Lecture 2: Network Flows

A wide variety of problems, including the matching problems discussed in the previous lectures, can be described and analyzed in terms of Flow Networks. In this lecture, we introduce the basic forms and laws governing Flow Networks, and develop several theorems useful for their analysis.

1 Definitions

1.1 Flow Networks

A flow network $G = (V, E)$ is a directed graph such that:

- $G$ contains a source node $s$ and a sink node $t$
- every vertex $v \in V$ is on some path between $s$ and $t$
- every edge $(u, v) \in E$ has a capacity $c(u, v) \geq 0$
- nodes have no storage capacity

Figure 1 shows an example of a flow network.

1.2 Flows

A flow $f$ in a network $G$ is a function that assigns real numbers to edges such that following constraints are satisfied:

- capacity constraint: $\forall(u, v) \in E, f(u, v) \leq c(u, v)$

---

1These lecture notes are adopted in part from a set of lecture notes scribed by Jared Hunter and Daniel Vukelich in 2004.
Figure 1: A flow network $G = (V, E)$ with edge capacities specified.

- **skew symmetry:** $\forall (u, v) \in E, f(u, v) = -f(v, u)$
- **flow conservation:** $\forall u, v \notin \{s, t\}, \sum f(u, v) = 0$

Since $c(u, v)$ is defined for each pair of nodes for which $\exists (u, v) \in E$, we can think of the capacity constraint as merely stating that no flow between two directly connected nodes can possibly exceed the capacity of the edge that exists between them.

The skew symmetry constraint follows from the fact that flow networks are directed graphs. Another definition would be $\forall u, v \in V, f(v, u) = -f(u, v)$, for only flows directed contrary to the underlying path direction will ever be labeled with negative values.

The flow conservation constraint applies to all nodes except source $s$ and sink $t$, because the output potential of a source and input potential of a sink are both $\infty$. For all other nodes, however, the total outbound flow must always equal the total inbound flow (nodes have no storage capacity), and semantics imposed by the skew symmetry constraint force the sum of all flows to zero.
1.3 Values of Flows and Related Properties

The value of a flow $f$ is defined as the total of all flows outward from the source $s$ which is equal to the total of flows inward to the sink $t$ (by flow conservation constraint):

$$ |f| = \sum_{v \in V} f(s, v) = \sum_{u \in V} f(u, t) \quad (1) $$

For sets of nodes $X, Y \subseteq V$ following properties hold:

$$ f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y) \quad (2) $$

$$ f(X, X) = 0 \quad (3) $$

$$ f(X, Y) = -f(Y, X) \quad (4) $$

and if $X \cap Y = \emptyset$:

$$ f(X \cup Y, Z) = f(X, Z) + f(Y, Z) \quad (5) $$

$$ f(Z, X \cup Y) = f(Z, X) + f(Z, Y) \quad (6) $$
The total flow between two sets of nodes is the sum of the flows from any member of the first set to a member of the second set. When the sets are identical, total net flow is zero by the flow conservation constraint. When the sets are not identical, the total net flows from each set to the other have equal absolute value and opposing sign. When the two sets are disjoint, the total net flow from their union to any third set \( Z \subseteq V \) is the sum of their individual net flows to that subset, and vice versa.

1.4 Residual Networks

Instead of viewing a flow network in terms of capacities in use, it can be useful to view it in terms of capacities remaining (unused capacities). This representation is called a residual network. Given a network \( G = (V, E) \) and a flow \( f \) in \( G \), the residual network is \( G_f(V, E_f) \) where:

- \( E_f = \{(u, v) \mid c_f(u, v) > 0\} \)
- \( c_f(u, v) = c(u, v) - f(u, v) \)

So if there is no remaining capacity along an edge in the flow network, that edge should not appear in our chosen representation of \( G_f \), an effect illustrated in Figure 3.

We can also draw residual networks with a complete set of back edges, whose values maintain the residual network’s compliance with the flow conservation constraint. For simplicity, these edges (call them \( E_b \)) are shown separately in Figure 4, these edges are called backward edges. The \( G_f \) is really defined by \( (V, E_f \cup E_b) \).

It’s important to note that the values of the backward edges are not always simply the values of the forward flows in \( G \). In cases where a network specifies forward edges \((i, j)\) and \((j, i)\) (as our network does between nodes A and C, for example), the values of the residual edges may be calculated. Given flows \( x_{ij} \) and \( x_{ji} \), and unused capacities \( u_{ij} \) and \( u_{ji} \) respectively:

- residual edge \( r_{ij} = (u_{ij} - x_{ij}) + x_{ji} \), for forward edges
- residual edge \( r_{ji} = (u_{ji} - x_{ji}) + x_{ij} \), for backward edges
Figure 3: The forward edges of residual network resulting from the flow in Figure 2.

Figure 4: The backward edges of residual network resulting from the flow in Figure 2.
Figure 5: The residual network $G_f$ resulting from the flow $f$ in Figure 2 with augmenting path $p$ shaded.

1.4.1 Augmenting Paths

If $G_f$ contains a path $p$ from $s$ to $t$, $p$ is an augmenting path with respect to the $f$ in $G$ that formed $G_f$:

- $|f| \leftarrow |f| + c(p)$
- $c(p) = \min\{c_f(u, v) \text{ for } \forall (u, v) \in p\}$

Figure 5 shows the residual network $G_f$ resulting from flow $f$ in Figure 2. The shaded edges demonstrate an augmenting path $p$.

1.5 Cuts

A cut of $G(V, E)$ is a partition of $V$ into $S, T$.

- $s \in S$, and $t \in T$
- $S \cup T = V$, and $S \cap T = \emptyset$
The **capacity** of a cut \((S, T)\) is:

\[
c(S, T) = \sum_{u \in S, v \in T} c(u, v)
\]  

(7)

**Claim 1.5.1** While the capacities of different cuts may be different, the total value of flow across a cut boundary is always constant.

![Figure 6: Three cuts of a flow network.](image)

We will not prove the claim here, but will try to bolster its intuitiveness with an example. Figure 6 shows cuts of our familiar flow network. We will calculate the capacity and flow across each cut separately.

- **\(S_1 = \{s\}, T_1 = \{A, B, C, D, t\}\)**
  
  
  \[
c(S_1, T_1) = c(s, A) + c(s, C) = 16 + 13 = 29
  
  f(S_1, T_1) = f(s, A) + f(s, C) = 9 + 11 = 20
  
- **\(S_2 = \{s, A, B, C, D\}, T_2 = \{t\}\)**
  
  
  \[
c(S_2, T_2) = c(B, t) + c(D, t) = 20 + 4 = 24
  
  f(S_2, T_2) = f(B, t) + f(D, t) = 16 + 4 = 20
  
- **\(S_3 = \{A, s, B, D\}, T_3 = \{C, t\}\)**
  
  
  \[
c(S_3, T_3) = c(A, C) + c(s, C) + c(B, C) + c(B, t) + c(D, t)
  
  c(S_3, T_3) = 4 + 13 + 9 + 20 + 4 = 50
  
  f(S_3, T_3) = f(s, C) + f(B, t) + f(D, t) - f(C, D) = 11 + 16 + 4 - 11 = 20
2 The Max-Flow Problem

Given \((G, s, t)\), the Max-Flow problem is that of finding a maximal-capacity flow in \(G\) from source \(s\) to sink \(t\). We call the value of this maximal flow \(f^*\).

2.1 The Max-Flow Min-Cut Theorem

The Max-Flow Min-Cut theorem states that \(f^*\), the value of the maximum flow, is equal to the minimum cut capacity out of all possible cuts of \(G\).

Theorem 2.1.1 (Maxflow/Mincut Theorem) Given a network, \(G\), and a flow, \(f\), the following are equivalent:

1. \(f\) is a maximum flow in \(G\)
2. the residual network \(G_f\) contains no augmenting paths
3. we can find some cut \((S, T)\) for which \(|f| = c(S, T)\)

Proof.

\(\cdot 1 \rightarrow 2\). If \(G_f\) contained an augmenting path then we could improve \(f\), in which case \(|f|\) would not be maximum and \(f\) would not be a maximum flow.

\(\cdot 2 \rightarrow 3\).

1. assume \(G_f\) has no augmenting path
2. let \(S = \{v \in V \text{ such that } v \text{ is reachable from } s \text{ in } G_f\}\)
3. let \(T = V - S\)
4. notice \(s \in S\), but \(t \notin S\). \(t\) is not reachable in \(G_f\) because there are no augmenting paths
5. \(\forall u \in S, \forall v \in T: f(u, v) = c(u, v)\)
   (a) assume \(f(u, v) \leq c(u, v)\) (maybe we can improve \(f(u, v)\))
(b) if \((u, v) \in E'\) (the edges of \(G_f\)), then \(v \in S\)
(c) but \(v \notin S\), so \(\rightarrow\leftarrow\)

6. 
\[
|f| = f(S, T) = \sum_u \sum_v f(u, v) = \sum_{u \in S} \sum_{v \in T} c(u, v) = c(S, T)
\]

• 3 \(\rightarrow\) 1.
  1. assume \(|f| = c(S, T)\)
  2. but \(|f| \leq f^* \leq c(S, T)\)
  3. so \(|f| = c(S, T) \rightarrow |f| = f^*\)

2.2 The Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm initializes a zero (null) flow in the network and then repeatedly augments the initial (null) flow with any path that augments it.

\textbf{FF} \((G, s, t)\):

1. Initialize flow \(f\) to 0
2. while there exists an augmenting path \(p\) in \(G_f\)
   • do augment \(f\) along \(p\)
3. return \(f\)

The first few iterations of the Ford-Fulkerson algorithm are shown in Figure 7. In the figure, (a) shows the initial network, and (b) is the residual network induced by choosing the augmenting path along the top of the graph. Within (b), the next augmenting path is shown in bold lines, and (c) shows the original network augmented with this path. Finally, (d) shows the residual network corresponding to (c), from which we may select a subsequent augmenting path. The algorithm terminates when there are no more augmenting paths.
Claim 2.2.1 The Ford-Fulkerson algorithm always terminates and always returns a maximum flow.

2.2.1 Time Complexity of Ford-Fulkerson Algorithm

Initialization of $f$ and the construction of $G_f$ are both $O(|E|)$, as is the process of finding an augmenting path. The while loop is executed at most $|f^*|$ times, since the augmenting path increases flow by at least one unit each iteration. It is therefore easily seen that worst case bounds are $O(|f^*||E|)$.

Now, consider the flow in Figure 8. In (a), a flow of 1 unit follows the path of dotted lines and, (b) shows the resulting residual network. Now, suppose the next augmenting path is that highlighted in (b); once again, an increase in flow of 1 unit will result, and this alternating pattern can continue $|f^*|$ times, exhibiting the worst case performance. For improved performance, we next sketch the Edmonds-Karp algorithm.
2.3 The Edmonds-Karp Algorithm

Instead of choosing any augmenting path each time through the while loop, we might instead choose, among augmenting paths to choose, a path that optimizes some criteria. The Edmonds-Karp algorithm always chooses a path with the minimum number of comprising edges. It makes a new network from residual network where edge weights are replace by '1' and then employs a breadth-first search to choose the shortest augmenting path from $s$ to $t$. That shortest augmenting path corresponds to an augmented path in the residual network with the fewest number of edges. Referring to Figure 8, choosing path with the fewest number of edges (namely the paths of two edges of 1,000,000 along the top and bottom of the graph) would have readily achieved the 2,000,000 capacity maximum flow by avoiding the extra edge of capacity 1 (the bottleneck). Moreover, maximum flow would then have been realized in two iterations of the while loop instead of 2,000,000 by Ford-Fulkerson!

2.3.1 Time Complexity of Edmonds-Karp Algorithm

**Lemma 2.3.1** If the E-K algorithm is run on a flow network with source $s$ and sink $t$, then for all verticie $v \in V - \{s,t\}$, the shortest path distance can only increase in the residual network.
Proof (by contradiction).

Let \( f \) be the flow before the first augmentation that decreases some shortest path distance and let \( f' \) be the flow just afterwards. Let \( v \) be the closest vertex to \( s \) in \( f' \)'s residual network: \( d_{f'}(s, v) \leq d_f(s, v) \) Let \( p = s \to u \to v \) be the shortest path from \( s \) to \( v \) in \( G_f \) such that \((u, v) \in E_f' \) and \( d_{f'}(s, u) = d_f(s, v) - 1 \). We choose \( v \) to be the closest vertex to \( s \) with distance decreasing. So \( d_{f'}(s, v) \leq d_f(s, u) \).

Claim 2.3.2 \((u, v) \notin E_f \): edge \((u, v)\) has been disappeared from \( E_f \)

Proof.

\[
(u, v) \in E_f \Rightarrow d_f(s, v) \leq d_f(s, u) + 1 \\
\leq d_{f'}(s, v) + 1 \\
= d_{f'}(s, u) + 1 \rightarrow \leftarrow
\]

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

It must be that we increased the flow from \( v \) to \( u \). Since E-C algorithm always augments flow along shortest path, the shortest path from \( s \) to \( u \) in \( G_f \) has \((v, u)\) as its last edge:

\[
d_{f'}(s, v) = d_f(s, u) - 1 \leq d_{f'}(s, u) = d_f(s, v) - 2
\]

but we assumed \( d_{f'}(s, v) \leq d_f(s, v) \) and it is a contradiction \( \rightarrow \leftarrow \).

Theorem 2.3.3 Edmonds-Karp algorithm will go for at most \( O(|V||E|) \) iterations

Proof.

Say edge \((u, v)\) in a residual network \( G_f \) is a critical on augmenting path \( p \) if the residual capacity of \( p \) is the residual capacity of \((u, v)\). That is if \( c_f(p) = c_f(u, v) \), at least one edge on every augmenting path is critical. We will show an edge can become critical at most \(|V|/2 - 1\) times.
Let $u, v$ be vertices in $V$ connected by an edge in $E$. We claim that from the time $(u, v)$ is critical to the next time it becomes critical again, its distance to $s$ must increase by 2.

Let $f$ be the flow when its *first* critical: $d_f(s, v) = d_f(s, u) + 1$ (since augmenting paths are shortest paths). Then $(u, v)$ disappears from residual network can not appear again until $(u, v)$ appears on an augmenting path. For some flow $f'$:

$$
d_{f'}(s, u) = d_{f'}(s, v) + 1
\geq d_{f'}(s, u) + 1
= d_f(s, u) + 2 \rightarrow \leftarrow
$$