Design and Analysis of Algorithms II Lecture 6: Improving Network Flow Algorithms

Gordon Lu (gl1589@nyu.edu)

October 12, 2023

1 Ford-Fulkerson Recap & Issues

In this lecture, the main focus will be getting Network Flow algorithms to be faster.

1.1 Recap: Ford-Fulkerson

1. Sometimes FF doesn't terminate.
    - FF terminates in the integral case (only integer flows / capacities)
2. If FF terminates, it produces max-flow.

1.2 Ford-Fulkerson Issues

There are some things to worry about when running FF:

1. How many augmentations will there be?
2. How much progress is in an augmentation?
3. How fast is an augmentation?

1.3 Ford-Fulkerson: Integral Case

Let $n = |V|$, $m = |E|$, U = max capacity (integers), and f^* = maximum flow. Each augmentation is $O(|f^*|)$.

- Each augmentating path will increase the flow by at least 1, and is therefore upper bounded by $O(|f^*|)$.

To find an augmenting path, it is O(m).

- We use an exploration algorithm such as BFS, DFS, etc. which will take O(m).
    - Note that it is really O(m+n), but since $m \geq n-1$ for a Flow Network (Connected graph), O(m+n) = O(m).

- We also need to construct the residual graph, which will take O(m+n), but this will get us O(2m), and we can drop the constant to get O(m).

We can bound $|f^*|$ by nU.
Therefore, the total running time of the simplest implementation of FF for the integral case is $O(mnU)$.

- This is called pseudo-polynomial time.

2 The Edmonds-Karp Algorithm

We follow the typical FF algorithm for finding maximum flow, but use BFS to find augmenting paths.

- This is called the Shortest Paths algorithm (in terms of links / hops)
- It will eventually terminate in polynomial time and has no dependence on capacities.

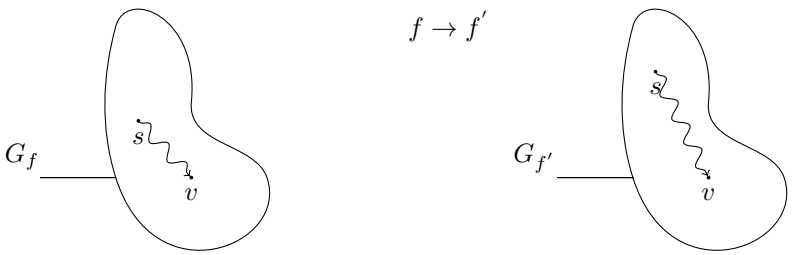
2.1 Edmonds-Karp Running Time: Part I

Claim:
The number of augmentations using Edmonds-Karp is $f^* = mn \implies O(mn)$.
The total running time is $O(m^2n)$.

The proof of this claim is based on distances between vertices in the residual network G_f. We introduce the idea of distance below:
Definition:
$\delta_f(u, v)$ = Length of Shorted Directed Path $u \rightsquigarrow v$ in G_f.

- This is the shortest directed path in terms of links / hops (if we considered each edge to have unit distance).

Lemma: Let s, t be the source and sink vertices.
Any $\delta_f(s, v)$ in G_f monotonically increases with each augmentation.
$v \neq s,t$
The below figures demonstrate what the lemma claims:



Proof (of Lemma):
For the sake of contradiction, suppose that $\delta_{f'}(s,v) < \delta_f(s,v)$ (*).

1. Pick v such that $\delta_{f'}(s,v)$ is minimum.
2. Let $p = s \rightsquigarrow u \rightsquigarrow v$ be a shortest path in $G_{f'}$ such that $(u,v) \in E_{f'}$ and $\delta_{f'}(s,u) = \delta_{f'}(s,v) - 1$
    - This simply means that the on the path from (s,v), u is one node away from v.

    Because of the way v was chosen, we know that $\delta_{f'}(s,u) \geq \delta_f(s,u)$. In other words, the distance from s to u did not decrease.

We claim that $(u,v) \notin E_f$ (The old residual graph). Why?
If $(u,v) \in E_f$, then we also have:

$$\delta_f(s,v) \leq \delta_f(s,u) + 1 \leq \delta_{f'}(s,u) + 1 = \delta_{f'}(s,v).$$

This contradicts our original assumption (*).

- Remember that v is picked such that $\delta(s,v)$ is minimum in $G_{f'}$, it does not guarantee any behavior about G_f.

How can we have $(u,v) \notin E_f$ and $(u,v) \in E_{f'}$?

- The augmenting path p that produced f' from f must have included (v,u).

As $(u,v) \in E_{f'}$, while $(u,v) \notin E_f$, we know that (v,u) must have been included in the augmenting path p with flow f. In order to get to $(u,v) \in E_{f'}$, we must have included the backedge (u,v) in the augmenting path p' with flow f'.

- Therefore, we can deduce that in the original graph, G, $(u,v) \notin G$, while $(v,u) \in G$. If $(u,v) \in G$, we would have been able to include (u,v) in the augmenting path p with flow f.

In particular, we have:

$$\delta_f(s,v) = \delta_f(s,u) - 1 \leq \delta_{f'}(s,u) - 1 = \delta_{f'}(s,v) - 2$$

which contradicts our assumption (*). We conclude that our assumption that such a vertex v exists is incorrect. ■

2.2 Edmonds-Karp Running Time: Part II

Theorem:

The number of augmentations in Edmonds-Karp is $O(mn)$.

Proof:
Definition:
An edge (u,v) in the residual network G_f is **critical** on an augmenting path p if the residual capacity of (u,v) is the bottleneck of the augmenting path.

- In other words, if $c_f(p) = c_f(u,v)$, (u,v) is critical.

Claim:
Each edge can become critical at most $\frac{n}{2}$ times.

- Note that $m_f \leq 2m$, since the edges in E_f are either the edges in E, their reversals, or both.

Proof of Claim:
Our intuition behind our proof is based on observations as critical edges progress through Edmonds-Karp.
Since augmenting paths are shortest paths (using Edmonds-Karp), when (u, v) is critical for the first time:

$$\delta_f(s, v) = \delta_f(s, u) + 1$$

Once the flow is augmented, the edge (u, v) disappears from the residual network, as it will be saturated.

- It cannot reappear on another augmenting path until the flow from (u, v) is decreased. In other words, when (v, u) appears on the augmenting path, (u,v) can be critical.

If f' is the flow when this event happens, then:

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1$$

- Since the edge (v, u) is on the augmenting path.

Since $\delta_f(s, v) \leq \delta_{f'}(s, v)$, we have:

$$\begin{aligned}\delta_{f'}(s, u) &= \delta_{f'}(s, v) + 1 \\ &\geq \delta_f(s, v) + 1 \\ &= \delta_f(s, u) + 2\end{aligned}$$

So, from the time that (u, v) becomes critical to when it next becomes critical, the distance of u to the source increases by at least 2.

- Since (u, v) is on an augmenting path, we know that $u \neq t$, therefore in any residual network that has a path from s to u, the shortest such path has at most $n - 2$ edges.

- So, after the first time (u, v) becomes critical, it can become critical at most $(n - 2)/2 = (n/2) - 1$, for a total of $n/2$ times.

Since there are $O(m)$ pairs of vertices that can have an edge between them in the residual network, the total number of critical edges during Edmonds-Karp is $O(mn)$. Each augmenting path therefore has at least one critical edge and hence the theorem holds. ■

As it takes $O(m)$ to find augmenting paths using BFS for Edmonds-Karp, the total running time is $O(m^2n)$.

3 Ford-Fulkerson: More Tricks

We will now discuss other ways to find augmenting paths.

3.1 Wide Pipe Algorithm

In this algorithm, we choose the augmenting path with max improvement.

- This is a modification of Dijkstra's Algorithm.
- Each augmentation takes $O(m \log U)$, so the total running time is $O(m^2 \log U)$.
- We store the best bottleneck at each level.

3.2 Edmonds-Karp Algorithm

This was discussed in detail above. The total running time is $O(m^2 n)$.

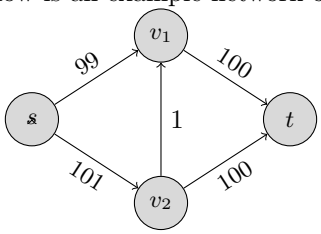
- Can get down to $O(mn^2)$ with some improvements (will not be discussed).

3.3 Capacity Scaling

In this algorithm, we focus on edges with larger capacities and gradually work down to smaller edges. This is especially if the flow network has lots of large and small edges.

- Each augmentation takes $O(m \log U)$, so the total running time is $O(m^2 \log U)$.
- We choose a large number and ignore anything below this number. This is typically the largest power of $2 < U$.
- Once we find an augmenting path, we repeat with a smaller power of 2 (divide it by 2) until we have found all augmenting paths.

Below is an example network of where capacity scaling works well:



We would start with our number as 64, and ignore all capacities < 64. Once we have found an augmenting path, the number becomes 32 and so on...

3.4 Dinitz / Dinic's Algorithm: Blocking Flow

There are two ways of augmenting flow:

- **Usual Way:** Augmenting flow by increasing flow along augmenting path by bottle neck: $f \to G_f \to p \to f \uparrow p$
- **New Way**: Use blocking flow f of $G_f \to f \uparrow f'$

Dinitz's Algorithms works as follows:

1. Construct the residual network, G_f from G.
2. Construct the Layered Network, G_L from G_f by performing a BFS from the source, s.
3. Perform BFS to find paths from $s \rightsquigarrow t$ until we reach a blocking flow, and go back to step 2. We stop when we cannot reach the sink from the Layered Network.

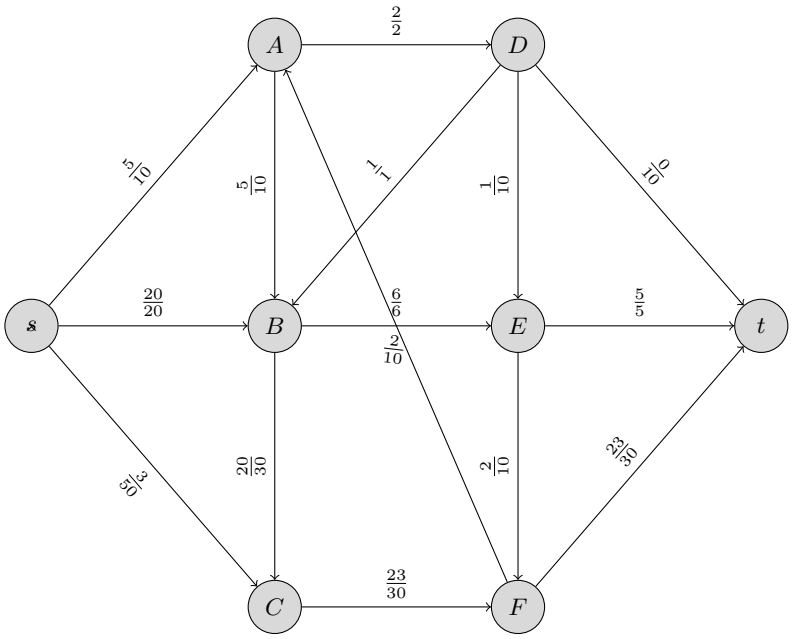
Definition: A **layered network**, G_L is constructed by performing a BFS on the residual network, G_f. It is defined as so:

1. We only include edges from layer L_i to L_{i+1}.
    - Backedges are ignored.

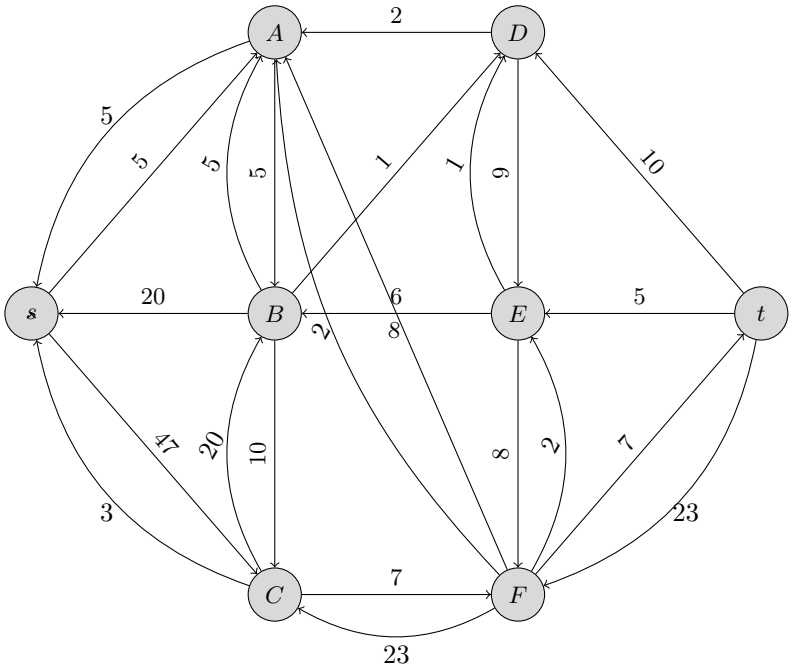
Definition: A **blocking flow**, f is a flow that can't be increased by an augmenting path.

1. When there are no more augmenting paths to be found with BFS, we have found a blocking flow, as we cannot augment the flow any further.
2. Note that a maximum flow is a blocking flow, but a blocking flow is not always a maximum flow.

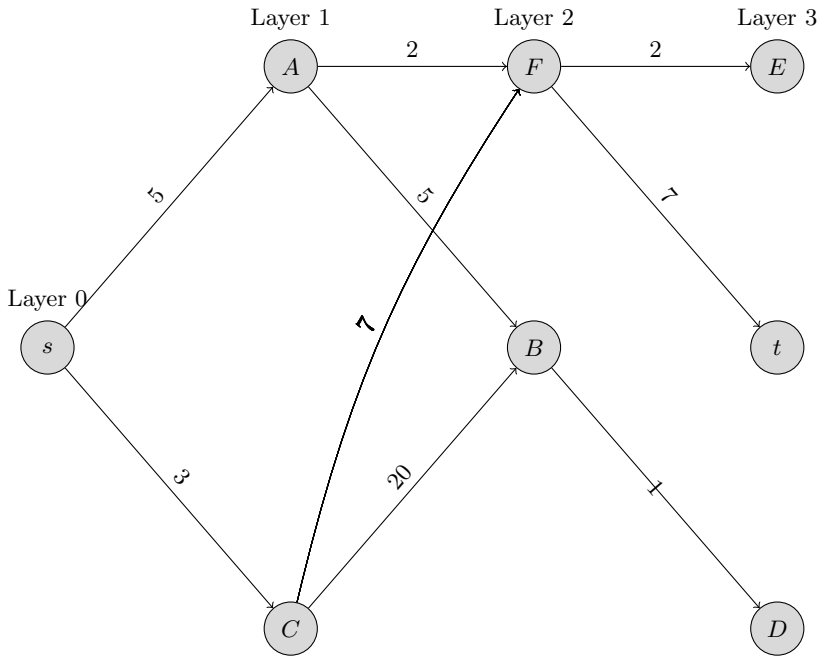
Example:
Suppose that we want to find the maximum flow on the following flow network:



We construct the residual network G_f as so:



We now construct the layered network G_L by determining all nodes reachable from the source to sink using BFS:



So the blocking flow consists of:

1. s, A, F, t with 2 units of flow,
2. s, C, F, t with 3 units of flow.

Therefore the blocking flow for this layered network is 5 units, with $|f| = 5$. We would repeat this process again once the flows have been augmented in the residual network.

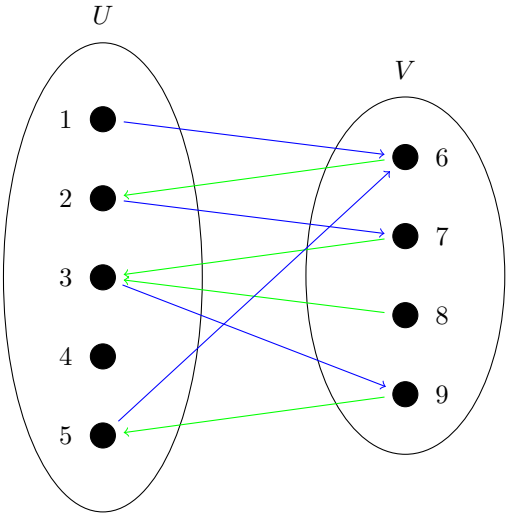
4 Network Flow Applications:

4.1 Bipartite Matching

A **bipartite graph** G, is defined as: $G = (V, E)$, where $V = A \cup B$ and $E = \{(u, v) | u \in A, v \in B\}$

- A and B are disjoint sets, and edges in E join the disjoint sets.

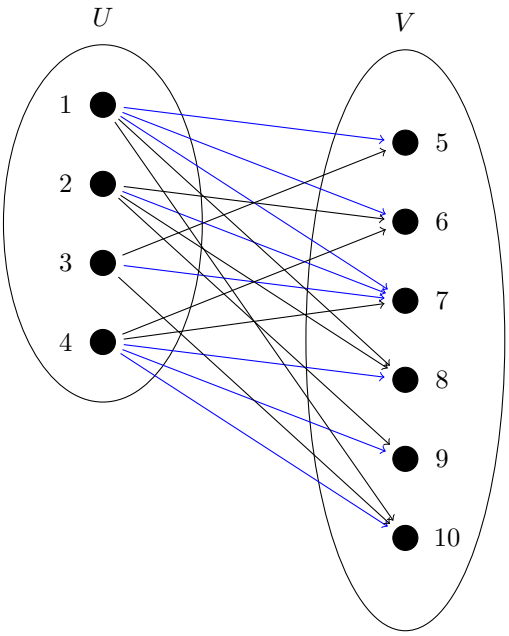
Below is an example of a bipartite graph:



Definition: We define a *Matching*, as a subset of edges $M \subseteq E$ such that for all vertices $v \in V$, at most one edge is incident on v.

- Formally, this means that if $(v, u) \in M \ \ \& \ \ (v', u) \in M \implies v = v'$ and $(v, u) \in M \ \ \& \ \ (v, u') \in M \implies u = u'$.

Below is an example of a matching. The blue edges are edges in the Matching, while the black edges are not.



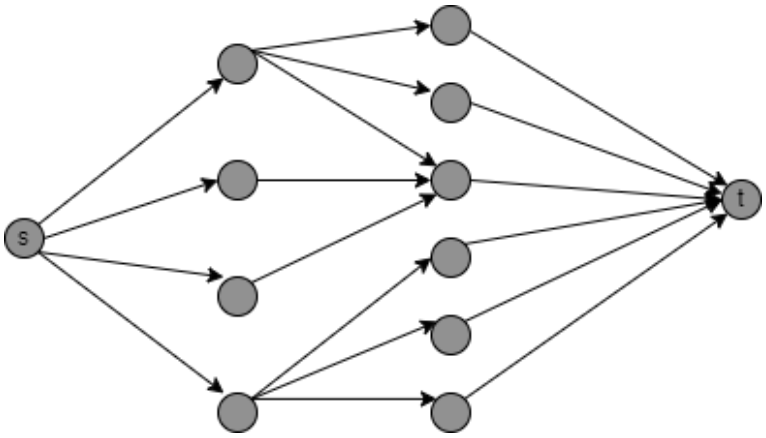
4.2 Maximum (Cardinality) Matching

There are lots of possible matchings, but what about getting the largest matching? Therefore, we now focus on **Maximum Matching**.
Definition:
A **maximum matching** is a matching of maximum cardinality. It is a matching M such that for any matching M', $|M| \geq |M'|$.

Finding a maximum matching:
We can convert a bipartite graph, G into a flow network quite easily:

1. We create new vertices s and v (source and sink)
2. Assign capacity 1 to all flows.
3. Connect s to all vertices in U and connect all vertices in B to t, where $V = A \cup B$.

Below is the matching from the above graph converted as a flow network. Let's call it G':



We now claim the following:

1. For any integral flow, f in G', $\exists$ a matching M in G with $|M| = |f|$.
    - Each vertex in A only has one edge entering in with at most 1 unit of flow in. By flow conservation, 1 unit must also leave. So this flow can enter at one edge and leave at one edge, which is a vertex in B. Therefore, M is a matching. It is straightforward to show that $|M| = |f|$.
2. For any matching M in G, $\exists$ an integral flow f in G' with $|M| = |f|$.
    - Each edge (u, v) in M corresponds to 1 unit of flow in G' that traverses $s \rightsquigarrow u \rightsquigarrow v \rightsquigarrow t$. It is straightforward to show that $|M| = |f|$ from this point.

The claims are intuitive if we go through a couple of examples and compare the constructed flow network G' to any matching in G.

Since we know that $|M| = |f|$, if we want the maximum cardinality matching, we can simply run Ford-Fulkerson to maximize flow on the converted flow network.

1. The integrality theorem is very important to get this to work.
2. Ford-Fulkerson takes $O(mnU) = O(mn)$ here since $U = 1$.

An even more fascinating problem is the **Min Cost Perfect Bipartite Matching**, but this will not be discussed.