Max Flow Recap

**Max-Flow Problem, Min Cut Problem**
- Definition of a s-t flow \( f(e) \) and a s-t cut \((A,B)\)
- Value of a flow \( f \)
- Capacity of a s-t cut \((A,B)\)

**Weak Duality Lemma:** For any flow \( f \) and s-t cut \( A,B \) we have \( v(f) \leq cap(A, B) \) (i.e., capacity of minimum cut is upper bound on max-flow)

**Finding a Max-Flow:**
- **Greedy algorithm fails!**
- Residual Graph
- **Ford-Fulkerson Algorithm**
  - Repeatedly find augmenting path in residual graph
  - Proof of Correctness
  - **Max-Flow Min-Cut Equivalence**
7.3 Choosing Good Augmenting Paths


Ford-Fulkerson: Exponential Number of Augmentations

Q. Is generic Ford-Fulkerson algorithm polynomial in input size?

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

\[
\begin{align*}
&
\text{Q. Is generic Ford-Fulkerson algorithm polynomial in input size?} \\
&
\text{A. No. If max capacity is } C, \text{ then algorithm can take } C \text{ iterations.}
\end{align*}
\]
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.
- Don’t worry about finding exact highest bottleneck path.
- Maintain scaling parameter $\Delta$.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least $\Delta$. 

![Diagram](image)
Capacity Scaling

Scaling-Max-Flow(G, s, t, c) {
    foreach e ∈ E f(e) ← 0
    Δ ← smallest power of 2 greater than or equal to C
    G_\Delta_f ← residual graph

    while (Δ ≥ 1) {
        G_\Delta_f(Δ) ← Δ-residual graph
        while (there exists augmenting path P in G_\Delta_f(Δ)) {
            f ← augment(f, c, P)
            update G_\Delta_f(Δ)
        }
        Δ ← Δ / 2
    }
    return f
}
Capacity Scaling: Correctness

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. \( \blacksquare \)
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 $f$, $t \notin A$.

\[
v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta \\
= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta \\
\geq \text{cap}(A,B) - m\Delta
\]
Dinic’s Max Flow Min-Cut Algorithm

Use Breadth First Search to Compute Level Graph

$G$: 

Discard cross-layer edges
Dinic’s Max Flow Min-Cut Algorithm

Use Breadth First Search to Compute Level Graph

\[ G: \]

\[ G_L: \]

Discard cross-layer edges
Dinic’s Max Flow Min-Cut Algorithm

Use Breadth First Search to Compute Level Graph

\[ G: \]

Level 0

Level 1

Level 2

Level 3

\[ G_L \]

Discard cross-layer edges
Find Blocking Flow
Dinic’s Max Flow Min-Cut Algorithm

Create Residual Graph $G_f$

$G_f$: [Diagram of the residual graph with capacities shown]

$G_L$: [Diagram of the residual graph with minimum flow paths shown]

Total Flow: 14
Dinic’s Max Flow Min-Cut Algorithm

Run BFS on $G_f$ to create level graph $G_{f,L}$

$G_f$:

Remark: Number of levels increased. This is not a coincidence!
Dinic’s Max Flow Min-Cut Algorithm

Run BFS on $G_f$ to create level graph $G_{f,L}$

$G_f$: Level 0

Level 1

Level 2

Level 3

Level 4

s

3

2

4

5

t

5

2

7

1

6

4

1

4

5

2

7

1

6

4

1

4

5

2

7

1

6

4
Dinic’s Max Flow Min-Cut Algorithm

Run BFS on $G_f$ to create level graph $G_{f,L}$

$G_f$:

$G_{f,L}$
Dinic’s Max Flow Min-Cut Algorithm

Run BFS on $G_f$ to create level graph $G_{f,L}$

$G_f$: Level 0

Level 1

Level 2

Level 3

Level 4

$G_{f,L}$

Blocking Flow for level graph $G_{f,L}$

Total Extra Flow: 5
Dinic’s Max Flow Min-Cut Algorithm

New Residual Graph $G_f$

$G_f$:  

$G_{f,L}$:  

Total Extra Flow: 5
Dinic’s Max Flow Min-Cut Algorithm

New Residual Graph $G_f$

$G_f$:  

Breadth First Search: Yields minimum $s$-$t$ cut! $\Rightarrow$ We are done!
Finding a Blocking Flow in $G_{f,L}$

**Definition:** $C_{f,L}(e)$ denotes the capacity of the edge $e$ in $G_{f,L}$

**Definition:** Given an augmenting flow $f'$ for the level graph $G_{f,L}$ and a path $P$ in $G_{f,L}$ we define $B(P, f') = \min_{e \in P} \{ C_{f,L}(e) - f'(e) \}$

**FindBlockingFlow($G_{f,L}$)**

- **Initialize:**
  - RemCap($e$) = $C_{f,L}(e)$ and $f'(e) = 0$ for each edge $e$ in $G_{f,L}$
- **While** there is a path $P$ with $B(P, f') > 0$
  - Update $f'(e) = f'(e) + B(P, f')$ for each edge $e \in P$
  - Update RemCap($e$) = RemCap($e$) − $B(P, f')$ for each edge $e \in P$

**Analysis:** Each iteration of the “while” loop eliminates an edge

**Implication:** Terminates after $O(m)$ iterations of while loop.
Finding a Blocking Flow in $G_{f,L}$

**Definition:** $C_{f,L}(e)$ denotes the capacity of the edge $e$ in $G_{f,L}$

**Definition:** Given an augmenting flow $f'$ for the level graph $G_{f,L}$ and a path $P$ in $G_{f,L}$ we define $B(P, f') = \min_{e \in P} \{ C_{f,L}(e) - f'(e) \}$

**FindBlockingFlow($G_{f,L}$)**
- **Initialize:**
  - RemCap$\left( e \right) = C_{f,L}(e)$ and $f'(e) = 0$ for each edge $e$ in $G_{f,L}$
- **While** there is a path $P$ with $B(P, f') > 0$
  - Update $f'(e) = f'(e) + B(P, f')$ for each edge $e \in P$
  - Update RemCap$\left( e \right) = \text{RemCap}(e) - B(P, f')$ for each edge $e \in P$

**Analysis:** Each iteration of the “while” loop eliminates an edge

**Implication:** Terminates after $O(m)$ iterations of while loop.

**Naïve Running Time Analysis:** $O(m(m+n))$
Finding a Blocking Flow in $G_{f,L}$

**Definition:** We let $C_{f,L}(e)$ denote the capacity of an edge $e$ in $G_{f,L}$

**Definition:** Given an augmenting flow $f'$ for $G_{f,L}$ and a s-t path $P$ we define $B(P) = \min_{e \in P} C_{f,L}(e)$

**FindBlockingFlow($G_{f,L}$)**
- Initialize $\text{RemCap}(e) = C_{f,L}(e)$
- While there exists a path $P$ with $B(P) > 0$
  - Set $f'(e) = f'(e) + B(P)$ for each edge $e \in P$
  - Set $\text{RemCap}(e) = \text{RemCap}(e) - B(P)$ for each edge $e \in P$

**Analysis:** Each iteration of while loop “eliminates” at least one edge.

**Implication:** Terminates after at most $m$ rounds.

**Naïve Running Time:** $O((m+n)m)$

**Amortization:** Can enumerate paths in amortized time $O(n)$ per path
Dinic’s Algorithm

1. Start with empty flow $f$
2. Construct $G_f$
3. Repeat until $s$ and $t$ are disconnected (no augmenting path)
   1. (Level Graph) Run BFS on $G_f$ to build $G_{f,L}$
   2. (Blocking Flow) Find blocking flow $f'$ in $G_{f,L}$
   3. (Augment) Let $f = f + f'$ and Construct $G_f$
4. Output $f$

Analysis:
**Claim:** Each time we iterate the loop we increase the depth of $G_f$

**Implication:** Must terminate in at most $n$ iterations!

**Time Per Iteration:** $O(nm)$ to find blocking flow $f'$

**Total Time:** $O(n^2m)$
Dinic’s Algorithm: Correctness and Running Time

Correctness follows directly from Augmenting Path Theorem.

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

Running Time Analysis: Let $f_i$ denote residual graph after iteration $i$ ($G_{f_0} = G$)

Definition: $\text{depth}(G_{f_i}) = \text{length of the shortest directed path from } s \text{ to } t$.

Key Claim: $\text{depth}(G_{f_{i+1}}) > \text{depth}(G_{f_i})$ (depth always increases)
Dinic’s Algorithm: Correctness and Running Time

**Running Time Analysis:** Let \( f_i \) denote residual graph after iteration \( i \) (\( G_{f_0} = G \))

**Definition:** \( \text{depth}(G_{f_l}) = \text{length of the shortest directed path from } s \text{ to } t \).

**Key Claim:** \( \text{depth}(G_{f_{i+1}}) > \text{depth}(G_{f_i}) \) (depth always increases)

**Proof:** Suppose (for contradiction) that \( \text{depth}(G_{f_{i+1}}) \leq \text{depth}(G_{f_i}) \).

- Then \( G_{f_{i+1}} \) contains an \( s-t \) path of length \( \leq \text{depth}(G_{f_i}) \).
- This path corresponds to an augmenting path for the flow \( f' = f_{i+1} - f_i \) in \( G_{f_l} \).
- But since the augmenting path has length \( \text{depth}(G_{f_i}) \) it is also an augmenting path in the level graph \( G_{f_i L} \).
- This contradicts the claim that \( f' \) is a blocking flow in \( G_{f_i L} \)!
Dinic's Algorithm: Correctness and Running Time

**Running Time Analysis**: Let $f_i$ denote residual graph after iteration $i$ ($G_{f_0} = G$)

**Definition**: $\text{depth}(G_{f_i}) =$ length of the shortest directed path from $s$ to $t$.

**Key Claim**: $\text{depth}(G_{f_{i+1}}) > \text{depth}(G_{f_i})$ (depth always increases)

**Implication**: \#iterations is at most $n$

Time to Compute Blocking Flow in Level Graph: $O(mn)$
- Using special data-structure called dynamic trees $O(m \log n)$

Total Time: $O(mn \log n)$ with dynamic trees or $O(mn^2)$ without.
7.7 Extensions to Max Flow
Circulation with Demands

Circulation with demands.
- Directed graph $G = (V, E)$.
- Edge capacities $c(e), e \in E$.
- Node supply and demands $d(v), v \in V$.

Def. A circulation is a function that satisfies:
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

Circulation problem: given $(V, E, c, d)$, does there exist a circulation?
Circulation with Demands

**Necessary condition:** \( \text{sum of supplies} = \text{sum of demands} \).

\[
\sum_{v : d(v) > 0} d(v) = \sum_{v : d(v) < 0} -d(v) =: D
\]

**Pf.** Sum conservation constraints for every demand node \( v \).
Circulation with Demands

Max flow formulation.

$G$: 

- Supply: -6
- Demand: 11
Max flow formulation.

- Add new source $s$ and sink $t$.
- For each $v$ with $d(v) < 0$, add edge $(s, v)$ with capacity $-d(v)$.
- For each $v$ with $d(v) > 0$, add edge $(v, t)$ with capacity $d(v)$.
- Claim: $G$ has circulation iff $G'$ has max flow of value $D$. 

![Graph $G'$ showing the circulation problem with supplies, demands, and edges labeled with capacities.](image)
Circulation with Demands

Integrity theorem. If all capacities and demands are integers, and there exists a circulation, then there exists one that is integer-valued.

Pf. Follows from max flow formulation and integrity theorem for max flow.

Characterization. Given \((V, E, c, d)\), there does not exists a circulation iff there exists a node partition \((A, B)\) such that \(\sum_{v \in B} d_v > \text{cap}(A, B)\).

Pf idea. Look at min cut in \(G'\).
Circulation with Demands and Lower Bounds

Feasible circulation.
- Directed graph $G = (V, E)$.
- Edge capacities $c(e)$ and lower bounds $\ell(e), e \in E$.
- Node supply and demands $d(v), v \in V$.

Def. A circulation is a function that satisfies:
- For each $e \in E$: $\ell(e) \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

Circulation problem with lower bounds. Given $(V, E, \ell, c, d)$, does there exist a circulation?
Idea. Model lower bounds with demands.
- Send $\ell(e)$ units of flow along edge $e$.
- Update demands of both endpoints.

Theorem. There exists a circulation in $G$ iff there exists a circulation in $G'$. If all demands, capacities, and lower bounds in $G$ are integers, then there is a circulation in $G$ that is integer-valued.

Pf sketch. $f(e)$ is a circulation in $G$ iff $f'(e) = f(e) - \ell(e)$ is a circulation in $G'$. 
7.8 Survey Design
Survey Design

Survey design.

- Design survey asking $n_1$ consumers about $n_2$ products.
- Can only survey consumer $i$ about product $j$ if they own it.
- Ask consumer $i$ between $c_i$ and $c_i'$ questions.
- Ask between $p_j$ and $p_j'$ consumers about product $j$.

Goal. Design a survey that meets these specs, if possible.

Bipartite perfect matching. Special case when $c_i = c_i' = p_i = p_i' = 1$. 
Algorithm. Formulate as a circulation problem with lower bounds.
- Include an edge \((i, j)\) if consumer \(j\) owns product \(i\).
- Integer circulation \(\iff\) feasible survey design.