson method.

Lemma 2. *When the Ford-Fulkerson method (Algorithm 2) terminates, the returned flow f is a valid flow.*

We only give a sketch of the proof. Initially, f is all 0 vector and the two conditions in Definition 1 are trivially satisfied. We show that the two conditions are maintained every time we call augument(P). The capacity constraint is maintained due to the way we define δ. To see that the flow conservation conditions are maintained, we consider an intermediate vertex v in the path P and let (u, v) and (v, w) be the incoming and outgoing edges of v in P. We have 4 cases depending on whether each of (u, v) and (v, w) is a forward edge or a backward edge. Using a case-by-case analysis, the conservation condition for v is maintained.

The crux of the analysis is in showing that the returned flow f gives the largest value.

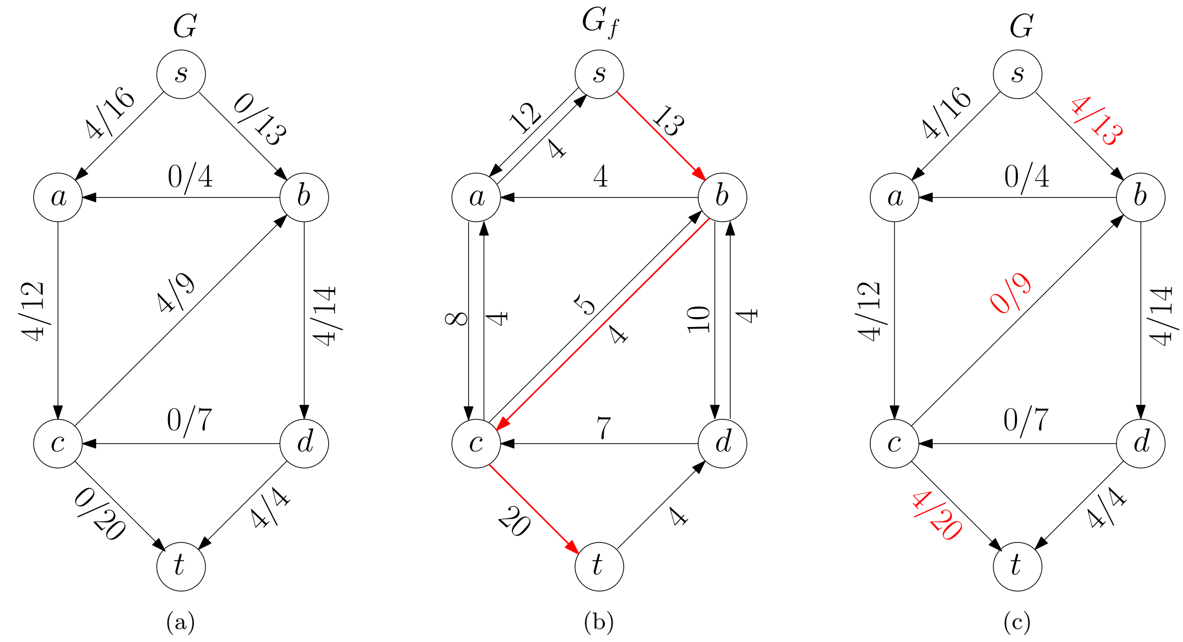

Figure 5: Augmenting the flow f along a path P. Figure 5(a) gives the initial flow f, Figure 5(b) gives the residual graph for this f and the path P on which we are going to call augment, and Figure 5 gives the flow f after the procedure augment(P).

Lemma 3. *When the Ford-Fulkerson method terminates, the returned flow f is optimum. I.e, it gives the maximum flow value for the problem.*

To prove the lemma, we define s-t cuts and their values. Then we show that the flow value of the f returned by the Ford-Fulkerson method is equal to the value of some s-t, which will establish the optimality of f. This analysis also proves the classic maximum flow minimum cut theorem.

Definition 4. *An s-t* cut *of $G = (V, E)$ is a pair $(S \subseteq V, T = V \setminus S)$ such that $s \in S$ and $t \in T$. The* cut value *of an s-t cut (S, T) is defined as*

$$\text{cut-value}(S, T) := \sum_{e=(u,v)\in E: u\in S, v\in T} c(e).$$

The following observation says that the value of any flow f is the amount of flow sent from S to T minus the amount of flow sent from T back to S, for any s-t cut (S, T).

Observation 5. *For every s-t flow f and s-t cut (S, T), we have*

$$\text{val}(f) = \sum_{(u,v)\in E: u\in S, v\in T} f((u, v)) - \sum_{(u,v)\in E: u\in T, v\in S} f((u, v)).$$

Proof. For every $v \in S\setminus\{s\}$, we have (a) $\sum_{e\in E: e \text{ into } v} f(e) - \sum_{e\in E: e \text{ out of } v} f(e) = 0$ by the flow conservation conditions. We also have (b) $\text{val}(f) - \sum_{e\in E: e \text{ out of } s} = 0$. We then sum up the equalities (a) over all $v \in S \setminus \{s\}$ and b. The left-side of the summation has the term $\text{val}(f)$. We consider how each $f(e)$ appears in the summation. Notice that we assumed that s does not have incoming edges.

- For every edge $e = (u, v) \in E$ with $u, v \in S$, the $+f(e)$ and $-f(e)$ terms will cancel each other.
- For an edge $e = (u, v)$ with $u \in S, v \in T$, $f(e)$ has a $-$ sign;

- For an edge $e=(v,u)$ with $u \in S, v \in T$, $f(e)$ has a $+$ sign.
- The other edges do not appear in the summation.

Thus, we have $\mathrm{val}(f) - \sum_{e=(u,v)\in E: u\in S, v\in T} f(e) + \sum_{e=(v,u)\in E: u\in S, v\in T} f(e) = 0$. Moving the two summation terms to the right side finishes the proof of the observation. $\square$

The observation implies

Corollary 6. *For every s-t flow f and every s-t cut (S,T), we have* $\mathrm{val}(f) \le \text{cut-value}(S,T)$.

Now consider the flow f returned by the Ford-Fulkerson Method. When the algorithm terminates, there is no path from s to t in G_f. This means that there is a s-t cut (S,T) such that there are no edges in G_f that go from S to T. This means

- For every edge $(u,v) \in E$ such that $u \in S$ and $v \in T$, we have $f((u,v)) = c((u,v))$. This must hold since otherwise we would have $(u,v) \in E_f$.
- For every edge $(v,u) \in E$ such that $v \in T$ and $u \in S$, we have $f((v,u)) = 0$. This must hold since otherwise we would have $(u,v) \in E_f$, contradicting that there are no edges going from S to T in G_f.

Thus, by Observation 5, we have

$$\mathrm{val}(f) = \sum_{(u,v)\in E: u\in S, v\in T} f((u,v)) - \sum_{(v,u)\in E: u\in S, v\in T} f((v,u)) = \sum_{(u,v)\in E: u\in S, v\in T} c((u,v)) = \text{cut-value}(S,T).$$

Since the value of every s-t flow f' has value at most cut-value(S,T), we proved that f gives the maximum value.

The above proof indeed gives the following *maximum flow minimum cut theorem.*

Theorem 7. $\sup_{s\text{-}t \text{ flow } f} \mathrm{val}(f) = \min_{s\text{-}t \text{ cut } (S,T)} \text{cut-value}(S,T).$

The theorem says that the maximum value of any s-t flow f equals the minimum value of any s-t cut. Corollary 6 already implies that LHS$\le$RHS. The Ford-Fulkerson method returns a flow f and a cut (S,T) such that $\mathrm{val}(f) = \text{cut-value}(S,T)$, implying the inequality holds with equality.

Finally, we briefly discuss the running time of the Ford-Fulkerson method. We only focus on the case when all the capacities are integers. The capacities might be very large compared to the size of the network. The running time of the Ford-Fulkerson method depends on how we choose the path from s to t in G_f.[2] It turns out if the path from s to t was chosen by an adversary, then the running time of the algorithm might be exponential. There are two strategies of choosing paths that will lead to a polynomial time algorithm: choosing the path from s to t in G_f with the smallest number of edges, and choosing the path with the largest bottleneck capacity (i.e, the one that will maximize δ in augment(P)).

[2]This is why sometimes we use Ford-Fulkerson method instead of Ford-Fulkerson algorithm: It does not define a way to choose the path P.