CS 3510 Algorithms Last updated: October 17, 2023

Lecture 14: Max-Flow Min-Cut

Lecturer: Sahil Singla *Scribe(s): Joseph Gulian, Allen Chang*

1 Maximum Flow Problem

The maximum flow problem in a network is a lot like matchings; we can actually solve maximum matchings using network flows. In this lecture we'll talk about flows, and two related problems dealing with network flows: maximum flow and minimum cut. We will discuss the most well-known algorithm to solve this problem as well as some of it's shortcomings.

2 Network Flows

Networks are a special type of graph $G = (V, E)$ with a source $s \in V$, a sink $t \in V$, and a non-negative capacity on each edge c_e (we'll only be dealing with integral capacities). Consider an $s - t$ path, any path in the network from s to t; all nodes in the network will be in an $s - t$ path. Lastly if there is an edge from u to v, there may not be an edge from v to u.

Now that we have an understanding of the network let us look at the flow through the network which must fit two constraints. The capacity constraint says that each edge has a flow f_e which validates the inequality $0 \leq f_e \leq c_e$, which is to say flows are non-negative but no more than the capacity. The conservation constraint says that the flow into a node must equal the flow leaving a node, or more formally $\sum_{(w,u)\in E} f_{(w,u)} = \sum_{(u,z)\in E} f_{(w,u)}$. Now under these constraints there can be many valid flows, but often we will care about the maximum flow.

Below is an an example network we will use to compute the flow. Assuming the maximum capacity on each edge is 1, what is the maximum flow between s and t. We can see that we can add a flow of 1 along the path $s \to a \to b \to t$. However this is not optimal, because we can actually get a total flow of 2 from s to t by using the paths $s \to a \to t$ and $s \to b \to t$ each with a flow of one.

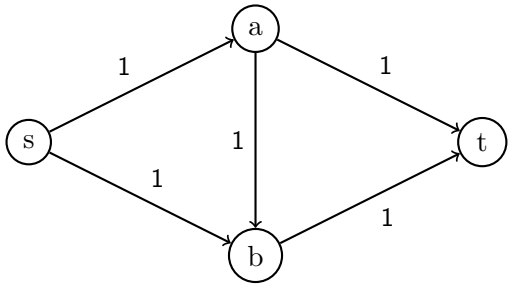

Figure 1: Simple Flow Network

You may be wondering when something like this would come up, and you could imagine that we need to move items through a logistics network or oil through pipes between cities. Looking at the graph below we can analyze the amount of movement that is possible through the network. We could also look at where the flow is limited, for instance, we could look at which edges can be increased to increase the overall flow.

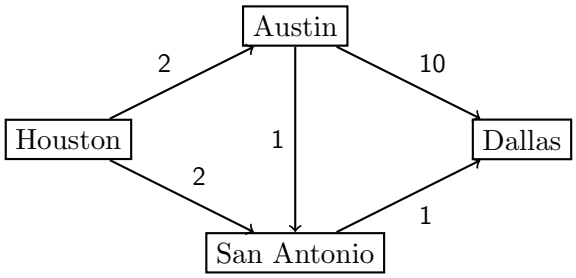

Figure 2: A simple flow network illustrating oil movement between cities in Texas.

Although this is an interesting problem on its own, there's other reasons we're interested in finding maximum flows. The first is that we can learn interesting techniques in algorithmic development, and the other reason is that we can solve other problems. Consider the matching problem from last class, given some matching like below, we can attach nodes s and t as well as directions. Now we have a network on which we could find the maximum flow which would equate to the matchings.

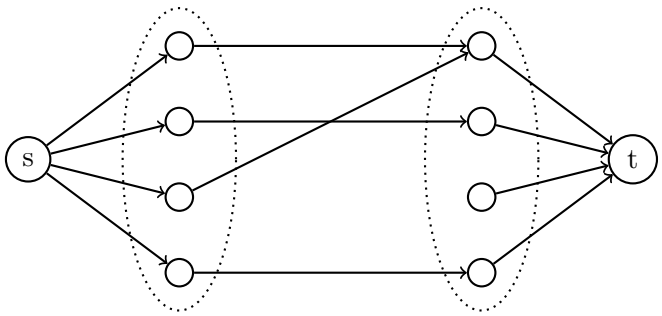

Figure 3: A Bipartite Graph Solved With Flow

Notice that we match the constraint that each node can only be matched once because the capacity from the source and sink to each node is 1.

3 Ford-Fulkerson Algorithm

How can we find the maximum flow? The idea is a lot like matchings; at each stage we have some flow through the network, and we attempt to increase that flow a little bit until we can not. In matchings when we make a mistake by picking an edge that should not be in the matching, we can easily undo that by finding an augmenting path.

We'd like to reuse this idea, and to achieve it we'll use a residual network. The essential idea of a residual network is that we can use it to understand how we can make adjustments

to our flows. One way to think about this would be that each edge in the residual network holds the "remaining capacity" in the flow network. Then the idea of Ford-Fulkerson becomes rather intuitive: find some $s-t$ path with available capacity and increase the flow along that path. As we increase the flow along each edge though, we must also increase the capacity going backwards. This essentially allows us to undo any mistake we make, by lowering the flow on some edge.

To formalize this idea of a residual network, we'll call the residual network $G^f = (V, EE^R)$. The capacity on each edge is defined by $c_e - f_e$ for $e \in E$ and f_e for $e \in E^R$.

Then to reiterate what we said before, the algorithm is quite simple: we find an $s-t$ path in G^f, and we update f to be the flow of each edge plus the flow of the path.

3.1 Example

Let's give an example; below we will run the algorithm. On the left, we have the total computed flow in the current state of the algorithm. On the right, we have the residual graph. Since we have not found an $s-t$ path yet, the flow network is 0 everywhere.

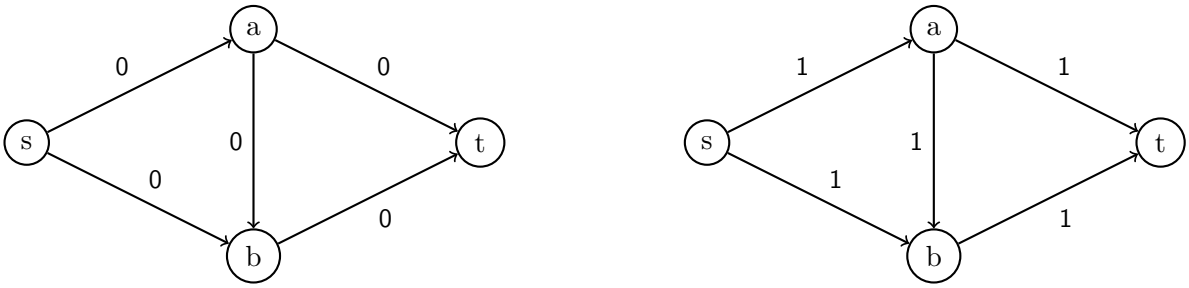

Figure 4: Simple Flow Network and a Residual Graph

Now if we take the path from s to t through a, b, and we operate this algorithm we get the following graph.

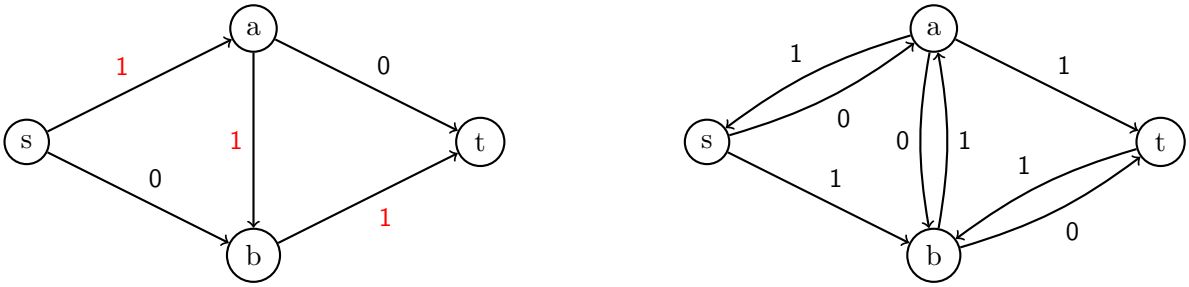

Figure 5: Simple Flow Network and a Residual Graph adjusted by one s-t path.

This is not the maximum flow in the graph, and in fact, we made a mistake. The maximum flow in this graph does not use the edge from a to b, but here's the beauty of the algorithm, we can simply undo this by taking the path from $s \to b \to a \to s$. As a reminder, although there was no edge from b to a in the original graph, we added an edge

with capacity one when we took that path in the previous step. Then if we perform another step with this edge, we will essentially "undo" that mistake by using the edge from $b - a$. Finally we'll end with a flow of 2.

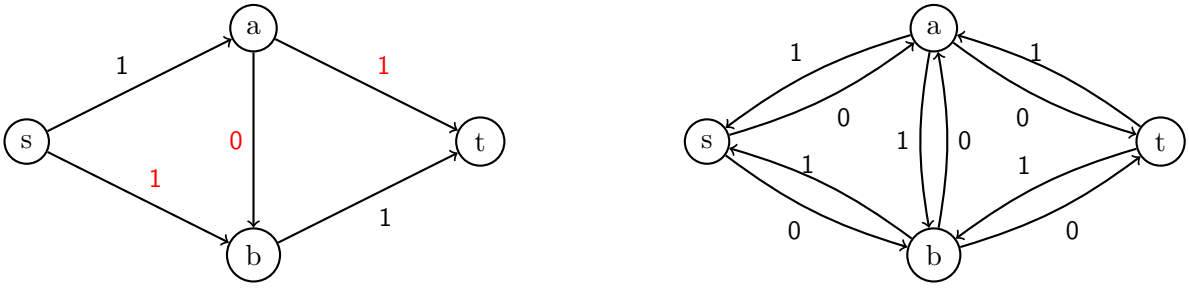

Figure 6: Simple Flow Network and a Residual Graph adjusted by one s-t path.

3.2 Runtime

How to we find the maximum flow with this algorithm? As it turns out we'll know we have the maximum flow when we are unable to find an $s - t$ path in the residual graph. We'll provide proof of this later, but for now if we simply repeat the algorithm repeatedly, we will achieve a maximum flow.

This leads to another question: how many iterations will it take? Well, if we only increase by one at each step, then the runtime of this algorithm is the magnitude of the maximum flow. With integer capacities the max-flow becomes dependent on the capacities, and more specifically we spend exponentially more time per bit of capacity we add. [1] [2] [3]

We would like to prove that this algorithm is correct. Specifically, we will prove the main lemma, that when there exists no $s - t$ path, we have found the maximum flow.

4 Min-Cut Problem

Before we discuss this though, we come across another interesting property: the minimum cut. The minimum cut is useful for us because it is hard to directly show that the flow across a network is maximum, but changing the maximum flow problem in to the minimum cut problem allows us to prove this more easily. In this problem we create an $s - t$ cut in the graph where s is in one part and t is in the other part. We can define this as a partition of G such partitions X and $V \backslash X$. In the above example, if we created a cut with vertices $X = \{s, a, b\}$, the capacity across this cut will be 2.

The key idea is that we want to find the minimum capacity crossing this cut, as it can be observed that value of the maximum flow is less than the value of the minimum cut (if

[1] Remember we care about runtime with respect to the size of the input, if capacities are represented 9 bit numbers, we would only have a max flow around 512, but if we add one more bit, suddenly we may have a max flow around 1024.

[2] It is "Pseudo-polynomial", and it would be good to understand this concept since students frequently get tripped up with it.

[3] As a remark, there does exists a polynomial time algorithm that computes the maximum flow even when the maximum flow is bounded exponentially.

it was not, the flows across the minimum cut would be violated, a contradiction).

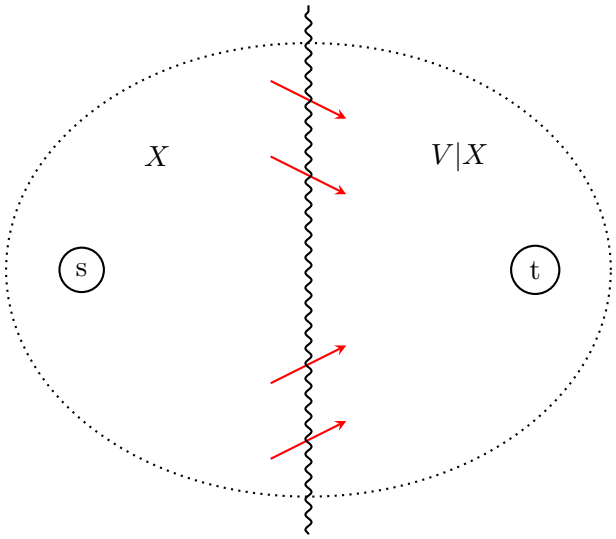

Figure 7: An $S - T$ cut

It turns out that the values are actually equal. Not only do cuts provide an upper bound of the maximum flow, but it is true that the maximum flow is equal to the minimum cut.

Proof Sketch. Let X be all vertices reachable from S in G^f, we claim that the cut between X and $V|X$ is the flow. Essentially if an edge crosses the cut, X to $V|X$, this will mean that the edge is absent. Since those edges are absent, this means that the flow across the network is at least the capacity of the cut. However we also no that there are no other flows from $V|X$ to X, which means whatever flow we have is maximal (showing that when no $s - t$ path exists the flow is maximal).