CS512 LECTURE NOTES - LECTURE 18

1 Network Flow

We will now explore a problem that can be solved using the techniques of computer science based on graph algorithms that we have been talking about.

Suppose that we are given a distribution network, where oil is sent from a node called the source to another node called the sink, but the pipes that connect the vertices have a finite capacity. The problem we are asked to solve consists of finding the maximum amount of "oil" that can flow through the network subject to the capacities of the pipes.

Let us formalize the problem description:

1.1 Max-flow problem

Input:

- A directed graph $G = (V, E)$ with two special vertices $s \in V$ (called a source) and $t \in v$ (called a sink), where the source vertex only has out-edges, and the sink vertex only has in-edges,
- A capacity function $c : E \to \mathbb{R}^+$ where

Output:

We want to find a flow function, i.e. $f : E \to \mathbb{R}^+ \cup \{0\}$ that tells us how much "oil" can be sent through each one of the edges. The flow function that we are looking for has to satisfy the following properties:

1. The flow through each edge must not be negative and cannot be greater than its capacity:

$$\forall (u, v) \in E \quad 0 \leq f(u, v) \leq c(u, v)$$

2. **Flow conservation**: except for the source and the sink vertices the flow going into a node must equal the flow coming out of the node.

$$\forall v \in V - \{s, t\}, \quad \sum_{(u,v)\in E} f(u, v) = \sum_{(v,u)\in E} f(v, u)$$

3. We want to find a flow function that maximizes the flow that can be sent through the network. Notice that the total flow that can be sent through the network is called the value of the flow and can be computed as the total flow coming out of the source:

$$v(f) = \sum_{(s,v)\in E} f(s, v)$$

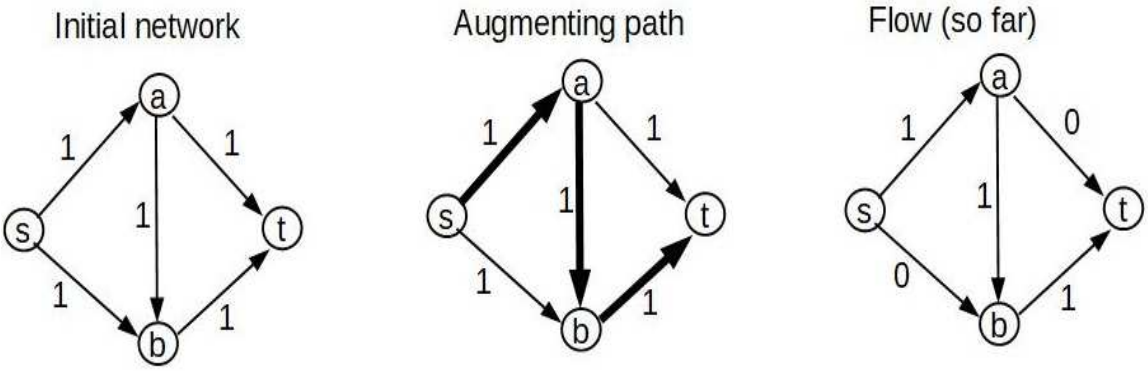
   So we want to find a flow f^* such that its value is maximized, i.e.

$$v(f) \leq v(f^*) \text{ for every flow } f$$

So the problem is to find a flow function f such that $v(f)$ is maximum.

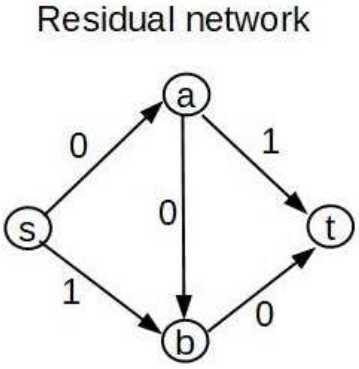
1.2 Idea of the algorithm

Let us see how to tackle this problem using known algorithms. To do this look at the following network:



The idea is to find a path (using BFS, DFS, or other path finding algorithm) from s to t that can be used to send a positive flow from s to t. This path is called an *augmenting path*. Suppose that the algorithm returns the augmenting path *sabt* in the case of the example. Notice that the maximum flow that can flow through that path is equal to the capacity of the minimum edge traversed, in this case equal to 1. So we can start building the flow function (see flow so far in the previous figure).

To continue, we can now compute a network with whatever capacities remain so that we can try to find another path. Let us call this network the "*residual network*", where the residual capacity of each edge in the direction of the capacity is computed by subtracting the flow from the capacity.



Notice that in the example there is no new augmenting path that

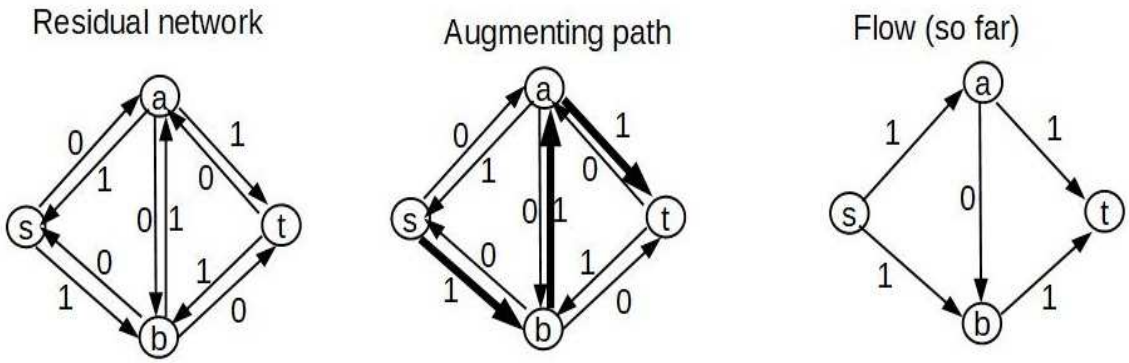
can be used to increment the flow. However, it is clear that we do not have the optimal solution yet.

The problem is that edge (a, b) is not used on the optimal solution, so we need to figure out a way in which the algorithm can give that edge back.

The idea is to assign edges going in the opposite direction of their capacities, representing the amount of flow that they carry, or said another way, the flow that can be given back. So the capacities c^f of the residual network can be computed in the following way:

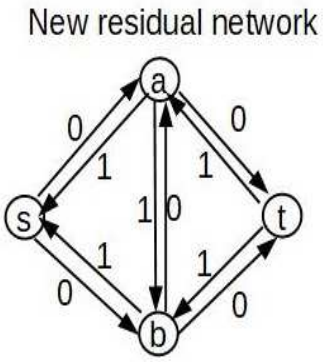
$$c^f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if} \quad (u,v) \in E \\ f(v,u) & \text{if} \quad (v,u) \in E \end{cases}$$

Using this new idea we can go through a second iteration, finding a new augmenting path and a new flow.



Notice that the new flow is computed by adding the old flow to the capacity of the augmenting path (called the bottleneck capacity). If the edge in the augmenting path goes backwards, then flow is subtracted.

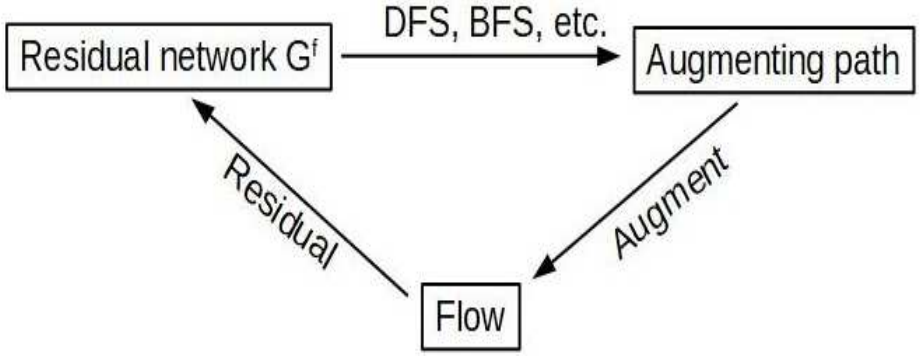
We can now compute a new residual network:



Notice that there is no augmenting path on this network, so we stop and claim that the flow that was found corresponds to a maximum flow (we will prove this fact later).

1.3 Ford and Fulkerson's Algorithm

We are now in a position to write down an algorithm to solve the problem. The process that we have followed can be illustrated in the following chart.



We start with an initial flow where $\forall(u, v) \in E\ f(u, v) = 0$, and a residual network G^f (computed from this initial flow), from which we obtain an augmenting path using our favorite algorithm (DFS, BFS). Once we have an augmenting path, we can update the flow using a method that we will call `Augment`, once we have the new flow, we can use it to compute the new residual network.

Pseudocode of the algorithm

Augment(P, f, c)
P is an augmenting path
f is the current flow
c is the original capacity function

b=bottleneckCapacity(P)
for each $(u, v) \in P$
    if $(u, v) \in E$
        $f(u, v) = f(u, v) + b$
    else
        $f(v, u) = f(v, u) - b$
return f

Residual(f, c)
f is the current flow
c is the original capacity function

for each $(u, v) \in E$
    $w(u, v) = c(u, v) - f(u, v)$
    $w(v, u) = f(u, v)$
return w

Ford_Fulkerson(G,c)
$G = (V, E)$ is a directed graph
c is the capacity function

for each $(u, v) \in E$
    f(u,v)=0
w^f=Residual(f, c)
while $\exists$ augmenting path P in $G^f = ((V, E \cup E^R), w^f)$
    f=Augment(P, f, c)
    w^f=Residual(f, c)
return f