

CS 331, Fall 2024
Lecture 25 (12/4)

- Today: - Odds & Ends
- Exponential-time algorithms
- Approximation algorithms

Odds & Ends

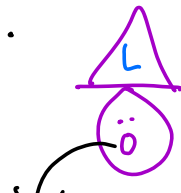
How to prove decision problem L is in...

P: give $A(x)$ that decides " $x \in L$ "
 $\underbrace{\hspace{2cm}}_{\text{poly}(|x|) \text{ time}}$

NP: give $A(x, w)$ s.t. $\underbrace{\hspace{2cm}}_{\text{poly}(|x|) \text{ time}}$ (can be easily proven)



YES $x \in L \Rightarrow$ there's a w s.t. $A(x, w) = \text{YES}$

NO $x \notin L \Rightarrow$ there's no w s.t. $A(x, w) = \text{YES}$



(can't be fixed)

NP-hard: reduce from $L' \in \text{NP-hard}$

$L' \leq_p L \iff$  simulates 

Mythbusters

- True/False:
- If $P \neq \text{NP}$ then all NP-Complete takes exponential time
 - If $P \neq \text{NP}$ then all NP problems take superpolynomial time

They're both false (as implications)

P , NP have very precise defs.

Conclusion requires stronger assumption, e.g. "ETH"

Today: Coping with NP-hardness

Sometimes, you just have to do it.

Option 1: Prove $P = NP$

Option 2: Sacrifice runtime

Option 3: Sacrifice correctness

} today

Some problems today... prove they're NP-complete

MinMakespan: Given list of job lengths, $k \in \mathbb{N}$
how to assign jobs to k machines
w/ min {largest total length on machine}

MaxSAT: Given 3CNF formula \mathcal{I} ,
how many clauses can we satisfy?

TSP: Compute **Hamilton Cycle** of min weight
(stay tuned...)

Exponential-time algorithms

3SAT: m clauses, n Boolean variables

e.g. $(x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_5) \dots$
OR AND

Naive: $O(2^n)$

Idea 1: Divide-and-conquer

Recurse:

- Find any unsatisfied clause C
- Try all 7 assignments to C
- Eliminate all satisfied clauses

$$T(n) \leq 7T(n-3) + O(m)$$

Can check: $T(n) = O\left(\underbrace{(7^{1/3})^n}_{\approx 1.913^n} m\right)$

Idea 2: Divide-and-conquer, reduce

Let $C = \underbrace{l_1 \vee l_2 \vee l_3}_{\text{1. tests}}$ be satisfied

Case 1: $l_1 = \text{TRUE}$

2: $l_1 = \text{FALSE}, l_2 = \text{TRUE}$

3: $l_1 = l_2 = \text{FALSE}, l_3 = \text{TRUE}$

$$T(n) \leq T(n-1) + T(n-2) + T(n-3) + O(m)$$

Works out to $T(n) = O(1.84^n m)$

root of $r^3 = r^2 + r + 1$

SOTA (theory):

3SAT in $\approx 1.31^n$ m time

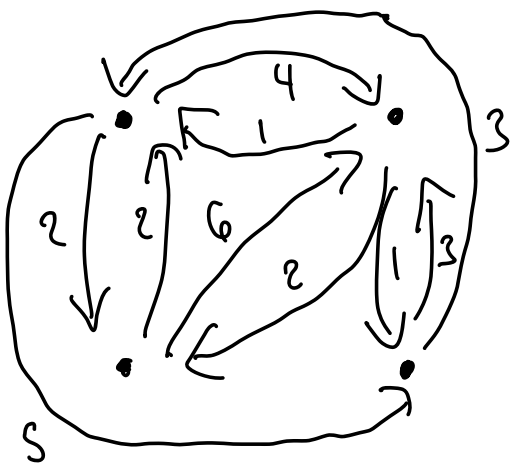
kSAT in $\approx 2^{(1-0.61/k)n}$ m time

SETH: For large constant k , kSAT "strong ETH" not solvable in time $\ll 2^n$

SOTA (practice):

SAT solvers work reasonably well. "CDCL"

TSP: Input complete weighted directed G
 > 0



What is least-weight Hamilton Cycle?

- Visit every vertex once
- Return to start

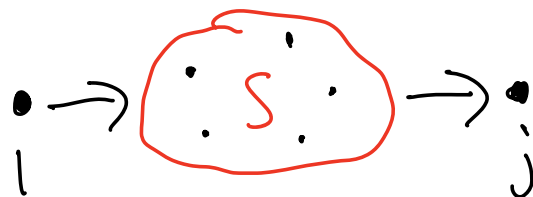
Naive algo: try every permutation (cycle)
 $\approx n!$ time. $\ddot{\smile}$

Bellman-Held-Karp: $\approx 2^n$ time via DP.

e.g. $30! = 2.65 \times 10^{32}$
 $2^{30} = 1.07 \times 10^9$

Key claim: let $C_{S,ij}$ = min cost of
cycle-free path that starts @ i , ends @ j ,
uses exactly intermediate vertices $S \subseteq V$

Then, $C_{S,ij} = \min_{k \in S} C_{S \setminus \{k\}, k} + W(k, j)$



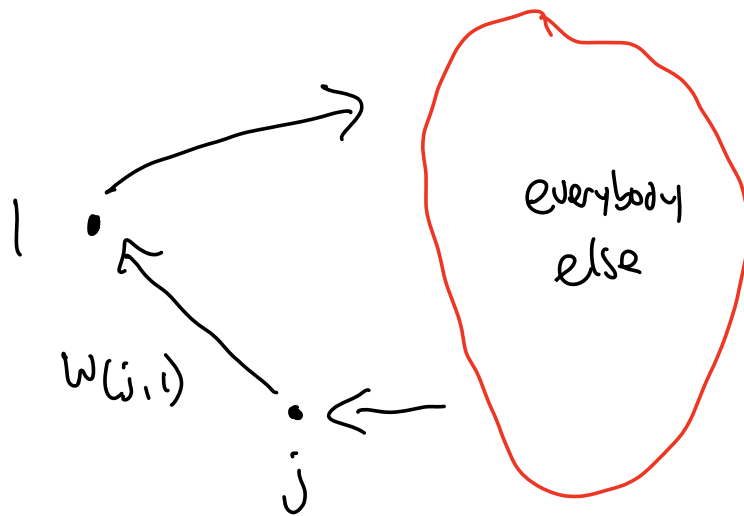
Idea: exponential-sized DP

$$DP[S][i] := C_{Sij}$$

$\leq 2^n$ n subprobs $\times O(n)$ time / subprob

If we know all C_{Sij} , just try all

$S = V \setminus \{i, j\}$ to find optimal



total cost: $C_{Sij} + w(i, j)$

Approximation algorithms

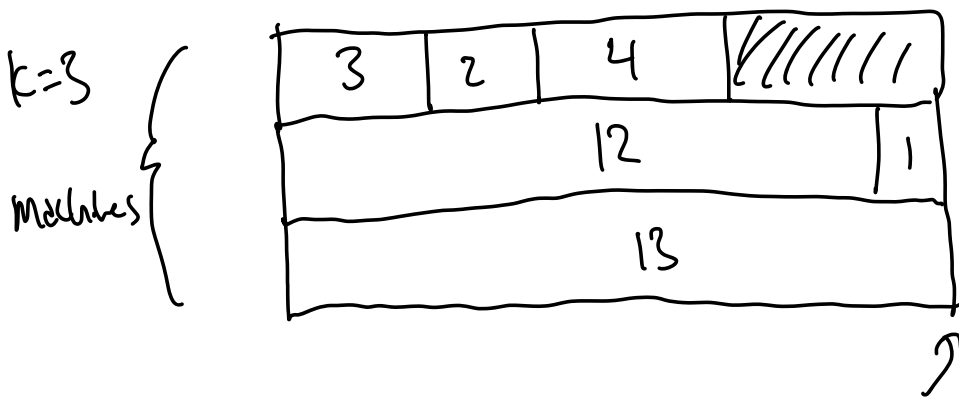
Usually our goal is to compute

$$\text{OPT} = \begin{matrix} \min \\ \text{(or max)} \end{matrix} \left(\text{Some problem, over choices} \right)$$

If the problem is too hard, maybe we're ok
w/ a solution between OPT and $C \cdot \text{OPT}$.

approximation
factor, hope fully small...

Min Makespan $\left(\{1, 2, 3, 4, 12, 13\}, 3 \right)$



makespan = max machine load

Idea: greedy assignment

For job $i \in [n]$:

Assign job i to machine $j \in [k]$

with current smallest load. (arbitrary tiebreak)

Quiz: What is runtime?

Naive: $O(nk)$. Better: $O(n \log(k))$ (heap)

Greedy achieves $C=2$ -approx!

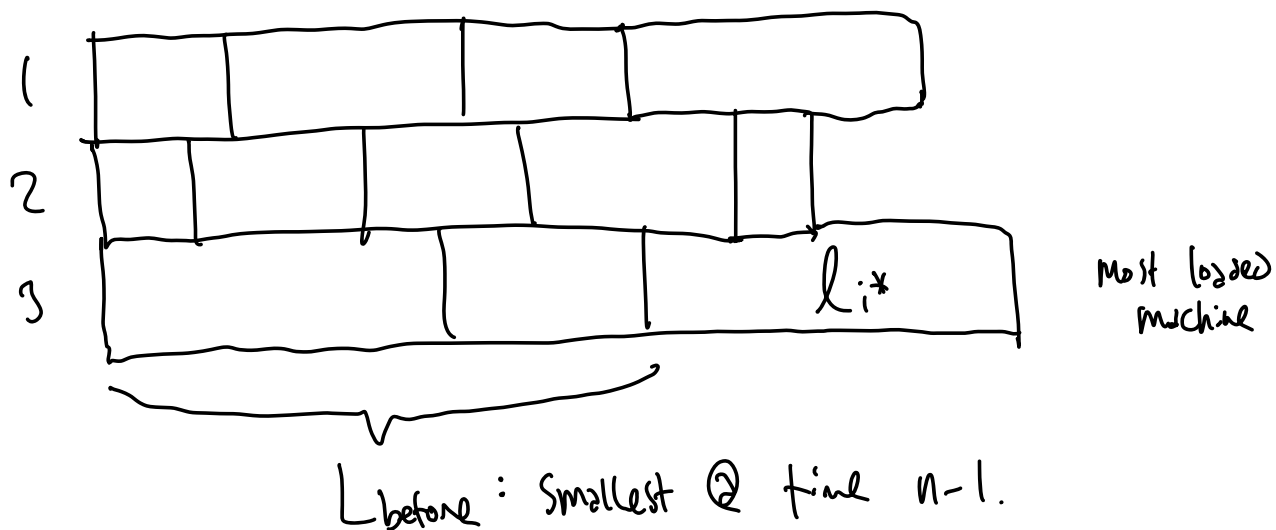
Let $OPT = \min$ makespan

l_1, l_2, \dots, l_n be lengths

Claim 1: $\forall i \in [n], \text{OPT} \geq l_i$

Claim 2: $\text{OPT} \geq \frac{1}{k} \sum_{i \in [n]} l_i$

Suppose greedy produces max load ALG



$$\text{ALG} = L_{\text{before}} + l_{i^*} \quad (\text{Claim 2}) \quad (\text{Claim 1})$$

$$\leq \frac{1}{k} \sum_{i \in [n]} l_i + l_{i^*} \leq \text{OPT} + \text{OPT}$$

$$\leq 2 \cdot \text{OPT} \quad \square \quad (\text{in fact } C = 1.5 \text{ w/ more careful analysis...})$$

Max 3SAT try to satisfy as many clauses

$$\Phi = \bigwedge_{i \in (m)} \phi_i, \quad \phi_i = l_{i1} \vee l_{i2} \vee l_{i3}$$

Idea: randomness. Each ϕ_i satisfied w.p. $\frac{7}{8}$

Pick uniformly random assignment

$$\mathbb{E} \left[\# \text{ clauses satisfied} \right]$$

$$= \mathbb{E} \left[\sum_{i \in (m)} \mathbb{1}(\phi_i \text{ satisfied}) \right]$$

$$\stackrel{(\text{lin. ex.})}{=} \sum_{i \in (m)} \underbrace{\mathbb{E} \left[\mathbb{1}(\phi_i \text{ satisfied}) \right]}_{= \Pr[\phi_i \text{ satisfied}] = \frac{7}{8}} = \frac{7}{8} m$$

We have randomized $C = \frac{8}{7}$ - approximation

(can be made deterministic in time $O(mn)$)

Proof sketch: Assign n one @ a time, best \mathbb{E} .

Suppose x_1, \dots, x_k assigned

$$\mathbb{E} \left[\# \text{ clauses satisfied} \mid x_1, \dots, x_k \right]$$

x_{k+1}
 \vdots
 x_n

$$= \frac{1}{2} \mathbb{E} \left[\# \text{ clauses satisfied} \mid x_1, \dots, x_k, x_{k+1} = \text{TRUE} \right]$$

\vdots
 x_n

$$+ \frac{1}{2} \mathbb{E} \left[\# \text{ clauses satisfied} \mid x_1, \dots, x_k, x_{k+1} = \text{FALSE} \right]$$

x_{k+2}
 \vdots
 x_n

We can determine which is bigger. (lin. ex. again)

Fun fact:

It's **NP**-hard to get $\left(\frac{8}{7} - o(1)\right)$ -approx.
for **Max 3SAT** (Moshkowitz-Raz '08)

Based on Smarter reduction: (Fracta)

Normally **NP**-hard to distinguish m vs. $m-1$. 1 vs. $1 - \frac{1}{m}$

Using amplification can boost $1 - \frac{1}{m} \rightarrow$ smaller
at same cost in problem size.

Turns out to be equivalent to "PCP theorem"

Probabilistically checkable proof: randomly examine
 $O(1)$ bits of witness string w