Lecture in a minute

Maximum flow.
"Greedy" augment path...
Except reverse old decisions..
Reverse residual capacities.
Optimality?
No augmenting path \( \Rightarrow \) 
\( s - t \) cut size = flow value.
Find flow and \( s - t \) cut with equal value!

Maximum Matching.
\( G = (V,E) \), find subset of one-to-one matches.
Reduction to max flow.
Augmenting Alternating Path
Algorithm \( \equiv \) Simplex.

S-T cut.

An \( s - t \) cut is a partition of \( V \) into \( S \) and \( T \) where \( s \in S \) and \( t \in T \). Its capacity is the total capacity of edges from \( S \) to \( T \).

Do you know the definition?

Find Flow: \( f \)
1. \( 0 \leq f_e \leq c_e \). "Capacity constraints."
2. If \( u \) is not \( s \) or \( t \)
\( \sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw} \).
3. maximize \( \sum_{s \in S} f_{su} \).

Valid or Invalid?

Algorithms.

FindFlow: \( f \)
1. \( 0 \leq f_e \leq c_e \). "Capacity constraints."
2. \( \sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw} \).
3. maximize \( \sum_{s \in S} f_{su} \).

Linear program!
Variables \( f_e \), linear constraints, linear optimization function.
Cool!
Note...
Integer? (Given integer capacities.)
Yes. There is an integer vertex solution!
Constraint matrix has every subdeterminant being 1, 0, \(-1\).
Vertex solution to linear program must be integral!
Ford-Fulkerson.

“Simplex” method.
Find s to t path with remaining capacity.
Add to flow variables along path.
Update remaining capacity.
Repeat.

Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path. Or reduce flow on reverse edge.
Update remaining capacity.
Reduce $r_e = c_e - f_e$
and add reverse $r_{ew} = f_{uw}$
Repeat.

Feasibility.

1. Capacity Constraints: $0 \leq f_e \leq c_e$.
Only increase flow to $c_e$.
Or use reverse arcs decrease to 0.
Flow values to be between 0 and $c_e$.

2. Conservation Constraints:
“flow into $v$” = “flow out of $v$” (if not s or t.)
Algorithm adds flow, say $f$, to path from s to t.
Each internal node has $f$ in, and $f$ out.

Optimality: upper bound.

$s-t$ Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.

Lemma: Capacity of any s−t cut is an upper bound on the flow.
$C(S, T) = \text{sum of capacities of all arcs from S to T}$
$\sum_{(u,v) \in E} c_{uv}$
For valid flow:
Flow out of $(S) = \text{Flow out of s}$.
$\sum_{v \in S} \sum_{(u,v) \in E} f(v, u) - \sum_{u \in S} \sum_{(u,v) \in E} f(u, v) = \sum_{(u,v) \in E} c_{uv}$
Flow into $(T) = \text{Flow into t}$.
For any valid flow, $f: E \to \mathbb{Z}^+$, the flow out of $S$ (into $T$)
$\sum_{u \in S} f(u) \leq \sum_{(u,v) \in E} c_{uv} - \sum_{(t,v) \in E} f(t, v) = C(S, T)$.
$\rightarrow$ The value of any valid flow is at most $C(S, T)$!
Optimality: max flow = min cut.
At termination of augmenting path algorithm.
No path with residual capacity!
Depth first search only starting at \( s \) does not reach \( t \).

\[ S = \text{reachable nodes.} \]
\[ \text{No arc with positive residual capacity leaving } S \]
\[ \implies \text{All arcs leaving } S \text{ are full.} \]
\[ \implies \text{No arcs into } S \text{ have flow.} \]

Total flow leaving \( S \) is \( C(S, T) \).

Valid flow \( \implies \) all that flow from source.
Value of flow equals value of \( C(S, T) \), and Optimal is \( \leq C(S, T) \).

"any flow" \( \leq " \text{any cut} \) and this flow = this cut.
\[ \implies \text{Maximum flow and minimum } s-t \text{ cut!} \]

Celebrated max flow -minimum cut theorem.

Theorem: In any flow network, the maximum \( s-t \) flow is equal to the minimum cut.
Celebrate!

Back to business: Algorithm Terminates?

It will!!
Flow keeps increasing.
How long?
One more unit every step!
O(mF) time where \( F \) is size of flow.

Efficiency.

Edmonds-Karp

Augment along shortest path.
Breadth first search!
O(\( |V| |E| \)) augmentations.
Analysis idea.
d(\( v \)) is distance to sink.
Only route flow on \( (u, v) \) if \( d(u) \geq d(v) \).
Only reverse flow on \( (u, v) \) if \( d(v) \leq d(u) \).
Maximum \( d(v) \) is \( |V| \).
Distances only go up. (To prove!)
Every O(\( |E| \)) time augment removes edge “at” a distance.
O(\( |V| |E| \)) removals.
O(\( |V| |E|^2 \)) time.

Bipartite Matching

Given a bipartite graph: \( B = (L, R, E) \) where \( E \subseteq L \times R \).
Katya
Shelby
Chen
Jacque
Ivan
Cindy
Ollie
Find largest subset of edges (“matches”) which are one to one.
Bipartite Matching
Algorithm by “Reduction.”:
From matching problem produce flow problem.
From flow solution produce matching solution.

Maximum Matching Problem.
Given a bipartite graph, \( G = (U, V, E) \), find a maximum sized matching.
Key Idea: Augmenting Alternating Paths.
Example:

Simplex Algorithm
\[
\begin{align*}
\text{max} & \quad c \cdot x, \\
Ax & \leq b, \\
x & \geq 0
\end{align*}
\]
Start at feasible point where \( n \) equations are satisfied.
E.g., \( x = 0 \).
This is a point.
Another view: intersection of \( n \) hyperplanes.
Drop one equation:
Points on line satisfy \( n - 1 \) ind. equations.
Intersection of \( n - 1 \) hyperplanes.
Move in direction that increases objective.
Until new tight constraint.
No direction increases objective.

Hyperplane View
\( x + y + z \leq 1 \)
On one side of hyperplane.
Normal to hyperplane? \( (1, 1, 1) \).
Why?
\[
\begin{align*}
(a, b, c) \text{ where } & \quad a + b + c = 1, \\
(a', b', c') \text{ where } & \quad a' + b' + c' = 1, \\
(a + 1, b + 1, c + 1) - (a, b, c) & = (1, 1, 1), \\
(a' - a, b' - b, c' - c) \cdot (1, 1, 1) & = (a' + b' + c' - (a + b + c)) = 0.
\end{align*}
\]
Normal to \( mx + ny + pz = C \) ? \( (m, n, p) \)
Points in hyperplane are related by nullspace of row.
Maximum Matching and Simplex.

\[
\begin{align*}
\text{max } & x + y + z \\
\text{s.t. } & x \leq 1 \\
& x + z \leq 1 \quad a = 1 \\
& z + y \leq 1 \quad b = 1 \\
& x \geq 0 \\
& y \geq 0 \\
& z \geq 0 \quad c = 1
\end{align*}
\]

Sum: \(x + z + y\).

Augmenting Path. Via Gaussian Elimination!

Lecture in a minute

Maximum Flow.

"Greedy" augment path...

Except reverse old decisions ..

Reverse residual capacities.

Optimality?

No augmenting path \(\implies\)

Find flow and \(s-t\) cut with equal value!

Maximum Matching.

\(G = (V,E)\), find subset of one-to-one matches.

Reduction to max flow.

Augmenting Alternating Path

Algorithm \(\equiv\) Simplex.