COMP 355
Advanced Algorithms
More on Network Flows
Section 7.1-7.3, 7.5-7.6 (KT)
Section 26.1-26.2 (CLRS)
Abstraction for material **flowing** through the edges.

- $G = (V, E) =$ directed graph, no parallel edges.
- Two distinguished nodes: $s =$ source, $t =$ sink.
- $c(e) =$ capacity of edge $e$. (non-negative)
Def. An s-t flow is a function that satisfies:

- For each $e \in E$: \[ 0 \leq f(e) \leq c(e) \] (capacity)
- For each $v \in V - \{s, t\}$: \[ \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e) \] (conservation)

Def. The value of a flow $f$ is:

\[ v(f) = \sum_{0 \text{ out of } s} f(e). \]
**Flows**

**Def.** An \( s-t \) flow is a function that satisfies:

- For each \( e \in E \):
  \[
  0 \leq f(e) \leq c(e) \quad \text{(capacity)}
  \]

- For each \( v \in V - \{s, t\} \):
  \[
  \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e) \quad \text{(conservation)}
  \]

**Def.** The value of a flow \( f \) is:

\[
\text{Value} = \sum_{e \text{ out of } s} f(e).
\]

![Flow Network Diagram]

---

*Value = 24*
Max flow problem. Find $s$-$t$ flow of maximum value.

Value = 28
Residual Graph

Forward edges: For each edge \((u, v)\) for which \(f(u, v) < c(u, v)\), create an edge \((u, v)\) in \(G_f\) and assign it the capacity \(c_f(u, v) = c(u, v) - f(u, v)\). Intuitively, this edge signifies that we can add up to \(c_f(u, v)\) additional units of flow to this edge without violating the original capacity constraint.

Backward edges: For each edge \((u, v)\) for which \(f(u, v) > 0\), create an edge \((v, u)\) in \(G_f\) and assign it a capacity of \(c_f(v, u) = f(u, v)\). Intuitively, this edge signifies that we can cancel up to \(f(u, v)\) units of flow along \((u, v)\). Conceptually, by pushing positive flow along the reverse edge \((v, u)\) we are decreasing the flow along the original edge \((u, v)\).

(a): A flow \(f\) in network \(G\)

(b): Residual network \(G_f\)

A flow \(f\) and the residual network \(G_f\).
Augmenting Paths

Consider a network $G$, let $f$ be a flow in $G$, and let $G_f$ be the associated residual network.

**Def.** An *augmenting path* is a simple path $P$ from $s$ to $t$ in $G_f$.

**Def.** The *residual capacity* (also called the bottleneck capacity) of the path is the minimum capacity of any edge on the path. It is denoted $c_f(P)$.

**Recall:** all the edges of $G_f$ are of strictly positive capacity, so $c_f(P) > 0$.

By pushing $c_f(P)$ units of flow along each edge of the path, we obtain a valid flow in $G_f$, and by the previous lemma, adding this to $f$ results in a valid flow in $G$ of strictly higher value.
Augmenting Path Algorithm

Augment\( (f, c, P)\) {  
  \( b \leftarrow \text{bottleneck}(P) \)  
  foreach \( e \in P \) {  
    if (\( e \in E \)) \( f(e) \leftarrow f(e) + b \)  
    else \( f(e^R) \leftarrow f(e) - b \)  
  }  
  return \( f \)  
}

Ford-Fulkerson\( (G, s, t, c)\) {  
  foreach \( e \in E \) \( f(e) \leftarrow 0 \)  
  \( G_f \leftarrow \text{residual graph} \)  
  while (there exists augmenting path \( P \)) {  
    \( f \leftarrow \text{Augment}(f, c, P) \)  
    update \( G_f \)  
  }  
  return \( f \)  
}
Ford-Fulkerson Algorithm

G:

\[ G: \]

- \( s \) to 2: 10
- 2 to 3: 2
- 3 to 10: 10
- 2 to 4: 4
- 4 to 10: 10
- 4 to 5: 6
- 5 to 10: 10
- 3 to 2: 8
- 4 to 3: 9

capacity
Ford-Fulkerson Algorithm

G:

Flow value = 0
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]

Flow value = 0

Residual capacity
Ford-Fulkerson Algorithm

G:

Flow value = 8

Gf:
Ford-Fulkerson Algorithm

G:

\[ \begin{align*}
G_f: & \\
\text{Flow value} &= 10 
\end{align*} \]
Ford-Fulkerson Algorithm

G:

Gf:

Flow value = 16
Ford-Fulkerson Algorithm

G:

\[G_f:\]

Flow value = 18
Ford-Fulkerson Algorithm

G:

s

2

3

4

5

t

G_f:

Flow value = 19
Ford-Fulkerson Algorithm

G:

Flow value = 19
Cut capacity = 19

Gf:
Def. An s-t cut is a partition \((A, B)\) of \(V\) with \(s \in A\) and \(t \in B\).

Def. The capacity of a cut \((A, B)\) is: 
\[
\text{cap}(A, B) = \sum_{e \text{ out of } A} c(e)
\]

The capacity of the cut shown in the diagram is calculated as follows:

\[
\text{Capacity} = 10 + 5 + 15 = 30
\]
**Def.** An s-t cut is a partition (A, B) of V with s ∈ A and t ∈ B.

**Def.** The capacity of a cut (A, B) is: \( \text{cap}(A, B) = \sum_{e \text{ out of } A} c(e) \)

![Diagram of a network with nodes and edges labeled with capacities. The shaded area represents set A, and the unshaded area represents set B. The cut is highlighted with red nodes and edges. The capacity calculation is shown as 9 + 15 + 8 + 30 = 62.](image)
Min s-t cut problem. Find an s-t cut of minimum capacity.

Capacity = 10 + 8 + 10 = 28
Flow value lemma. Let $f$ be any flow, and let $(A, B)$ be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving $s$.

$$
\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)
$$

Flows and Cuts

Flow value lemma. Let $f$ be any flow, and let $(A, B)$ be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving $s$. 

$$
\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)
$$
Flow value lemma. Let $f$ be any flow, and let $(A, B)$ be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving $s$.

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$
Flow value lemma. Let $f$ be any flow, and let $(A, B)$ be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving $s$.

\[
\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)
\]
Flows and Cuts

Flow value lemma. Let $f$ be any flow, and let $(A, B)$ be any $s$-$t$ cut. Then

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f).$$

Pf.

$v(f) = \sum_{e \text{ out of } s} f(e)$

by flow conservation, all terms except $v = s$ are 0

$$\rightarrow = \sum_{v \in A} \left( \sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right)$$

$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e).$$
Flows and Cuts

**Weak duality.** Let $f$ be any flow, and let $(A, B)$ be any s-t cut. Then the value of the flow is at most the capacity of the cut.

Cut capacity = 30 $\Rightarrow$ Flow value $\leq$ 30
Weak duality. Let $f$ be any flow. Then, for any $s$-$t$ cut $(A, B)$ we have $v(f) \leq \text{cap}(A, B)$.

Pf.

\[
v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
\leq \sum_{e \text{ out of } A} f(e) \\
\leq \sum_{e \text{ out of } A} c(e) \\
= \text{cap}(A, B)
\]
Corollary. Let \( f \) be any flow, and let \((A, B)\) be any cut. If \( v(f) = \text{cap}(A, B) \), then \( f \) is a max flow and \((A, B)\) is a min cut.

Value of flow = 28
Cut capacity = 28  \( \Rightarrow \)  Flow value \( \leq 28 \)
Max-Flow Min-Cut Theorem

Augmenting path theorem. Flow f is a max flow iff there are no augmenting paths.

Max-flow min-cut theorem. [Ford-Fulkerson 1956] The value of the max flow is equal to the value of the min cut.

Proof strategy. We prove both simultaneously by showing the TFAE:

(i) There exists a cut \((A, B)\) such that \(v(f) = \text{cap}(A, B)\).
(ii) Flow f is a max flow.
(iii) There is no augmenting path relative to f.

(i) \(\implies\) (ii) This was the corollary to weak duality lemma.

(ii) \(\implies\) (iii) We show contrapositive.

• Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.
Proof of Max-Flow Min-Cut Theorem

(iii) \implies (i)

- Let \( f \) be a flow with no augmenting paths.
- Let \( A \) be set of vertices reachable from \( s \) in residual graph.
- By definition of \( A \), \( s \in A \).
- By definition of \( f \), \( t \notin A \).

\[
\nu(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = \sum_{e \text{ out of } A} c(e) = \text{cap}(A, B)
\]

original network
Running Time

Assumption. All capacities are integers between 1 and C.

Invariant. Every flow value $f(e)$ and every residual capacities $c_f(e)$ remains an integer throughout the algorithm.

Theorem. The algorithm terminates in at most $v(f^*) \leq mC$ iterations.

Pf. Each augmentation increase value by at least 1. ▪

Corollary. If $C = 1$, Ford-Fulkerson runs in $O(mn)$ time.

Integrality theorem. If all capacities are integers, then there exists a max flow $f$ for which every flow value $f(e)$ is an integer.

Pf. Since algorithm terminates, theorem follows from invariant. ▪
Q. Is generic Ford-Fulkerson algorithm polynomial in input size?

A. No. If max capacity is $C$, then algorithm can take $C$ iterations.

![Graphs showing iterations of the Ford-Fulkerson algorithm.](image)
Choosing Good Augmenting Paths

Use care when selecting augmenting paths.
- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal: choose augmenting paths so that:
- Can find augmenting paths efficiently.
- Few iterations.

Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]
- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.
Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

- The sum of capacities of the edges leaving $s$ is

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

- Define $\Delta$ to be the largest power of 2, such that $\Delta \leq C$

- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least $\Delta$. 
Scaling-Max-Flow(G, s, t, c) {
    foreach e ∈ E  \( f(e) \leftarrow 0 \)
    \( \Delta \leftarrow \) smallest power of 2 greater than or equal to \( C \)
    \( G_f \leftarrow \) residual graph

    while (\( \Delta \geq 1 \)) {
        \( G_f(\Delta) \leftarrow \Delta\)-residual graph
        while (there exists augmenting path P in \( G_f(\Delta) \)) {
            \( f \leftarrow \) augment(\( f, c, P \))
            update \( G_f(\Delta) \)
        }
        \( \Delta \leftarrow \Delta / 2 \)
    }
    return \( f \)
}
Assumption. All edge capacities are integers between 1 and C.

Integrality invariant. All flow and residual capacity values are integral.

Correctness. If the algorithm terminates, then f is a max flow.

Pf.
• By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$.
• Upon termination of $\Delta = 1$ phase, there are no augmenting paths. □
Lemma 1. The outer while loop repeats \(1 + \lceil \log_2 C \rceil\) times.

**Pf.** Initially \(C \leq \Delta < 2C\). \(\Delta\) decreases by a factor of 2 each iteration. □

Lemma 2. Let \(f\) be the flow at the end of a \(\Delta\)-scaling phase. Then the value of the maximum flow is at most \(v(f) + m \Delta\). ← proof on next slide

Lemma 3. There are at most \(2m\) augmentations per scaling phase.

- Let \(f\) be the flow at the end of the previous scaling phase.
- \(L2 \Rightarrow v(f^*) \leq v(f) + m (2\Delta)\).
- Each augmentation in a \(\Delta\)-phase increases \(v(f)\) by at least \(\Delta\). □

**Theorem.** The scaling max-flow algorithm finds a max flow in \(O(m \log C)\) augmentations. It can be implemented to run in \(O(m^2 \log C)\) time. □
Lemma 2. Let $f$ be the flow at the end of a $\Delta$-scaling phase. Then value of the maximum flow is at most $v(f) + m \Delta$.

Pf. (almost identical to proof of max-flow min-cut theorem)

• We show that at the end of a $\Delta$-phase, there exists a cut $(A, B)$ such that $\text{cap}(A, B) \leq v(f) + m \Delta$.

• Choose $A$ to be the set of nodes reachable from $s$ in $G_f(\Delta)$.

• By definition of $A$, $s \in A$.

• By definition of $f$, $t \notin A$.

$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$
$$\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta$$
$$= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta$$
$$\geq \text{cap}(A, B) - m\Delta$$
Edmonds-Karp Algorithm

- Neither of the algorithms we have seen so far runs in “truly” polynomial time
- Edmonds and Karp developed the first polynomial-time algorithm for flow networks.
  - Uses Ford-Fulkerson as basis
  - Modification: when finding the augmenting path, we compute the s-t path in the residual network having the smallest number of edges
    - Note that this can be accomplished by using BFS to compute the augmenting path
  - It can be shown that the total number of augmenting steps using this method is $O(nm)$ (Proof in CLRS)
  - Overall runtime = $O(nm^2)$
Other Algorithms

- KT discusses pre-flow push algorithm
  - Number of variants of this algorithm
  - Simplest version runs in $O(n^3)$ time

- Another quite sophisticated algorithm runs in time $O(\min(n^{2/3}, m^{1/2}) m \log n \log U)$, where $U$ is an upper bound on the largest capacity.
Applications of Max-Flow

Huge number of applications

• Bipartite Matching
• Perfect Matching
• Disjoint Paths
Matching.

- Input: undirected graph $G = (V, E)$.
- $M \subseteq E$ is a matching if each node appears in at most one edge in $M$.
- Max matching: find a max cardinality matching.
Bipartite matching.

- **Input:** undirected, bipartite graph $G = (L \cup R, E)$.
- $M \subseteq E$ is a matching if each node appears in at most one edge in $M$.
- **Max matching:** find a max cardinality matching.

- Matching: $1-2', 3-1', 4-5'$
Bipartite matching.

- Input: undirected, bipartite graph $G = (L \cup R, E)$.
- $M \subseteq E$ is a matching if each node appears in at most one edge in $M$.
- Max matching: find a max cardinality matching.

Max matching:
1-1', 2-2', 3-3', 4-4'

Diagram:

- Nodes are divided into two sets $L$ and $R$.
- Edges connect nodes from $L$ to $R$.
Max flow formulation.

- Create digraph $G' = (L \cup R \cup \{s, t\}, E')$.
- Direct all edges from $L$ to $R$, and assign infinite (or unit) capacity.
- Add source $s$, and unit capacity edges from $s$ to each node in $L$.
- Add sink $t$, and unit capacity edges from each node in $R$ to $t$. 

Bipartite Matching

![Graph G']
Theorem. Max cardinality matching in G = value of max flow in G'.

Pf. ≤
- Given max matching M of cardinality k.
- Consider flow f that sends 1 unit along each of k paths.
- f is a flow, and has cardinality k. □
Bipartite Matching: Proof of Correctness

Theorem. Max cardinality matching in $G = \text{value of max flow in } G'$.

Pf. $\geq$

- Let $f$ be a max flow in $G'$ of value $k$.
- Integrality theorem $\Rightarrow k$ is integral and can assume $f$ is 0-1.
- Consider $M = \text{set of edges from L to R with } f(e) = 1$.
  - each node in L and R participates in at most one edge in $M$
  - $|M| = k$: consider cut $(L \cup s, R \cup t)$

\[ \begin{array}{c}
1 & \rightarrow & 1' \\
\downarrow & & \downarrow \\
1 & \rightarrow & 1' \\
\downarrow & & \downarrow \\
2 & \rightarrow & 2' \\
\downarrow & & \downarrow \\
2 & \rightarrow & 2' \\
\downarrow & & \downarrow \\
3 & \rightarrow & 3' \\
\downarrow & & \downarrow \\
3 & \rightarrow & 3' \\
\downarrow & & \downarrow \\
4 & \rightarrow & 4' \\
\downarrow & & \downarrow \\
4 & \rightarrow & 4' \\
\downarrow & & \downarrow \\
5 & \rightarrow & 5' \\
\downarrow & & \downarrow \\
5 & \rightarrow & 5' \\
\end{array} \]

\[ \begin{array}{c}
1 & \rightarrow & 1' \\
\downarrow & & \downarrow \\
2 & \rightarrow & 2' \\
\downarrow & & \downarrow \\
3 & \rightarrow & 3' \\
\downarrow & & \downarrow \\
4 & \rightarrow & 4' \\
\downarrow & & \downarrow \\
5 & \rightarrow & 5' \\
\end{array} \]
Def. A matching $M \subseteq E$ is **perfect** if each node appears in exactly one edge in $M$.

Q. When does a bipartite graph have a perfect matching?

**Structure of bipartite graphs with perfect matchings.**
- Clearly we must have $|L| = |R|$.
- What other conditions are necessary?
- What conditions are sufficient?
**Perfect Matching**

**Notation.** Let S be a subset of nodes, and let N(S) be the set of nodes adjacent to nodes in S.

**Observation.** If a bipartite graph \( G = (L \cup R, E) \), has a perfect matching, then \( |N(S)| \geq |S| \) for all subsets \( S \subseteq L \).

**Pf.** Each node in S has to be matched to a different node in N(S).

![Diagram of a bipartite graph](image)

**No perfect matching:**

\( S = \{2, 4, 5\} \)

\( N(S) = \{2', 5'\} \).
Which max flow algorithm to use for bipartite matching?

- Generic augmenting path: $O(m \text{ val}(f^*) ) = O(mn)$.
- Capacity scaling: $O(m^2 \log C ) = O(m^2)$.
- Shortest augmenting path: $O(m n^{1/2})$.

Non-bipartite matching.

- Structure of non-bipartite graphs is more complicated, but well-understood. [Tutte-Berge, Edmonds-Galai]
- Blossom algorithm: $O(n^4)$. [Edmonds 1965]
- Best known: $O(m n^{1/2})$. [Micali-Vazirani 1980]
Disjoint path problem. Given a digraph $G = (V, E)$ and two nodes $s$ and $t$, find the max number of edge-disjoint s-t paths.

**Def.** Two paths are edge-disjoint if they have no edge in common.

**Ex:** communication networks.
Disjoint path problem. Given a digraph $G = (V, E)$ and two nodes $s$ and $t$, find the max number of edge-disjoint $s$-$t$ paths.

Def. Two paths are edge-disjoint if they have no edge in common.

Ex: communication networks.
Max flow formulation: assign unit capacity to every edge.

Theorem. Max number edge-disjoint s-t paths equals max flow value.

Pf. ≤

- Suppose there are k edge-disjoint paths $P_1, \ldots, P_k$.
- Set $f(e) = 1$ if $e$ participates in some path $P_i$; else set $f(e) = 0$.
- Since paths are edge-disjoint, $f$ is a flow of value $k$. ▪
Max flow formulation: assign unit capacity to every edge.

**Theorem.** Max number edge-disjoint s-t paths equals max flow value.

**Pf.**

- Suppose max flow value is $k$.
- Integrality theorem $\Rightarrow$ there exists 0-1 flow $f$ of value $k$.
- Consider edge $(s, u)$ with $f(s, u) = 1$.
  - by conservation, there exists an edge $(u, v)$ with $f(u, v) = 1$
  - continue until reach $t$, always choosing a new edge
- Produces $k$ (not necessarily simple) edge-disjoint paths.

\[\text{can eliminate cycles to get simple paths if desired}\]
Network connectivity. Given a digraph $G = (V, E)$ and two nodes $s$ and $t$, find the minimum number of edges whose removal disconnects $t$ from $s$.

**Def.** A set of edges $F \subseteq E$ disconnects $t$ from $s$ if all $s$-$t$ paths uses at least one edge in $F$. 
Theorem. [Menger 1927] The max number of edge-disjoint s-t paths is equal to the min number of edges whose removal disconnects t from s.

Pf. ≥
- Suppose max number of edge-disjoint paths is k.
- Then max flow value is k.
- Max-flow min-cut ⇒ cut (A, B) of capacity k.
- Let F be set of edges going from A to B.
- |F| = k and disconnects t from s. □
Theorem. [Menger 1927] The max number of edge-disjoint s-t paths is equal to the min number of edges whose removal disconnects t from s.

Pf. ≤

• Suppose the removal of $F \subseteq E$ disconnects t from s, and $|F| = k$.
• All s-t paths use at least one edge of F. Hence, the number of edge-disjoint paths is at most k. □
Next Time

Extensions of Network Flow