MAT344 Lecture 24

2019/Aug/8

1 Announcements

1. Please complete the course evaluation.

2 This week

This week, we are talking about

1. The max flow/min cut theorem
2. The Ford-Fulkerson algorithm

3 Recap

Last time we talked about

1. Cuts and flows

4 The Max flow/Min cut theorem

Recall that we are trying to prove

Theorem 4.1 (Ford-Fulkerson, Theorem 13.8 in [KT17])**.** *Let $G = (V, E)$ be a network. Then let v_0 be the maximum value of a flow, and let c_0 be the minimum capacity of a cut. Then*

$$v_0 = c_0.$$

We are trying to find paths through our network where we can increase the flow.

Definition 4.2. *Let f be a flow in our network. We will say that the sequence $P = x_0, x_1, \ldots, x_k$ is a **special path**, or **augmenting path** from x_0 to x_k if for each i, $1 \leq i \leq k$, either*

1. *$e = (x_{i-1}, x_i)$ is an edge with $c(e) - f(e) > 0$, or*
2. *$e = (x_i, x_{i-1})$ is an edge with $f(e) > 0$.*

Consider the network on Figure 1

Edges e with $f(e) = c(e)$ are said to be *saturated* and the two conditions above can be stated in words as requiring the 'forward' edges of the path to be unsaturated while requiring 'backward' edges to have positive flow.

For example, on the flow in Figure 2, the path in red is an augmenting path.

Notice that we can increase the flow strength if, for all 'forward' edges in the augmenting path, we increase the flow value by 1, and for the 'backward' edge, we decrease the flow value by 1 to get the flow on Figure 3

For an augmenting path, we define two quantities

$$\delta_1 = \min\{c(x_{i-1}, x_i) - f(x_{i-1}, x_i) | (x_{i-1}, x_i) \text{ is a 'forward' edge of } P\}$$
$$\delta_2 = \min\{f(x_i, x_{i-1}) | (x_{i-1}, x_i) \text{ is a 'forward' edge of } P\}$$

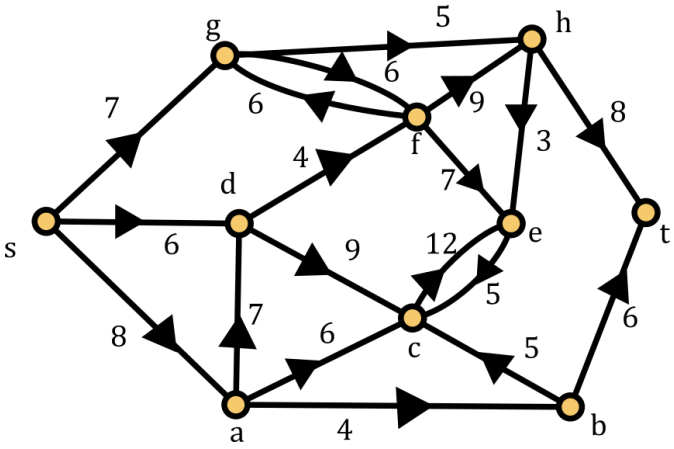

Figure 1: A network

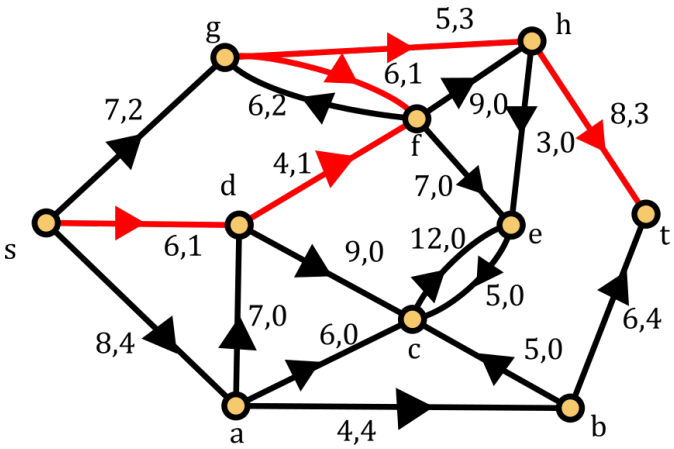

Figure 2: A flow and an augmenting path

Then we let $\delta = \min\{\delta_1, \delta_2\}$. What does this δ represent? This is the amount of flow that can be 'pushed through' the augmenting path. We do the same thing as in the example, we increase the flow by δ at the 'forward' edges and decrease at the 'backward' ones. This new assignment is then also a flow.

Suppose there exists an augmenting path P from s to t. Then we can increase the total flow value by δ.

What if for a flow, there is no augmenting path from s to t? We will produce a cut in our network the following way: Let X be the set of vertices that can be reached from s using an augmenting path, and let Y be the set of remaining vertices. Clearly $s \in X$, $t \in Y$ and $X \cup Y = V$, so this is indeed a cut.

Notice that all edges $e = (x, y)$ with $x \in X, y \in Y$ must be saturated (at full capacity) because otherwise we could reach y from s using an augmenting path. Also, all edges $e(y, x)$ with $y \in Y, x \in X$ must have $f(e) = 0$ (same argument). Therefore we have (using the fact that the capacity of any cut is an upper bound for any flow)

$$|f| = f(X, Y) - f(Y, X) = c(X, Y).$$

Notice what this equality implies. Since the capacity of any cut is an upper bound for the strength of any flow, and we have a flow whose strength is equal to a capacity of a cut, the cut we constructed must be minimal, and the flow we had must have been maximal. This proves Theorem 4.1.

5 The Ford-Fulkerson algorithm

How do we find augmenting paths? The Ford-Fulkerson algorithm is a *breadth-first* search. To stay consistent, we will label the vertices as $\{s, t, a, b, c, d, \ldots\}$ (with s the source and t the sink). Our book calls this pseudo-alphabetic order. We will label the vertices and scan from them (similarly to 'visiting' and scanning in Dijkstra's

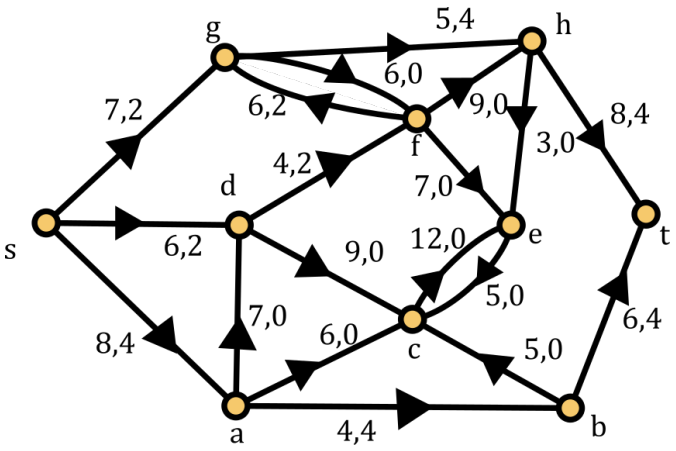

Figure 3: Augmenting the flow

algorithm). The labels on a vertex v will be triples $(u, \pm, p(v))$ where the first entry u indicates a previous vertex in an augmenting path, the second entry is a $+$ if the edge from u to v is forward and $-$ if it is backward, and $p(v)$ is a number that is the **potential** – the value of flow that can be transmitted from u to v using this augmenting path. We are trying to label the sink, because this will mean that we have found an augmenting path.

Given a network and a flow f:

1. We first label the source with the triple $(*, +, \infty)$.
2. We scan from the next labeled vertex u that has positive potential. We look at all vertices that are yet unlabeled that are adjacent to u (by either a forward or backward edge). We will label the (not yet labeled) vertices in the pseudo-alphabetic order.
3. The label of the vertex v that is next in order as follows:
   (a) If we are following a forward edge (u, v) that is not saturated then we will label the vertex v as $(u, +, \min\{p(u), c(u, v) - f(u, v)\})$.
   (b) If we are following a backward edge (v, u) that has positive flow then we will label the vertex v as $(u, -, \min\{p(u), f(u, v)\}$.
   (c) Otherwise we don't label v and proceed to the next vertex.
   (d) The algorithm halts if the sink is labeled or if no further labeling is possible.

Consider the small network on Figure 4. Let's see how the algorithm produces a maximal flow.

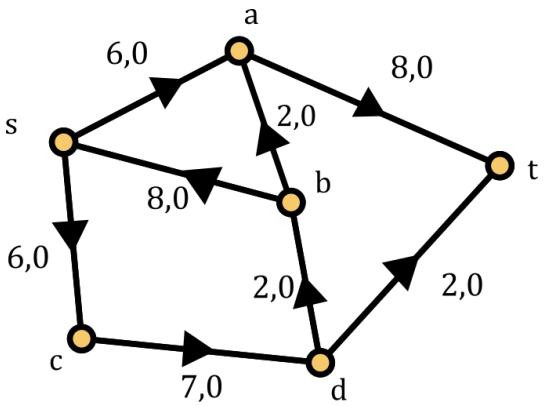

Figure 4: The zero flow

1. We label the source with $(*, +, \infty)$. We scan, finding vertices a, b, c.
   (a) The label on a is $(s, +, 6)$
   (b) b does not get labeled, as there is no flow from b to s
   (c) The label on c is $(s, +, 6)$
2. We scan from a, finding t.
   (a) We label t with $(a, +, 6)$ and the algorithm halts.

We have found the augmenting path that we can use to push through 6 units of flow. The result is on Figure 5

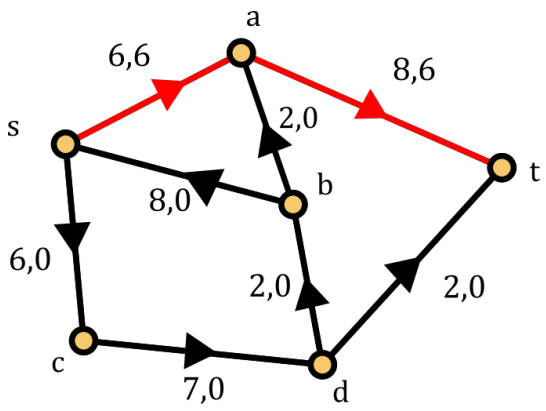

Figure 5: First run

We start the algorithm again

1. We label the source $(*, +, \infty)$. We scan, finding vertices a, b, c.
   (a) We do not label a as the edge (s, a) is saturated.
   (b) We do not label b as the edge (b, s) is not used.
   (c) We label c with $(s, +, 6)$.
2. We scan from c, finding d.
   (a) We label d with $(c, +, 6)$.
3. We scan from d, finding b and t.
   (a) We label t with $(d, +, 2)$ and the algorithm halts.

We found another augmenting path 6
We restart the algorithm

1. We label the source $(*, +, \infty)$. We scan, finding vertices a, b, c.
   (a) We do not label a as the edge (s, a) is saturated.
   (b) We do not label b as the edge (b, s) is not used.
   (c) We label c with $(s, +, 4)$.
2. We scan from c, finding d.
   (a) We label d with $(c, +, 4)$.
3. We scan from d, finding b and t.
   (a) We do not label t because (d, t) is saturated.

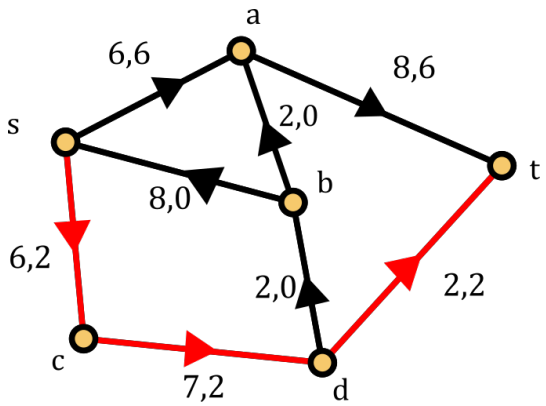

Figure 6: Second run

(b) We label b with $(d, +, 2)$.

4. We scan from b, finding s and a.

   (a) s is already labeled

   (b) We label a by $(b, +, 2)$.

5. We scan from a, finding t.

   (a) We label t by $(a, +, 2)$ and the algorithm halts.

The end result is the maximal flow on Figure 7

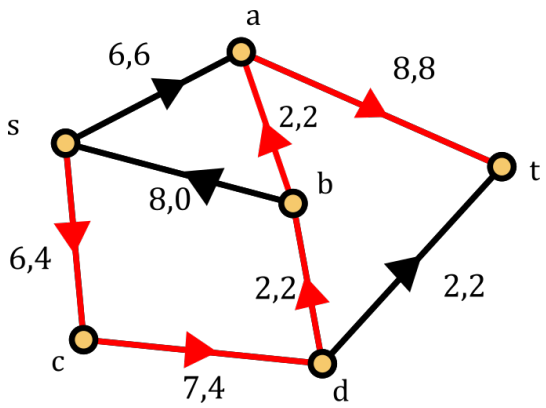

Figure 7: Third run

You can check that a next run of the algorithm would not be able to label the sink.

Corollary 5.1. *If the capacities in a network are all integers, there exist a maximum flow which assigns an integer value to every edge.*

Proof. This follows from the way the algorithm updates the labels, if all the capacities are integers, we will always update the flow by some integer values.

Q.E.D.

6 The Marriage Theorem

As an application, we will prove a famous theorem of graph theory using the max flow/min cut theorem. More applications of network flows to combinatorics are in Chapter 14 of [KT17].

Definition 6.1. *Let $G = (V, E)$ $V = X \cup Y$ be a bipartite graph. A **matching** is a subset $E_1 \subseteq E$ such that no vertex is incident with more than one edge in E_1. A **complete matching** from X to Y is a matching such that every vertex in X is incident with an edge in E_1.*

For a graph $G = (V, E)$ and a subset $A \subseteq V$ we define $N(A) = \{v \in V | v \text{ is adjacent to a vertex in } A\}$. It's sometimes called the neighborhood of A.

Theorem 6.2 (Hall's Marriage Theorem, Theorem 14.2 in [KT17])**.** *A bipartite graph $G = (V, E)$ with bipartition $V = X \cup Y$ has a complete matching if and only if for any subset $S \subseteq X$, we have $|S| \leq |N(S)|$.*

Proof. If the condition fails for some subset then clearly a matching is impossible, so the condition is necessary.

To prove that it is also sufficient, consider a network as in Figure 8

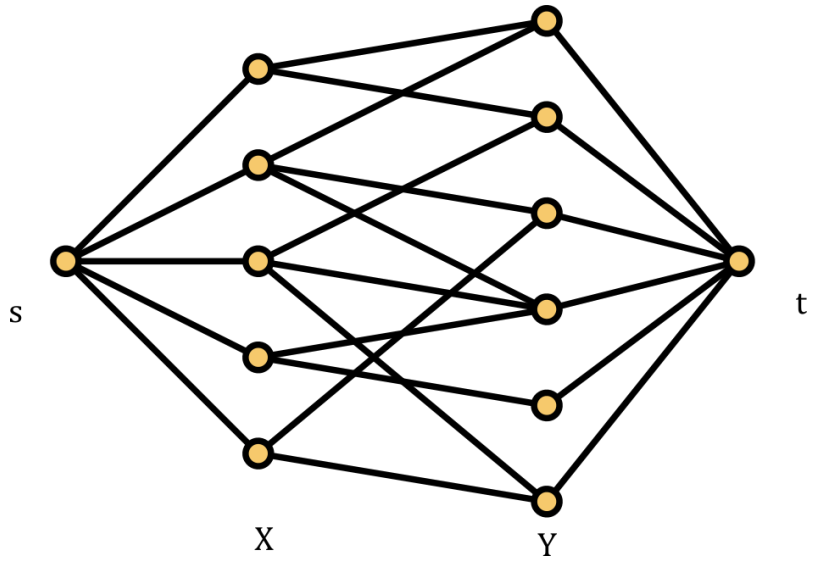

Figure 8: The network from a bipartite graph

Orient all edges from left to right, and assign capacity 1 to all edges from the sink to X, infinity to all edges between X and Y and 1 to all edges from Y to t. A minimum cut in this network will have capacity $\leq |X|$, since the cut $\{s\} \cup (X \cup Y \cup \{t\})$ has capacity $|X|$.

If the minimum cut has capacity $|X|$, then there exists a maximum flow whose values are integers. Since each vertex in $|X|$ has inflow 1 and integral outflow, and each vertex in $|Y|$ has at most 1 outflow, so can have at most 1 inflow, all edges between X and Y that have positive flow must have flow value exactly 1. These edges then define a complete matching from X and Y.

So if there is a flow (necessarily maximal) of strength $|X|$, we have a complete matching. If the maximum flow has strength strictly less than $|X|$, there must be a minimum cut of that capacity. Assume such a minimum cut exists. Since the edges between X and Y have infinite capacity, they can not be a part on any minimum cut. Therefore a minimum cut can not involve any edges that are between X and Y. This means that if a minimum cut contains any subset $A \subseteq X$, it must contain $N(A)$ as well, otherwise it would contain an infinite-capacity edge. By the same argument, there can not be edges from $X \setminus A$ to $N(A)$. Therefore we can compute the capacity of the minimum cut as

$$|X| - |A| + |N(A)|.$$

(see Figure 9 for an example where this fails)

But we assumed that for any subset $A \subseteq X$, $|N(A)| \geq |A|$, so this is a contradiction. Therefore there is no cut with capacity strictly less than $|X|$.

Q.E.D.

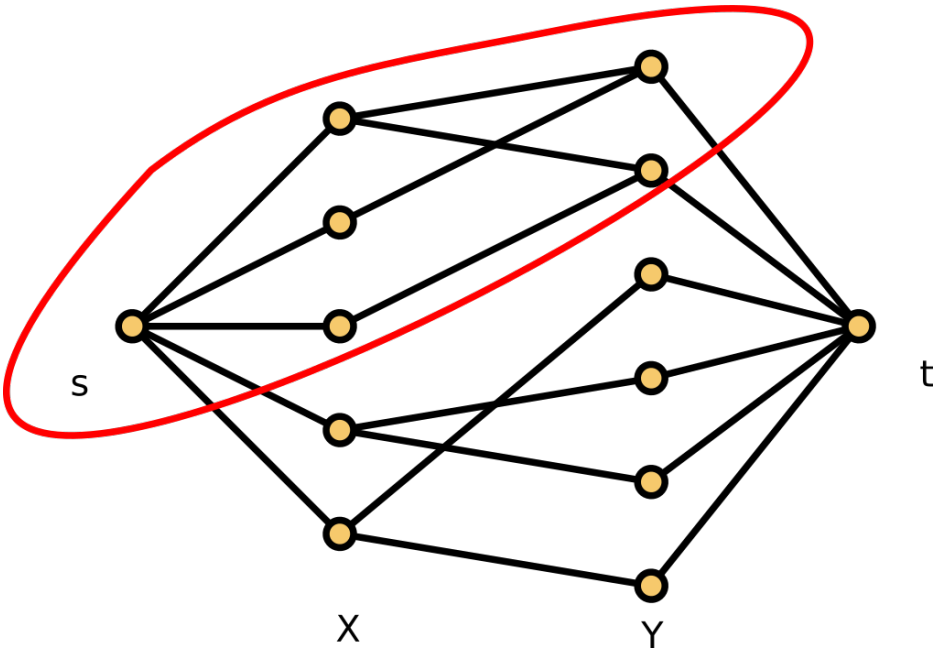

Figure 9: The minimum cut identifying the subset failing the condition

References

[KT17] Mitchel T. Keller and William T. Trotter. *Applied Combinatorics*. Open access, 2017. Available at http://www.rellek.net/appcomb/. 1, 6