Last time(s):
- reductions
- NP-hardness (and NP-completeness)
- reductions to establish NP-hardness & several natural problems

Today:

How to cope with NP-hardness?
You are interested to solve a computational task A.

Try to show that $A \in P$. (directly, or reduce to ShortestPaths, MaxFlow, LP, ...)

If you succeed then good.

Otherwise, try to show that $A$ is NP-hard. (reduce from 3SAT, ...)

You are likely to succeed. (few problems are not believed to be in $P$ nor NP-hard)

What to do if $A$ is NP-hard?

A. find a special case & $A$ that is in $P$ (NP-hardness uses abstruse instances)

B. intelligent exponential search (mitigate the exponential)
via techniques such as backtracking, branch and bound, ...

C. use an approximation algorithm
   - efficient and incorrect, but not by much

D. use heuristics: no guarantees on running time or approximation,
   but informed by intuition of problem and inputs of interest
Approximation Algorithms for Optimization Problems

input: instance \( x \in I \), which induces a solution space \( S_x \) and value function \( \text{val}_x(.) \)

output: \( S^* \in S_x \) s.t. \( \text{val}_x(S^*) = \text{opt}(x) \) (\( \max_{s \in S_x} \text{val}_x(s) \), or \( \min_{s \in S_x} \text{val}_x(s) \))

Ex: maximum independent set, smallest-weight tour, ...

The approximation ratio of an algorithm \( A \) is

- for maximization problems: \( \chi(A) := \max_{x \in I} \frac{\text{opt}(x)}{\text{val}_x(A(x))} \in [1, \infty) \)
- for minimization problems: \( \chi(A) := \max_{x \in I} \frac{\text{val}_x(A(x))}{\text{opt}(x)} \in [1, \infty) \)

New goal: design efficient algorithms for NP-complete problems with as small approximation ratio as possible
**Vertex Cover**

A vertex cover $S \subseteq V$ is a vertex cover if $\forall e \in E \exists v \in S$ that is an endpoint of $e$.

**Input:** Undirected graph $G = (V, E)$

**Output:** Vertex cover $S \subseteq V$

**Goal:** Minimize $|S|$

VC is a special case of SetCover (given $S_1, \ldots, S_m \subseteq U$, find smallest $I \subseteq [m]$ s.t. $\cup_{i \in I} S_i = U$):

- Set $U := E$ and $S_i := \text{"edges incident to vertex } i\text{"}$.

VC is **NP-hard**: VC reduces to the NP-hard problem IS (if $S$ is a vertex cover)

**Theorem:** VC has an approximation algorithm with approx ratio = 2

Idea: exploit a connection to matchings
**def:** M&E is a matching if edges in M don't share vertices

**claim:** \( S \subseteq V \) vertex cover \( \implies |M| \leq |S| \) (hence \( \max |M| \leq \min |S| \))

**proof:** \( \forall e \in M \exists v \in S \) that touches e (and no other edge) \( \blacksquare \)

**def:** For M&E define \( V(M) := \) all endpoints & edges in M.

**claim:** M&E maximal matching \( \implies V(M) \) vertex cover of size \( 2|M| \)

**proof:** Since M is a matching, we know that \( V(M) \) has \( 2|M| \) vertices. Moreover, if \( V(M) \) is not a vertex cover then \( \exists e \in E \) not touched by \( V(M) \), and so can add e to M. \( \blacksquare \)

This leads to a simple algorithm:

\[
A(G) := 1. \text{ Find a maximal matching } \widetilde{M} \text{ in } G. \\
2. \text{ Output } S := V(\widetilde{M}).
\]

• A **outputs a vertex cover** & is efficient
• A **has approx ratio 2:** \( \frac{\text{val}_G(A(G))}{\text{opt}(G)} = \frac{|V(\widetilde{M})|}{\min_S |S|} = \frac{2|\widetilde{M}|}{\min_S |S|} \leq \frac{2|\widetilde{M}|}{\max_M |M|} \leq \frac{2|\widetilde{M}|}{|\widetilde{M}|} = 2 \)
Hardness of Approximation

Not every NP-hard problem has approximation ratio 2.

**Claim:** if TSP has approx ratio 2 then $P=NP$  

**Proof:** We show how to solve HamCycle (which is NP-complete) in polynomial time.

If $G \in \text{HamCycle}$ then $G'$ has tour of length $n$.  
If $G \not\in \text{HamCycle}$ then every tour must use at least one new edge and so must have length at least $(n-1) \cdot 1 + 1 \cdot 2n = 3n-1$.

An algorithm for TSP with approx ratio 2 can tell the difference.

The same argument also rules out any approx ratio $\omega(n)$ that is poly-time computable! (E.g. $\omega(n) = 2^n$.)

The study of inapproximability involves beautiful tools. See $\Rightarrow$
Heuristics

Say that we want to find maximum of \( f : \mathbb{R} \to \mathbb{R} \).

**Naive idea:** try inputs to \( f \) at random \( \Rightarrow \) this will not get us far

**Better idea:** follow the "up" direction (until you reach a maximum or get tired)

This is a fundamental idea from optimization known as **GRADIENT ASCENT**

\[ z := \text{random starting point} \]

repeat \( M \) times

\[ z' := \text{random point near } z \]

\[ \text{if } \text{val}(z') > \text{val}(z) : z := z' \]

- \( M \) (\# iterations) is chosen heuristically
- "near" means from a neighborhood of \( z \), and choosing this definition matters a lot

Eg for TSP: pick two edges at random & cross them

The behavior depends on how \( f \) looks:

- finds maximum
- may find max after retrying
- \( \text{gradient ascent} \) works badly
Simulated Annealing

Idea: move to worse options with some probability

Fix a temperature schedule: probabilities $p_1 > p_2 > ... > p_n$ with an exponential decay.

The algorithm is:

$z := \text{random starting point}$

for $i=1,2,3,\ldots,N$:

repeat $M$ times:

$z' := \text{random point near } z$

if $\text{val}(z') > \text{val}(z)$: $z := z'$

else w.p. $p_i$: $z := z'$

A reasonable first attempt to solve an NP-complete problem.