Part I

Algorithm(s) for Maximum Flow

Greedy Approach

- Begin with $f(e) = 0$ for each edge
- Find a $s$-$t$ path $P$ with $f(e) < c(e)$ for every edge $e \in P$
- Augment flow along this path
- Repeat augmentation for as long as possible.
Greedy Approach: Issues

- Begin with $f(e) = 0$ for each edge
- Find a $s$-$t$ path $P$ with $f(e) < c(e)$ for every edge $e \in P$
- Augment flow along this path
- Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!
Need to “push-back” flow along edge $(u, v)$

Residual Graph

Definition

For a network $G = (V, E)$ and flow $f$, the residual graph $G_f = (V', E')$ of $G$ with respect to $f$ is

- $V' = V$
- **Forward Edges**: For each edge $e \in E$ with $f(e) < c(e)$, we have $e \in E'$ with capacity $c(e) - f(e)$
- **Backward Edges**: For each edge $e = (u, v) \in E$ with $f(e) > 0$, we have $(v, u) \in E'$ with capacity $f(e)$
Residual Graph Example

Figure: Flow in red edges

Figure: Residual Graph

Residual Graph Property

Observation: Residual graph captures the “residual” problem exactly.

Lemma

Let \( f \) be a flow in \( G \) and \( G_f \) be the residual graph. If \( f' \) is a flow in \( G_f \) then \( f + f' \) is a flow in \( G \) of value \( v(f) + v(f') \).

Lemma

Let \( f \) and \( f' \) be two flows in \( G \) with \( v(f') \geq v(f) \). Then there is a flow \( f'' \) of value \( v(f') - v(f) \) in \( G_f \).

Definition of + and - for flows is intuitive and the above lemmas are easy in some sense but a bit messy to formally prove.
Residual Graph Property: Implication

Recursive algorithm for finding a maximum flow:

MaxFlow(G, s, t):
   If the flow from s to t is 0
      return 0
   Find any flow f with v(f) > 0 in G
   Recursively compute a maximum flow f' in G_f
   Output the flow f + f'

Iterative algorithm for finding a maximum flow:

MaxFlow(G, s, t):
   Start with flow f that is 0 on all edges
   While there is a flow f' in G_f with v(f') > 0 do
      f = f + f'
      Update G_f
   endWhile
   Output f

Ford-Fulkerson Algorithm

for every edge e, f(e) = 0
G_f is residual graph of G with respect to f
while G_f has a simple s-t path
   let P be simple s-t path in G_f
   f = augment(f, P)
   Construct new residual graph G_f

augment(f, P)
   let b be bottleneck capacity, i.e., min capacity of edges in P
   for each edge (u,v) in P
      if e=(u,v) is a forward edge
         f(e) = f(e) + b
      else (* (u,v) is a backward edge *)
         let e = (v,u) (* (v,u) is in G *)
         f(e) = f(e) - b
   return f
Example

Example continued
Example continued

\[\begin{array}{c}
\text{Example continued} \\
\begin{array}{c}
\begin{array}{c}
\text{u} \\
\text{s} \\
\text{v} \\
\text{t}
\end{array}
\end{array}
\end{array} \]

\[\begin{array}{c}
\begin{array}{c}
\text{Example continued} \\
\begin{array}{c}
\begin{array}{c}
\text{u} \\
\text{s} \\
\text{v} \\
\text{t}
\end{array}
\end{array}
\end{array} \]

\[\begin{array}{c}
\begin{array}{c}
\text{Example continued} \\
\begin{array}{c}
\begin{array}{c}
\text{u} \\
\text{s} \\
\text{v} \\
\text{t}
\end{array}
\end{array}
\end{array} \]

\[\begin{array}{c}
\begin{array}{c}
\text{Example continued} \\
\begin{array}{c}
\begin{array}{c}
\text{u} \\
\text{s} \\
\text{v} \\
\text{t}
\end{array}
\end{array}
\end{array} \]
Lemma

If \( f \) is a flow and \( P \) is a simple \( s-t \) path in \( G_f \), then \( f' = \text{augment}(f, P) \) is also a flow.

Proof.

Verify that \( f' \) is a flow. Let \( b \) be augmentation amount.

- **Capacity constraint:** If \((u, v) \in P\) is a forward edge then \( f'(e) = f(e) + b \) and \( b \leq c(e) - f(e) \). If \((u, v) \in P\) is a backward edge, then letting \( e = (v, u) \), \( f'(e) = f(e) - b \) and \( b \leq f(e) \). Both cases \( 0 \leq f'(e) \leq c(e) \).

- **Conservation constraint:** Let \( v \) be an internal node. Let \( e_1, e_2 \) be edges of \( P \) incident to \( v \). Four cases based on whether \( e_1, e_2 \) are forward or backward edges. Check cases (see fig next slide).
Properties about Augmentation: Integer Flow

**Lemma**

At every stage of the Ford-Fulkerson algorithm, the flow values \( f(e) \) and the residual capacities in \( G_f \) are integers.

**Proof.**

Initial flow and residual capacities are integers. Suppose lemma holds for \( j \) iterations. Then in \( j + 1 \)st iteration, minimum capacity edge \( b \) is an integer, and so flow after augmentation is an integer.

---

Progress in Ford-Fulkerson

**Proposition**

Let \( f \) be a flow and \( f' \) be flow after one augmentation. Then \( v(f) < v(f') \).

**Proof.**

Let \( P \) be an augmenting path, i.e., \( P \) is a simple \( s-t \) path in residual graph.

- First edge \( e \) in \( P \) must leave \( s \).
- Original network \( G \) has no incoming edges to \( s \); hence \( e \) is a forward edge.
- \( P \) is simple and so never returns to \( s \).
- Thus, value of flow increases by the flow on edge \( e \).
Termination Proof

Theorem

Let $C$ be the minimum cut value; in particular $C \leq \sum_{e \text{ out of } s} c(e)$. Ford-Fulkerson algorithm terminates after finding at most $C$ augmenting paths.

Proof.

The value of the flow increases by at least 1 after each augmentation. Maximum value of flow is at most $C$.

Running time

- Number of iterations $\leq C$
- Number of edges in $G_f \leq 2m$
- Time to find augmenting path is $O(n + m)$
- Running time is $O(C(n + m))$ (or $O(mC)$).

Efficiency of Ford-Fulkerson

Running time $= O(mC)$ is not polynomial. Can the running time be as $\Omega(mC)$ or is our analysis weak?

Ford-Fulkerson can take $\Omega(C)$ iterations.
Correctness of Ford-Fulkerson Augmenting Path Algorithm

**Question:** When the algorithm terminates, is the flow computed the maximum s-t flow?

Proof idea: show a cut of value equal to the flow. Also shows that maximum flow is equal to minimum cut!

Recalling Cuts

**Definition**

Given a flow network an s-t cut is a set of edges $E' \subseteq E$ such that removing $E'$ disconnects s from t: in other words there is no directed $s \rightarrow t$ path in $E - E'$. Capacity of cut $E'$ is $\sum_{e \in E'} c(e)$.

Let $A \subseteq V$ such that
- $s \in A$, $t \not\in A$
- $B = V - A$ and hence $t \in B$

Define $(A, B) = \{(u, v) \in E \mid u \in A, v \in B\}$

**Claim**

$(A, B)$ is an s-t cut.

Recall: Every minimal s-t cut $E'$ is a cut of the form $(A, B)$.
Ford-Fulkerson Correctness

**Lemma**

*If there is no s-t path in G_f then there is some cut (A, B) such that v(f) = c(A, B)*

**Proof.**

Let A be all vertices reachable from s in G_f; B = V \ A

- s \in A and t \in B. So (A, B) is an s-t cut in G
- If e = (u, v) \in G with u \in A and v \in B, then f(e) = c(e) (saturated edge) because otherwise v is reachable from s in G_f

\[ \nabla f = \nabla \text{out}(A) - \nabla \text{in}(A) = c(A, B) - 0 = c(A, B) \]

**Lemma Proof Continued**

**Proof.**

- If e = (u', v') \in G with u' \in B and v' \in A, then f(e) = 0 because otherwise u' is reachable from s in G_f
- Thus,

\[ v(f) = \nabla \text{out}(A) - \nabla \text{in}(A) = \nabla \text{out}(A) - 0 = c(A, B) - 0 = c(A, B) \]
Ford-Fulkerson Correctness

**Theorem**

The flow returned by the algorithm is the maximum flow.

**Proof.**

- For any flow $f$ and $s$-$t$ cut $(A, B)$, $v(f) \leq c(A, B)$
- For flow $f^*$ returned by algorithm, $v(f^*) = c(A^*, B^*)$ for some $s$-$T$ cut $(A^*, B^*)$
- Hence, $f^*$ is maximum
Max-Flow Min-Cut Theorem and Integrality of Flows

**Theorem**

For any network $G$, the value of a maximum $s$-$t$ flow is equal to the capacity of the minimum $s$-$t$ cut.

**Proof.**

Ford-Fulkerson algorithm terminates with a maximum flow of value equal to the capacity of a (minimum) cut.

**Theorem**

For any network $G$ with integer capacities, there is a maximum $s$-$t$ flow that is integer valued.

**Proof.**

Ford-Fulkerson algorithm produces an integer valued flow when capacities are integers.

**Efficiency of Ford-Fulkerson**

Running time $= O(mC)$ is not polynomial. Can the upper bound be achieved?
**Question:** Is there a polynomial time algorithm for maxflow?

**Question:** Is there a variant of Ford-Fulkerson that leads to a polynomial time algorithm? Can we choose an augmenting path in some clever way? Yes! Two variants.
- Choose the augmenting path with largest bottleneck capacity.
- Choose the shortest augmenting path.

### Augmenting Paths with Large Bottleneck Capacity

- Pick augmenting paths with largest bottleneck capacity in each iteration of Ford-Fulkerson.
- How do we find path with largest bottleneck capacity?
  - Assume we know $\Delta$ the bottleneck capacity.
  - Remove all edges with residual capacity $\leq \Delta$.
  - Check if there is a path from $s$ to $t$.
  - Do binary search to find largest $\Delta$.
  - Running time: $O(m \log C)$.
- Can we bound the number of augmentations? Can show that in $O(m \log C)$ augmentations the algorithm reaches a max flow. This leads to an $O(m^2 \log^2 C)$ time algorithm.

Book gives a simpler variant called **Capacity Scaling** algorithm that runs in $O(m^2 \log C)$ time.
Augmenting Paths with Large Bottleneck Capacity

How do we find path with largest bottleneck capacity?

- Max bottleneck capacity is one of the edge capacities. Why?
- Can do binary search on the edge capacities. First, sort the edges by their capacities and then do binary search on that array as before.
- Algorithm’s running time is \( \mathcal{O}(m \log m) \).
- Different algorithm that also leads to \( \mathcal{O}(m \log m) \) time algorithm by adapting Prim's algorithm.

Removing Dependence on \( C \)

- [Edmonds-Karp, Dinitz] Picking augmenting paths with fewest number of edges yields a \( \mathcal{O}(m^2n) \) algorithm, i.e., independent of \( C \). Such an algorithm is called a strongly polynomial time algorithm since the running time does not depend on the numbers (assuming RAM model). (Many implementation of Ford-Fulkerson would actually use shortest augmenting path if they use BFS to find an \( s-t \) path).
- Further improvements can yield algorithms running in \( \mathcal{O}(mn \log n) \), or \( \mathcal{O}(n^3) \).
Finding a Minimum Cut

**Question:** How do we find an actual minimum \( s-t \) cut?

Proof gives the algorithm!

- Compute an \( s-t \) maximum flow \( f \) in \( G \)
- Obtain the residual graph \( G_f \)
- Find the nodes \( A \) reachable from \( s \) in \( G_f \)
- Output the cut \( (A, B) = \{(u, v) \mid u \in A, v \in B\} \). **Note:** The cut is found in \( G \) while \( A \) is found in \( G_f \)

Running time is essentially the same as finding a maximum flow.

**Note:** Given \( G \) and a flow \( f \) there is a linear time algorithm to check if \( f \) is a maximum flow and if it is, outputs a minimum cut. How?