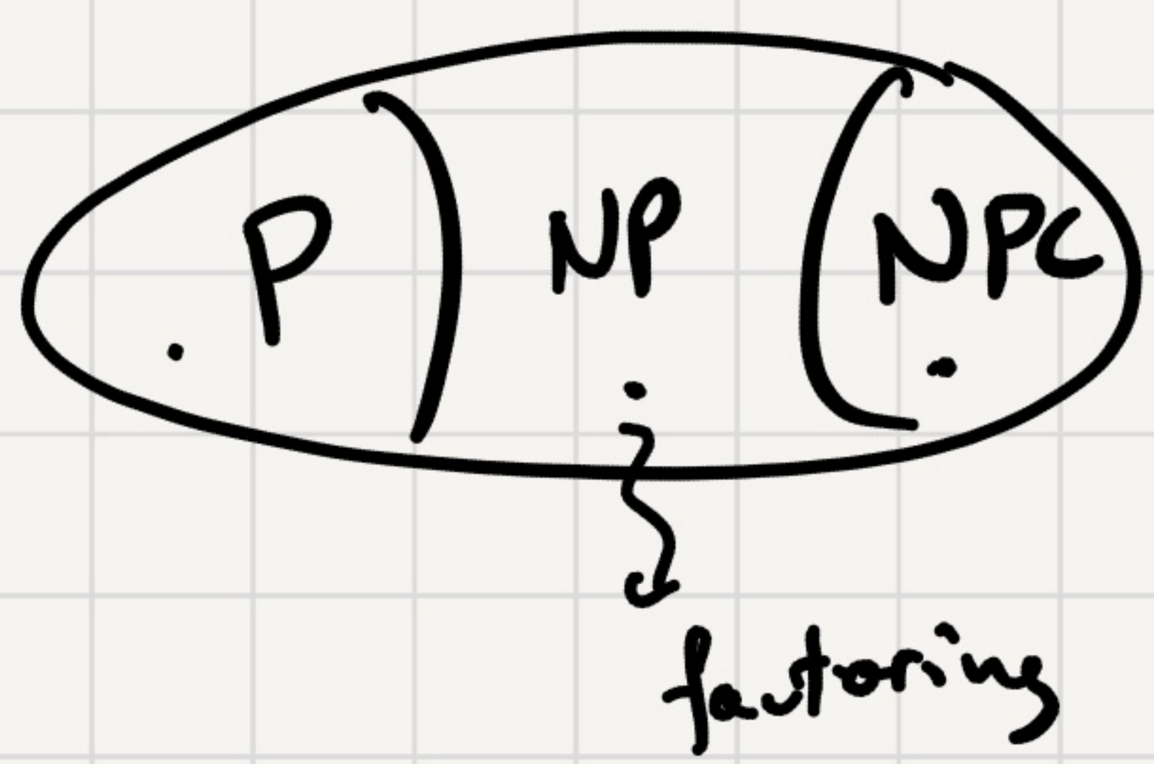
Lecture 21: Coping with NP Hardness

Chapter 9, DPV
Nov 17, 2020.

Suppose you want to solve some problem A

- If you're lucky A has a polynomial time algorithm (either direct or by reducing to Shortest Path, 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, Rud Cycle, ILP, and more and more...)

check out wikipedia
↳ NPC-Problems



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: HornSAT easy
  2SAT is easy.
  3SAT with bounded occurence is hard.

- Negotiation:

- Use correct but inefficient algorithm.

backtracking

branch & bound

- 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 for the well-studied problems
- Fixed Parameter Tractiability. (FPT)

  Algos that are efficient provided that some natural parameter is small.

Backtracking

Important Example: DPLL algorithm for SAT.

Given a CNF formula:

$$\phi = (w \vee x \vee y) \wedge (\bar{x} \vee y) \wedge (\bar{w} \vee z)$$

$$\phi|_{w \leftarrow 1} \underset{\uparrow}{=} (\bar{x} \vee y) \wedge (z).$$

assign 1 to w & simplifies

DPLL(ϕ):

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($\phi|_{v \leftarrow 1}$)
4. If a variable v " " negatively, " " $v \leftarrow 0$ & return DPLL($\phi|_{v \leftarrow 0}$)
5. If ϕ contains a clause with one literal, v or $\bar{v}$, then set v to value b in order to satisfy that literal & return DPLL($\phi|_{v \leftarrow b}$).

two options
6. Otherwise, pick a variable v, return DPLL($\phi|_{v \leftarrow 0}$) $\vee$ DPLL($\phi|_{v \leftarrow 1}$)

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 :

- OPT(I) - value of optimal solution.
- ALG(I) - value of the solution produces by your efficient algorithm

An algorithm is called an approximation algorithm with approx ratio α if

$\forall$ instance I $\quad ALG(I) \leq \alpha \cdot OPT(I).$

$$OPT(I) \leq ALG(I) \leq \alpha \cdot OPT(I).$$

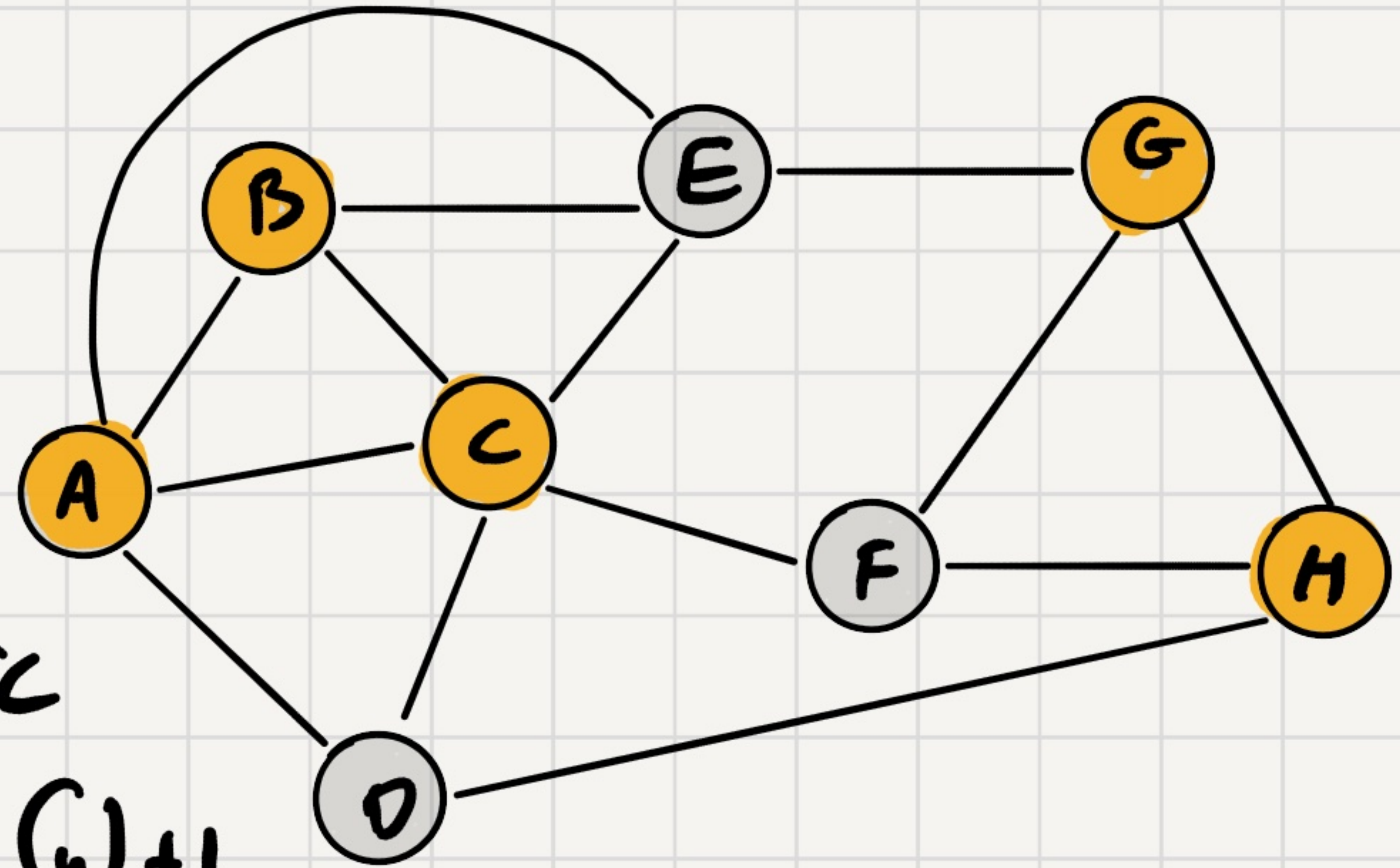
Approximation Algorithms for Set Cover, Vertex Cover

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

$\forall$ instance I $\quad ALG(I) \le OPT(I) \cdot [\ln(n)+1]$

$\alpha = \ln(n)+1$. approx ratio α

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.

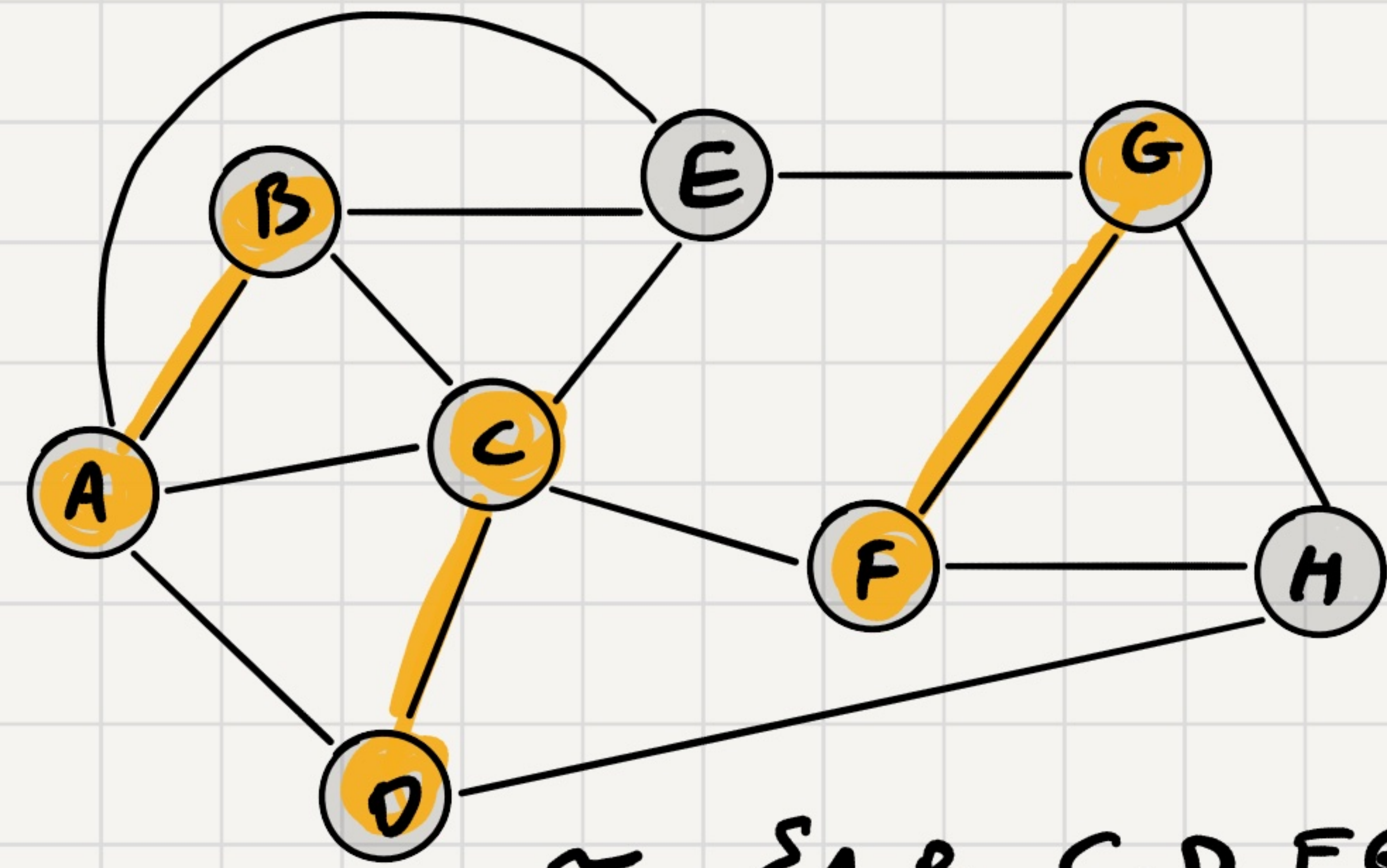
set Cover alg $\Rightarrow$ Approx Alg for VC with $\alpha = \ln(n)+1$.

Next: Approx Alg for VC with $\alpha = 2$.

Approximation Algorithm for VC (vertex cover)

Idea: leverage connection between VC and matchings.

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



$\tilde{S} = \{A, B, C, D, F, G\}$.

Idea: Find a maximal matching $M \subseteq E$

maximal: 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 $|M| \le |S|$.

Proof: S should at least cover the edges in M,

but to do that $\forall e \in M$ either $u \in S$ or $v \in S$.

$e = (u, v)$ $\Rightarrow$ at least $|M|$ vertices in S.

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

$|\tilde{S}| = 2 \cdot |M| \le 2 \cdot OPT(G)$. & also $\tilde{S}$ is a VC.

$\le$: claim 1.

$\Rightarrow$ Approx Alg with ratio 2.

Claim: $\tilde{S}$ is a vertex cover for G.

Proof: Let's assume by contradiction that $\tilde{S}$ is not a VC.

$\exists e \in E$ s.t. $\tilde{S}$ doesn't touch e.

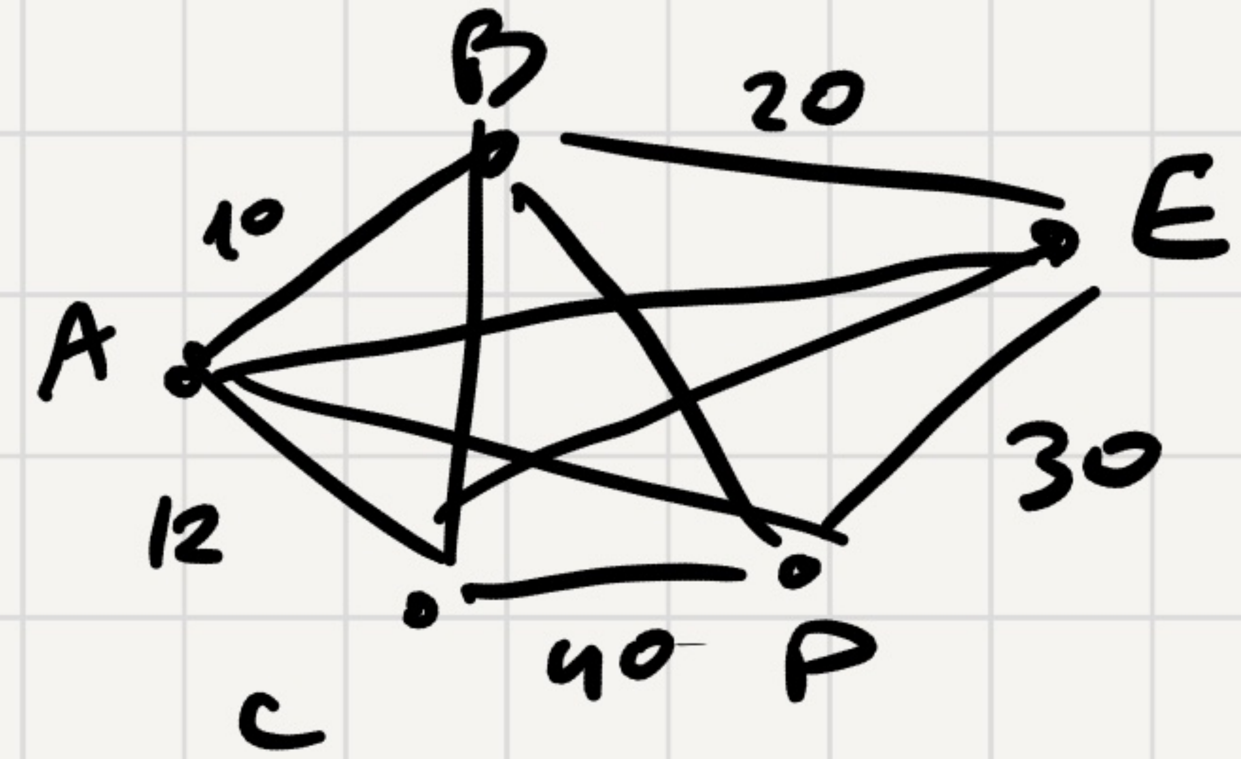
$\Rightarrow$ e doesn't touch M.

$\Rightarrow$ contradiction to the fact that M was maximal. $\square$

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



$$d_{i,j} \quad \forall i,j \in \{1,\dots,n\}.$$

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

$$d_{\pi_1,\pi_2} + d_{\pi_2,\pi_3} + d_{\pi_3,\pi_4} + \dots + d_{\pi_n,\pi_1}.$$

Thm: $\forall C > 1$ If TSP has a C-approx ratio alg in polynomial time Then, $P = NP$.

Rudcycle $\to$ TSP.

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

$$G': \begin{cases} \forall i,j \in V & \text{set } d_{i,j} = 1 \text{ if } (i,j) \in E \\ & \text{set } d_{i,j} = C \cdot n \text{ if } (i,j) \notin E \end{cases}$$

If G had a Rudcycle $\Rightarrow$ G' has a tour of length n.

If G has no Rud Cycle $\Rightarrow$ the best tour in G' has length at least Cn.

If we could approximate TSP to ratio C on G' then we could solve RudCycle exactly on G.

Run $ALG(G')$ $\nearrow$ $val \le c \cdot n$, return Yes

$\searrow$ $val > c \cdot n$, return No. $\square$

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 $1.\underbrace{4999\ldots}_{35\ 9's}9$ [Karlin, Klein, Gharan '20]

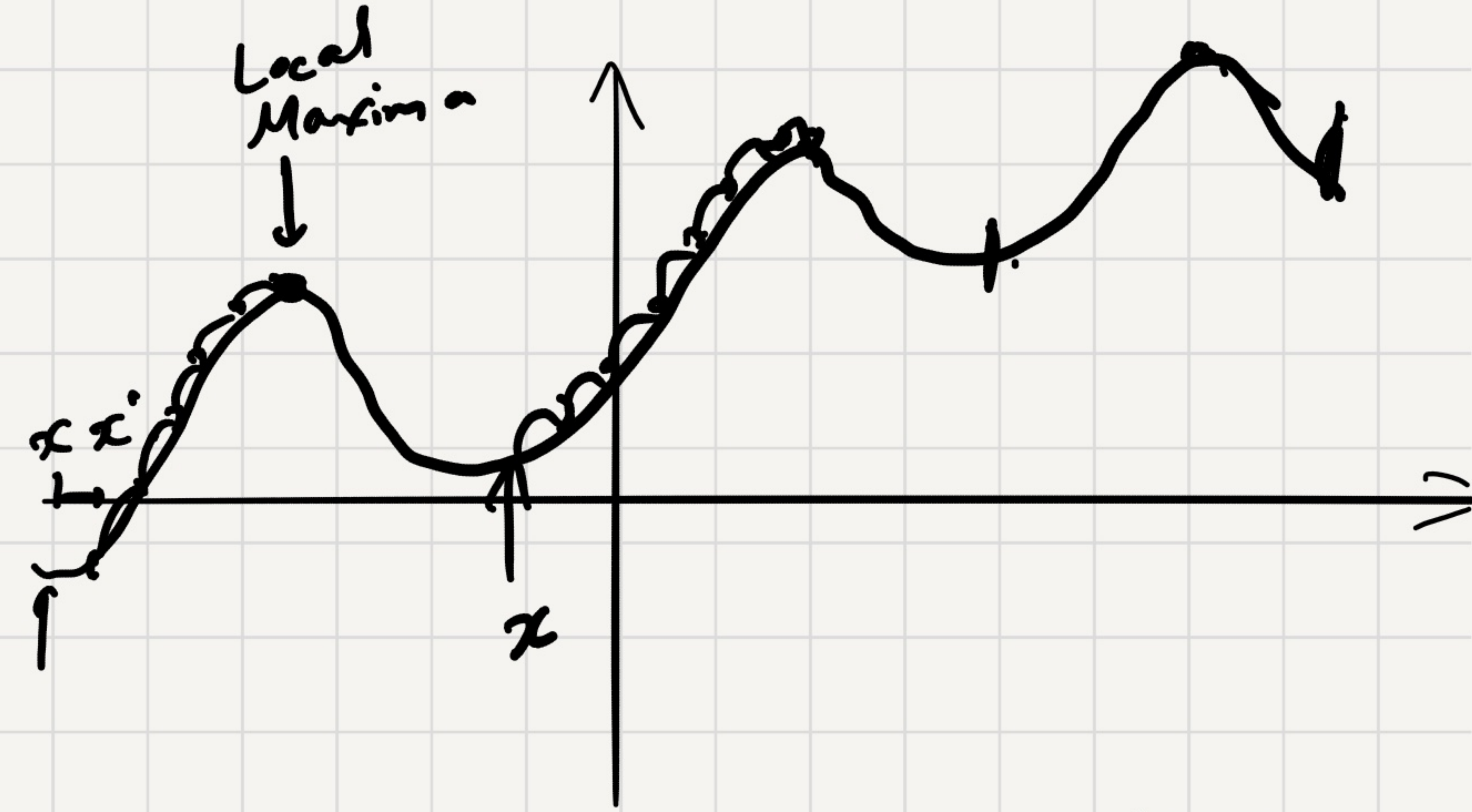
[Arora]: • If distances are Euclidean $\Rightarrow$ can get ratio $1+\varepsilon$ $\forall \varepsilon > 0$.

Local Search / Hill climbing

Let X – discrete solution space (usually of expontial size).

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

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 w. Local Maxima

Restarts & randomization.

Simulated Annealing.

Dealing with Local Maxima

Let X – discrete solution space (usually of expontial size).

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

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 $e^{-(f(x)-f(x'))/temp}$ $x \leftarrow x'$.