Maximum Network Flows, the Max Flow-Min Cut Theorem, and the Ford-Fulkerson Algorithm

- The Problem:

  Given a directed graph D in which each arc (u, v) has a finite non-negative weight, $c(u, v)$, and in which two vertices, s and t, are labelled, determine the maximum throughput from s to t, treating the arc weights as maximum capacities.

- Flows and Flow Augmenting Paths

  A *flow* is an assignment to each arc (u, v) a usage or load $f(u, v)$ such that

  1. $f(u, v) \leq c(u, v)$, and
  2. for each $v \in V(D) \setminus \{s, t\}$, $\displaystyle\sum_{u \in V(D)} f(u, v) = \sum_{u \in V(D)} f(v, u)$.

  A *k-flow* is a flow in which $\displaystyle\sum_{u \in V(D)} f(s, u) = \sum_{u \in V(D)} f(u, t) = k$.

  Initially, we can always find a 0-flow, given that each arc has non-negative weight.

  In order to devise an algorithm which finds a maximum flow, we require an efficient method of taking a k-flow and finding a better flow. This is accomplished by considering *flow augmenting paths*. A *flow augmenting path* (which we will often abbreviate to *fap*) is a path $v_0, v_1, \ldots, v_{m-1}, v_m$ such that:

  1. $s = v_0$, and
  2. $t = v_m$, and
  3. either $f(v_i, v_{i+1}) < c(v_i, v_{i+1})$ or $f(v_{i+1}, v_i) > 0$.

  So, a fap is an undirected path from s to t for which arcs traversed in agreement with their direction have unused capacity in the current flow, while arcs traversed against their direction have non-zero load in the current flow.

  Given a k-flow and a fap, we can find a larger flow by augmenting the k-flow along the fap. Specifically, for each arc in the fap which is traversed in agreement with its direction, we increase the load by 1, and for each arc traversed against its direction we decrease the load by 1. (Note here that we have assumed that each arc's weight is integral; in the event of non-integer weights, we would augment not by 1, but rather by the minimum of the unused capacities and current loads)

  This provides us with the basis for an algorithm to find the maximum flow:

  ```
  1. Start with a 0-flow.
  2. Find a fap and augment the current flow.
  3. Repeat (2) until no further fap's can be found.
  ```

For such an algorithm to be both correct and efficient, we need to show that fap's can be found efficiently, and that the absence of a fap implies the current flow is maximum. Note that there is indeed a maximum possible flow, since each arc has finite weight.

- The Max Flow-Min Cut Theorem

We first define an *s-t cut* to be an ordered pair $(S,\bar{S}) = \{(u,v) \in A(D) \mid u \in S, v \in \bar{S}\}$, where $\bar{S} = V(D) \setminus S$, $s \in S$, and $t \in \bar{S}$.

Lemma 1 *For any k-flow and any s-t cut $(S,\bar{S})$, $f(S,\bar{S}) - f(\bar{S},S) = k$. That is:*

$$\sum_{(u,v)\in(S,\bar{S})} f(u,v) \quad - \sum_{(u,v)\in(\bar{S},S)} f(u,v) \;=\; k.$$

Proof: For each $v \in V(D) \setminus \{s,t\}$, we have $\sum_{u\in V(D)} f(u,v) = \sum_{u\in V(D)} f(v,u)$. So, for each $v \in \bar{S} \setminus \{t\}$, we have:

$$\sum_{u\in V(D)} f(u,v) - \sum_{u\in V(D)} f(v,u) = 0.$$

And for t we have:

$$\sum_{u\in V(D)} f(u,t) = k.$$

We add these equations for each vertex in $\bar{S}$ together. Notice that when we perform this addition, each arc (i,j) with $i \in \bar{S}$ and $j \in \bar{S}$ will appear twice, once as $f(i,j)$ and once as $-f(i,j)$, and hence the two appearances will cancel each other. What remains of this addition is therefore:

$$\sum_{(u,v)\in(S,\bar{S})} f(u,v) \quad - \sum_{(u,v)\in(\bar{S},S)} f(u,v) \;=\; k.$$

□

What this effectively means, in plain English, is that the net flow across any s-t cut, from S to $\bar{S}$, in any k-flow is precisely k.

Lemma 2 *Given a k-flow and an s-t cut $(S,\bar{S})$, $k \le c(S,\bar{S})$. That is:*

$$k \le \sum_{(u,v)\in(S,\bar{S})} c(u,v).$$

Moreover, $k = c(S,\bar{S})$ if and only if $f(u,v) = c(u,v)$ for each $(u,v) \in (S,\bar{S})$ and $f(u,v) = 0$ for each $(u,v) \in (\bar{S},S)$.

Proof: From Lemma 1, it follows that $k = f(S,\bar{S}) - f(\bar{S},S) \le c(S,\bar{S})$. If $k = c(S,\bar{S})$, then necessarily $f(S,\bar{S}) = c(S,\bar{S})$ and $f(\bar{S},S) = 0$. And, conversely, if $f(S,\bar{S}) = c(S,\bar{S})$ and $f(\bar{S},S) = 0$, then $c(S,\bar{S}) = f(S,\bar{S}) - f(\bar{S},S) = k$. □

The consequence of Lemma 2 is that if we have a k-flow in which there is some s-t cut $(S,\bar{S})$ for which $f(S,\bar{S}) = k$, then the load travelling across this cut is travelling entirely in one direction, namely from S to $\bar{S}$.

We are now ready for the Max Flow-Min Cut Theorem:

Theorem 1 *Let $(S,\bar{S})$ be an s-t cut for which $c(S,\bar{S})$ is minimum over all possible s-t cuts. Then the maximum flow possible is a $c(S,\bar{S})$-flow.*

In English, the size of a maximum flow is equal to the capacity of a minimum s-t cut.

Proof: Suppose we have a k-flow which is maximum. Let $S = \{u \in V(D) \mid \exists$ a fap from s to $u\}$. Notice that $s \in S$. Also notice that $t \in \bar{S}$, since there is no fap from s to t (otherwise our k-flow would not be maximum). We will now show that $c(S,\bar{S})$ is equal to the size of our flow (ie. that $c(S,\bar{S}) = k$).

Consider any arc $(u,v) \in (S,\bar{S})$. If $f(u,v) < c(u,v)$, then there would exist a fap from s to v and we would have previously let $v \in S$; but we did not and hence it follows that $f(u,v) = c(u,v)$ for all arcs $(u,v) \in (S,\bar{S})$.

Now consider any arc $(u,v) \in (\bar{S},S)$. If $f(u,v) > 0$, then there would exist an fap from s to u and we would have previously let $u \in S$; but we did not and hence it follows that $f(u,v) = 0$ for all arcs $(u,v) \in (\bar{S},S)$.

By Lemma 2, it now follows that $k = c(S,\bar{S})$. □

Corollary 1 *Any k-flow for which no s-t fap's exist is a maximum flow.*

Proof: If no s-t fap's exist, then we select S as in the proof of the Max Flow-Min Cut Theorem and find that $k = c(S,\bar{S})$ and hence that k is a maximum flow. □

- The Ford-Fulkerson Algorithm

  We now have shown that our short algorithm for finding a maximum flow is indeed correct. What remains is to devise an efficient means of finding an s-t fap if one exists.

  One way in which this could be accomplished is by constructing a directed graph D' having the same vertex set as the digraph D. For the arcs of D', we include (u,v) if (u,v) is in the original graph and $f(u,v) < c(u,v)$, and we include (v,u) if (u,v) is in the original graph and $f(u,v) > 0$. To find a fap, it now suffices to find any directed path in D' from s to t; we can accomplish this by means of the Bellman-Ford algorithm.

  However, we need not construct an auxiliary digraph as just suggested; we could use the Ford-Fulkerson algorithm as outlined below. The algorithm not only finds a fap, but augments the current flow by the maximum amount possible (recall we had naïvely assumed earlier that each arc's weight was integral and that we always augmented by 1).

1. `Initialisation...`
   Let $S = \emptyset$ and assign to s the label $(-, \infty)$.
2. `Test for maximum flow...`
   If each labelled vertex is in S, then the current flow is maximum and $(S, \bar{S})$ is a minimum s-t cut.
   Otherwise, choose a labelled vertex $u \notin S$; suppose that u has the label $(p^{\pm}, \epsilon)$.
3. `Scanning...`
   For any unlabelled vertex v such that $(u, v) \in A(D)$ and $f(u, v) < c(u, v)$, assign to v the label $(u^+, \min\{\epsilon, c(u, v) - f(u, v)\})$.
   For any unlabelled vertex v such that $(v, u) \in A(D)$ and $f(v, u) > 0$, assign to v the label $(u^-, \min\{\epsilon, f(v, u)\})$.
   Let $S = S \cup \{u\}$, denoting that we have now scanned u.
   If t is unlabelled, then return to (2); otherwise proceed to (4).
4. `We have a fap along which we augment...`
   Suppose that t has label (w, ϵ) and let $v = t$.
   While $v \neq s$
       if v has label (u^+, δ)
         then let $f(u, v) = f(u, v) + \epsilon$
       else if v has the label (u^-, δ)
         then let $f(v, u) = f(v, u) - \epsilon$
       let $v = u$

To now find a maximum flow in a directed graph, we could begin with a 0-flow and repeatedly use this algorithm until a maximum flow and an accompanying minimum cut are found in step (2).