We will discuss randomization.

Idea: relaxing constraints can make way for faster algorithms that give approximate solns.

Yet another technique to tackle "hard" problems: randomization

Guarantees' types

↳ Produces best / correct soln with a certain probability (monte carlo) ⇒ gambles on optimality of soln

↳ Running time is polynomial with a certain probability (Las Vegas) ⇒ gambles on running time

Any Las Vegas algorithm can be turned into a monte carlo algorithm.

How?

↳ Run LV algorithm a few times and stop it each time it takes more than polynomial time

} suboptimal solns in poly.

} optimal solns in poly.

↳ polynomial cut-off

→ If we have a MC algo. with $p = \frac{1}{2}$, how often do we have to run it to be 99.9% sure of the soln?

1st run → 50% sure it's suboptimal

2nd run → 25% sure " "

3rd run → 12.5% " "

⋮

n^{th} runs → 0.01% sure

$$\left(\frac{50}{100}\right)^n = \frac{0.1}{100}$$

$$n \log \frac{1}{2} = \log \frac{1}{1000}$$

$$n = \frac{(\log 1 - \log 1000)}{(\log 1 - \log 2)} = \frac{3}{\log 2} \approx \frac{10}{1} \approx 10$$

We should insist on guarantees even for randomized algorithms

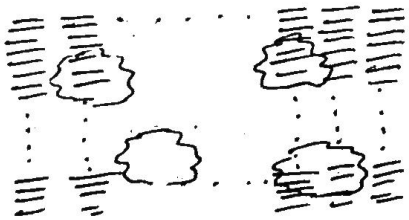
Unlike areas such as encryption, symmetry breaking or simulation, we donot desire perfect randomness in solving np complete problems

Consider clique, where we try to solve it by assigning a 0/1 randomly to each vertex.
Probability of it being the largest clique $= \left(\frac{1}{2}\right)^n$

So what should be our strategy?

A common approach:

- Consider a huge list of solns.



- Land in "good" places
- Explore locally

Consider a randomized algorithm for a NP complete problem that has 90% error probability. Should we expect it to run in polynomial time? NO.

Error probability (when run 50 times) $= (0.9)^{50} \approx 0.5\%$
(50 is a constant)

↳ In polynomial time, we shouldn't expect an almost guaranteed soln.

If there are exponential no. of solns, a poly. time algo. can only check a small subset of it.

We currently don't know if there's an algorithm that runs in polynomial time with fixed error to solve any np-complete problem.

Polynomial time v/s Fixed error

Choose 1

Randomized 3SAT solving

i/p : a boolean formula with n variables

Pick a random 0/1 assignment for the variables

for i in (0, 3 * n):

    Pick a clause that is not satisfied

    Randomly flip one of the variables in the clause

Success of this algo. depends on

I → how far off is the initial assignment

II → how successful is the flipping

I

- we ~~get~~ expect about half of the variables to be correct

$\binom{n}{n/2}$

II

- at least one variable has must be flipped
- chances of flipping the right variable → $\geq \frac{1}{3}$
- probability (mistake) $\leq \frac{2}{3}$

random assignment ($n/2$ away from satisfying assignment)

$\frac{n}{2} \rightarrow \frac{n}{2} + 1 \quad (p \Rightarrow \geq \frac{1}{3})$

$\frac{n}{2} \rightarrow \frac{n}{2} - 1 \quad (p \Rightarrow \leq \frac{2}{3})$

In Each of the 3n iterations, we go ~~from~~ one variable ahead with a probability of $\geq \frac{1}{3}$ and go one variable behind with a probability of $\leq \frac{2}{3}$

probability (making $\frac{n}{2}$ right steps) decreases exponentially with n

$\Rightarrow \left(\frac{1}{3}\right)^{n/2}$ for $n/2$ iterations

for $3n$ iterations,

$x \Rightarrow$ right move

$y \Rightarrow$ wrong move

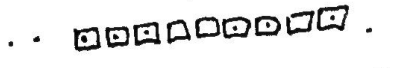
$x + y = 3n$ (effectively $\frac{n}{2}$ right moves)

$x - y \Rightarrow \frac{n}{2} \Rightarrow 2x \geq \frac{7n}{2} \quad x \geq \frac{7n}{4}$

$$\left(\left(\frac{1}{3}\right)^{x} \cdot \left(\frac{2}{3}\right)^{y}\right)\left(\frac{(x+y)!}{x! \cdot y!}\right)$$

$$\left(\frac{1}{3}\right)^{\frac{7n}{4}} \left(\frac{2}{3}\right)^{\frac{5n}{4}} \cdot \frac{(3n)!}{\left(\frac{7n}{4}\right)! \left(\frac{5n}{4}\right)!}$$

$2x = \frac{6n+n}{2} \Rightarrow x = \frac{7n}{4} \qquad y = \frac{5n}{4}$



$3n$

$$\frac{(3n)!}{\left(\frac{7n}{4}\right)! \left(\frac{5n}{4}\right)!}$$

$$\frac{(x+y)(x+y-1)\cdots(x+1)\,x!}{x!\,y!}$$

$$\frac{3n \cdot 3n-1 \cdots \frac{4n}{4}}{\left(\frac{5n}{4}\right)!}$$

Algorithm has faster exponential running time than deterministic algorithm at the cost of a certain error probability.

→ Very little known results for V.C., ind. set, clique in randomizations

When can one use randomized algo. w/o guarantees?

→ Stakes are low

→ No better algorithm is available or known to someone

Randomization vs Determinism

→ A debate whether randomization is really useful

→ On the other hand, randomized algorithms are harder to trick into worst case behavior and could be easier to analyse.

Recap

- Optimizing search trees (preprocessing)
- Approximation
  - ↳ constant
  - ↳ PTAS
- Randomization

P → complexity class of all problems that can be solved in polynomial time on deterministic RAM

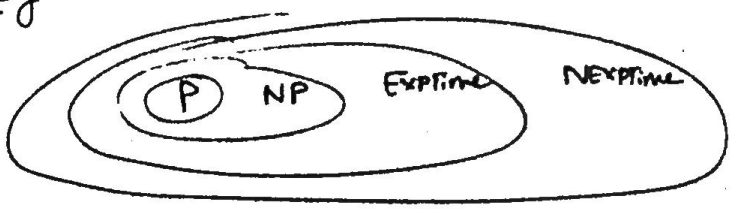
NP → complexity class of all problems that can be solved in polynomial time on non-deterministic RAM

To be more rigorous;

P and NP were discussed & introduced in the context of decision problems and the model of a Turing machine.

	RAM	non deterministic RAM
exponential time	EXPTIME	NEXPTIME
polynomial time	P	NP

Hierarchy



P $p(n)$

NP $p(n)$ NDRAM

EXPTIME $O(2^{p(n)})$

NEXPTIME $O(2^{p(n)})$ NDRAM

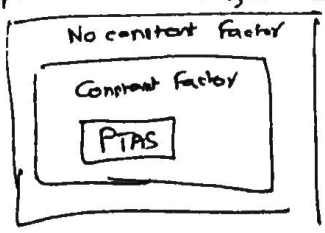
IS EXPTime = NEXPTime?

Who knows!

* Finding a strategy to solve games usually are provably harder than NP-complete problems (unless P=

These are also problems that are harder than np-complete problems even if $P = NP$.

Approximation algorithms



Randomized algorithms