CSE 421
Polytime Max Flow
Linear Programming
Shayan Oveis Gharan
Summary

• If a problem is NP-complete it does not mean that all instances are hard, e.g., Vertex-cover has a polynomial-time algorithm in trees

• We learned the crucial idea of polynomial-time reduction. This can be even used in algorithm design, e.g., we know how to solve max-flow so we reduce image segmentation to max-flow

• NP-Complete problems are the hardest problem in NP

• NP-hard problems may not necessarily belong to NP.

• Polynomial-time reductions are transitive relations
Q. Is generic Ford-Fulkerson algorithm polynomial in input size?

A. No. If max capacity is $C$, then algorithm can take $C$ iterations.
Choosing Good Augmenting Paths

Use care when selecting augmenting paths.
- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.

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 $\Delta$. 

$$
\begin{array}{c}
\text{Capacity Scaling} \\
\text{Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.} \\
\begin{itemize}
  \item Don't worry about finding exact highest bottleneck path.
  \item Maintain scaling parameter $\Delta$.
  \item Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least $\Delta$.
\end{itemize}
\end{array}
$$
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_f ← residual graph

    while (Δ ≥ 1) {
        G_f(Δ) ← Δ-residual graph
        while (there exists augmenting path P in G_f(Δ)) {
            f ← augment(f, c, P)
            update G_f(Δ)
        }
        Δ ← Δ / 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 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 find 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 $f$, $t \notin A$.

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

\[
\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta
\]

\[
= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta
\]

\[
\geq \text{cap}(A, B) - m\Delta
\]
Lemma 1. The outer while loop repeats $1 + \lceil \log_2 C \rceil$ times.

Proof. Initially $C \leq \Delta < 2C$. $\Delta$ decreases by a factor of 2 each iteration. ▪

Lemma 2. 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, when $m > n$. ▪
Linear Programming
Linear System of Equations

In high school we learn Gaussian elimination algorithm to solve a system of linear equations

\[
\begin{align*}
x_1 + x_3 &= 7 \\
2x_2 + x_1 &= 5 \\
3x_1 + 7x_2 - x_3 &= 1
\end{align*}
\]

We set \( x_3 = 7 - x_1 \) and we substitute in the following equations.

Then we substitute \( x_2 = \frac{5-x_1}{2} \) in to the third equations.
The third equational uniquely defines \( x_1 \).
Linear Programming

Optimize a linear function subject to linear inequalities

\[ \text{max } 3x_1 + 4x_3 \]
\[ \text{s.t., } x_1 + x_2 \leq 5 \]
\[ x_3 - x_1 = 4 \]
\[ x_3 - x_2 \geq -5 \]
\[ x_1, x_2, x_3 \geq 0 \]

• We can have inequalities,
• We can have a linear objective functions
Applications of Linear Programming

Generalizes: $Ax=b$, 2-person zero-sum games, shortest path, max-flow, matching, multicommodity flow, MST, min weighted arborescence, …

Why significant?
- We can solve linear programming in polynomial time.
- Useful for approximation algorithms
- We can model many practical problems with a linear model and solve it with linear programming

Linear Programming in Practice:
- There are very fast implementations: IBM CPLEX, Gorubi in Python, CVX in Matlab, ….
- CPLEX can solve LPs with millions of variables/constraints in minutes
Example 1: Diet Problem

Suppose you want to schedule a diet for yourself. There are four categories of food: veggies, meat, fruits, and dairy. Each category has its own (p)rice, (c)alory, and (h)appiness per pound:

<table>
<thead>
<tr>
<th></th>
<th>veggies</th>
<th>meat</th>
<th>fruits</th>
<th>dairy</th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>$p_v$</td>
<td>$p_m$</td>
<td>$p_f$</td>
<td>$p_d$</td>
</tr>
<tr>
<td>calorie</td>
<td>$c_v$</td>
<td>$c_m$</td>
<td>$c_f$</td>
<td>$c_d$</td>
</tr>
<tr>
<td>happiness</td>
<td>$h_v$</td>
<td>$h_m$</td>
<td>$h_f$</td>
<td>$h_d$</td>
</tr>
</tbody>
</table>

**Linear Modeling:** Consider a linear model: If we eat 0.5lb of meat, 0.2lb of fruits we will be $0.5 \ h_m + 0.2 \ h_f$ happy

- You should eat 1500 calories to be healthy
- You can spend 20 dollars a day on food.

**Goal:** Maximize happiness?
Diet Problem by LP

- You should eat 1500 calories to be healthy
- You can spend 20 dollars a day on food.

**Goal:** Maximize happiness?

<table>
<thead>
<tr>
<th></th>
<th>veggies</th>
<th>meat</th>
<th>fruits</th>
<th>dairy</th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>$p_v$</td>
<td>$p_m$</td>
<td>$p_f$</td>
<td>$p_d$</td>
</tr>
<tr>
<td>calorie</td>
<td>$c_v$</td>
<td>$c_m$</td>
<td>$c_f$</td>
<td>$c_d$</td>
</tr>
<tr>
<td>happiness</td>
<td>$h_v$</td>
<td>$h_m$</td>
<td>$h_f$</td>
<td>$h_d$</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{max} & \quad x_v h_v + x_m h_m + x_f h_f + x_d h_d \\
\text{s.t.} & \quad x_v p_v + x_m p_m + x_f p_f + x_d p_d \leq 20 \\
& \quad x_v c_v + x_m c_m + x_f c_f + x_d c_d \leq 1500 \\
& \quad x_v, x_m, x_f, x_d \geq 0 \\
\end{align*}
\]

#pounds of veggies, meat, fruits, dairy to eat per day
How to Design an LP?

• Define the set of variables

• Put constraints on your variables,
  • should they be nonnegative?

• Write down the constraints
  • If a constraint is not linear try to approximate it with a linear constraint

• Write down the objective function
  • If it is not linear approximation with a linear function

• Decide if it is a minimize/maximization problem
Example 2: Max Flow

Define the set of variables
• For every edge $e$ let $x_e$ be the flow on the edge $e$

Put constraints on your variables
• $x_e \geq 0$ for all edge $e$ (The flow is nonnegative)

Write down the constraints
• $x_e \leq c(e)$ for every edge $e$, (Capacity constraints)
• $\sum_{e \text{ out of } v} x_e = \sum_{e \text{ into } v} x_e \forall v \neq s, t$ (Conservation constraints)

Write down the objective function
• $\sum_{e \text{ out of } s} x_e$

Decide if it is a minimize/maximization problem
• $\text{max}$
Example 2: Max Flow

\[
\begin{align*}
\text{max} & \quad \sum_{e \text{ out of } s} x_e \\
\text{s.t.} & \quad \sum_{e \text{ out of } v} x_e = \sum_{e \text{ in to } v} x_e \quad \forall v \neq s, t \\
& \quad x_e \leq c(e) \quad \forall e \\
& \quad x_e \geq 0 \quad \forall e
\end{align*}
\]

Q: Do we get exactly the same properties as Ford Fulkerson?
A: Not necessarily, the max-flow may not be integral
Example 3: Min Cost Max Flow

Suppose we can route 100 gallons of water from $s$ to $t$. But for every pipe edge $e$ we have to pay $p(e)$ for each gallon of water that we send through $e$.

**Goal**: Send 100 gallons of water from $s$ to $t$ with minimum possible cost

\[
\begin{align*}
\min & \quad \sum_{e \in E} p(e) \cdot x_e \\
\text{s.t.} & \quad \sum_{e \text{ out of } v} x_e = \sum_{e \text{ into } v} x_e \quad \forall v \neq s, t \\
& \quad \sum_{e \text{ out of } s} x_e = 100 \\
& \quad x_e \leq c(e) \quad \forall e \\
& \quad x_e \geq 0 \quad \forall e
\end{align*}
\]
Summary (Linear Programming)

• Linear programming is one of the biggest advances in 20th century

• It is being used in many areas of science: Mechanics, Physics, Operations Research, and in CS: AI, Machine Learning, Theory, …

• Almost all problems that we talked can be solved with LPs, Why not use LPs?
  • Combinatorial algorithms are typically faster
  • They exhibit a better understanding of worst case instances of a problem
  • They give certain structural properties, e.g., Integrality of Max-flow when capacities are integral

• There is rich theory of LP-duality which generalizes max-flow min-cut theorem
What is next?

- **CSE 431 (Complexity Course)**
  - How to prove lower bounds on algorithms?

- **CSE 521 (Graduate Algorithms Course)**
  - How to design streaming algorithms?
  - How to design algorithms for high dimensional data?
  - How to use matrices/eigenvalues/eigenvectors to design algorithms
  - How to use LPs to design algorithms?

- **CSE 525 (Graduate Randomized Algorithms Course)**
  - How to use randomization to design algorithms?
  - How to use Markov Chains to design algorithms?