Suppose you want to solve some problem A.

- If you're lucky, A has a polynomial time algorithm (either direct or by reducing to ShortestPath, SCC, FFT, Max Flow, LP).
- Otherwise, you can try to prove that A is NP-hard, by reducing some NPC problem to A (e.g., 3SAT, IS, Red Cycle, ILP, and more and more...).
Proving that Problem A is NP-Hard is not the end of the story. What to do next?

- change the problem...
  - maybe notice some structure that makes the problem easy

For example: Horn SAT is easy
  2SAT is easy.
  2SAT with bounded occurrence is hard.

- Negotiation:
  - backtracking
  - branch & bound
  - use correct but inefficient algorithm.

- Use efficient (poly-time algorithms) but relax correctness
  (settle for near optimal solutions).

Approximation Algorithms.
- Find Heuristics and perform local search to find optimal solution faster than brute-force.
- Reduce the problem to a well-studied problem (e.g. SAT, ILP) and use off-the-shelf algorithms for the well-studied problem.
- Fixed Parameter Tractability (FPT): Algorithms that are efficient provided that some natural parameter is small.
Backtracking

Important Example: DPLL algorithm for SAT.

Given a CNF formula:
\[ \phi = (\omega \lor x \lor y) \land (\overline{x} \lor y) \land (\omega \lor \overline{y}) \]
\[ \phi \mid _{\omega \leftarrow 1} = (\overline{x} \lor y) \land (\overline{y}). \]

\[ \text{DPLL}(\phi): \]
- Assign 1 to \( \omega \) which simplifies.

edge cases \{ 
1. If \( \exists \) empty clause in \( \phi \) \( \rightarrow \) return FALSE (unsat)
2. If no more clauses in \( \phi \) \( \rightarrow \) return TRUE (sat)
\}

one option \{ 
3. If a variable \( v \) appears only positively, then set \( v \leftarrow 1 \) \& return DPLL(\( \phi \mid _{v \leftarrow 1} \))
4. If a variable \( v \) \( \neg \) negatively, \( \neg \) \( v \leftarrow 0 \) \& return DPLL(\( \phi \mid _{v \leftarrow 0} \))
5. If \( \phi \) contains a clause with one literal, \( v \) or \( \overline{v} \), then set \( v \) to value \( 6 \) in order to satisfy that literal \& return DPLL(\( \phi \mid _{v \leftarrow 0} \))
\}

two options \{ 
6. Otherwise, pick a variable \( v \), \( \text{return} \ DPLL(\phi \mid _{v \leftarrow 0}) \lor DPLL(\phi \mid _{v \leftarrow 1}) \)
\}
Approximation Algorithms (for Optimization Problems)

Recommended Reading: Approximation Algorithms book (Vijay Vazirani).

General Setting: You are trying to find an optimal solution to problem \( A \) that minimizes some objective function (e.g. Vertex Cover, Set Cover, TSP).

Given Instance \( I \):
- \( \text{OPT}(I) \) - value of optimal solution.
- \( \text{ALG}(I) \) - value of the solution produced by your efficient algorithm.

An algorithm is called an approximation algorithm with approx ratio \( \alpha \) if

A instance \( I \)

\[ \text{OPT}(I) \leq \alpha \cdot \text{ALG}(I) \leq \alpha \cdot \text{OPT}(I). \]
Approximation Algorithms for Set Cover, Vertex Cover

Recall: We already saw an approx. alg. for Set Cover.

\[ \forall \text{ instance } I \quad ALG(I) \leq OPT(I) \cdot (\ln(n)+1) \]

Recall: A subset \( S \subseteq V \) is a vertex cover of a graph \( G = (V,E) \) if \( S \) touches all edges in the graph.

Last lecture: VC is NP-complete.

Next: Approx Alg for VC

Set Cover alg. = Approx Alg for VC with \( \alpha = \ln(n)+1 \).

Next: Approx Alg for VC with \( \alpha = 2 \).
Approximation Algorithm for VC (Vertex Cover)

**Idea:** Leverage connection between VC and matchings.

**Recall:** A matching $M \subseteq E$ is a matching if no two edge in $M$ share an endpoint.

**Idea:** Find a maximal matching $M \subseteq E$ that cannot be extended.

**How?** Add edges to $M$ greedily as long as $\exists e \in E$ that doesn't touch $M$.

**Claim 1:** If $M$ is a matching & $S$ is a VC then $|M| \leq |S|$.

**Proof:** $S$ should at least cover the edges in $M$, but to do that $\forall e \in M$ either $u \in S$ or $v \in S$.

Let $\bar{S} = \{\text{all endpoints of edges in } M\}$

$$|\bar{S}| = 2 \cdot |M| \leq 2 \cdot \text{opt}(G).$$

A also $\bar{S}$ is a VC.

$\Rightarrow$ Approx Alg with ratio 2.
Claim: \( \mathcal{S} \) is a vertex cover for \( G \).

Proof: Let's assume by contradiction that \( \mathcal{S} \) is not a VC.

\[ \exists e \in E \text{ s.t. } \mathcal{S} \text{ doesn't touch } e. \]

\[ \Rightarrow e \text{ doesn't touch } M. \]

\[ \Rightarrow \text{ contradiction to the fact that } M \text{ was maximal.} \]

Unique Games Conjecture \( \Rightarrow \) NP-Hard to approx VC to ratio 1.999.
Hardness of Approximation - Traveling Salesperson Problem (TSP)

Given: \( n \) cities with pairwise distances between them

\[ d_{ij} \quad \forall i, j \in \{1, \ldots, n\} \]

Goal: Find shortest tour, i.e., a cycle \( \pi_1, \pi_2, \pi_3, \ldots, \pi_n, \pi_1 \) that visits every city once and minimizes

\[ d(\pi_1, \pi_2) + d(\pi_2, \pi_3) + d(\pi_3, \pi_4) + \ldots + d(\pi_n, \pi_1) \]

Thm: \( A \subset \mathbb{R} \) If TSP has a \( C \)-approx ratio algo in polynomial time \( \Rightarrow \) \( A \subset \mathbb{R} \)

Proof: Given \( G = (V, E) \) unweighted

\[ G' = \left\{ \forall i, j \in V \right. \]

set \( d_{ij} = 1 \) if \((i, j) \in E \)

set \( d_{ij} = C \cdot n \) if \((i, j) \not\in E \)

If \( G \) had a Rucycle

\[ \Rightarrow G' \] has a tour of length \( n \).

If \( G \) has no Rucycle

If the best tour in \( G' \) has length at least \( Cn \).
If we could approximately TSP to ratio $C$ on $G'$, then we could solve RUDCycle exactly on $G$.

Run $\text{ALG}(G')$ \[ \rightarrow \text{val} \leq c.n \quad \text{return Yes} \]
\[ \quad \rightarrow \text{val} > c.n \quad \text{return No.} \]

**Change The Problem:**

If distances satisfy the triangle inequality:

- $\exists$ an approx. algorithm with ratio $2$.
- $\exists$ an approx. algo with ratio $1.5$.
  - Both based on MST.

*This year!* $\exists$ an approx. algo with ratio $1.4999...$.

[Arora]: $\exists$ if distances are Euclidean \[ \Rightarrow \] can get ratio $1 + \frac{3}{\log A} > 2$. [Karlin, Klein, Gharan’20]
Local Search / Hill Climbing

Let $X$ - discrete solution space (usually of exponential size).

$f : X \rightarrow \mathbb{R}$

Goal: maximize $f(x)$ for $x \in X$.

Algorithm:

Pick random $x \in X$.

Repeat $T$ times:

Pick random neighbor $x'$ of $x$

If $f(x') > f(x)$:

$x \leftarrow x'$.
Dealing with Local Maxima

Let $X$ - discrete solution space (usually of exponential size).

$f : X \rightarrow \mathbb{IR}$

Goal: maximize $f(x)$ for $x \in X$.

Simulated Annealing

Pick random $x \in X$

Repeat $t_1$ times: set temp

Repeat $t_2$ times:

Pick random neighbor $x'$ of $x$

If $f(x') > f(x)$:

$x \leftarrow x'$

else: with probability $e^{- (f(x) - f(x'))/\text{temp}}$

$x \leftarrow x'$.