Dealing with NP completeness

- Try efficient exponential searches
- Close to best soln is as good as the best soln.

Laws of NP completeness (need to find loopholes)

→ Unless P=NP, there's no poly algo for NP problems

Maybe we can find smarter expo. algorithm. Maybe we can find approx. solns.

Running time (I) $2^n \cdot n^2$ vs (II) $\frac{2^n \cdot n^2}{10}$

↑
10 times faster!
(misleading...)

Case (I)

→ max. i/p size we can handle = 30

Case (II)

$$2^n \cdot n^2 = \frac{2^m \cdot m^2}{10}$$

$$2^{30} \cdot 30^2 = \frac{2^m \cdot m^2}{10}$$

$$2^{31} \cdot 30^2 \cdot 5 = 2^m \cdot m^2$$

$m \approx 33$ (not much bigger than 30)

Case (III) 100 x faster

$$\frac{2^n \cdot n^2}{100} = 2^{30} \cdot 30^2$$

$n \approx 36$ (sheeeeesh!)

Brute force on clique

- explores <u>all</u> possibilities

Sometimes it's useful:

→ not much time to design & program

→ small instances ($n < 20$)

→ need to solve only few times

when can we use brute force (subjective) $\frac{2}{0}$

# instances ↓ \ Size of each instance →	≤10	≤20	≤30	> 100
1	✓	✓	✓	X
100	✓	✓	✓	X
10000	✓	✓	X(?)	X

Loophole 1: "No polynomial time algorithm unless P=NP"

↳ What about $2^{\sqrt{n}}$ ← not polynomial, but faster than 2^n

$n=50$, $2^{\sqrt{n}} = 2^{\sqrt{50}} = 2^7$ ~~= 128~~

~~23~~ $2^n = 2^{50}$

$\frac{2^{50}}{2^7} = 2^{43}$ times faster

loophole 2:

↳ What about $1.1^{\sqrt{n}}$

$n = 50$, $1.1^{\sqrt{n}} = 1.1^{\sqrt{50}}$

$2^n = 2^{50}$

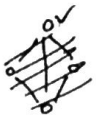
~~$2^{\sqrt{50}}$~~ $\frac{2^{50}}{1.1^{\sqrt{50}}}$ > trillion

if we have small exponents (or) small bases

our algorithm will be reasonably fast albeit exponential

Brute force on V.C.

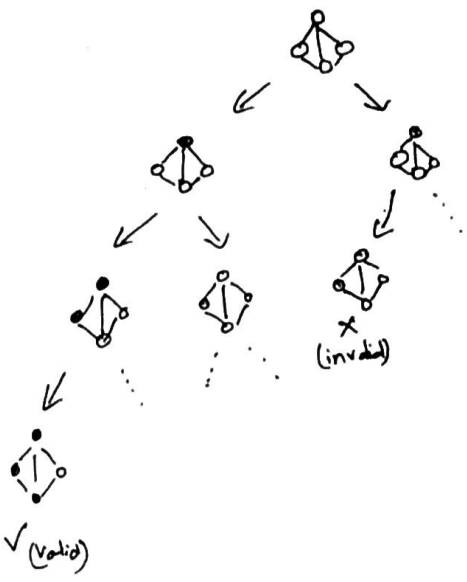
consider  , we have to validate 2^4 possibilities



Some of these possibilities are redundant and non-sensical.

if we find a soln with 2 vertices, why try with 3 vertices?

Search tree ?



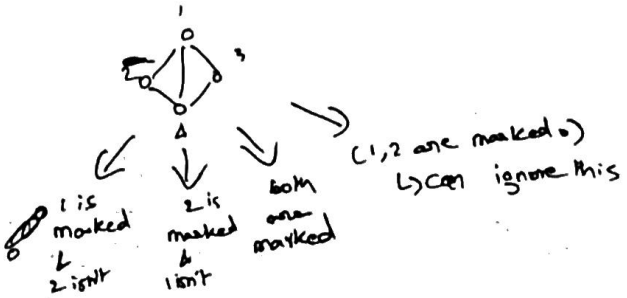
~~selected~~ shaded (1)
dotted (0)

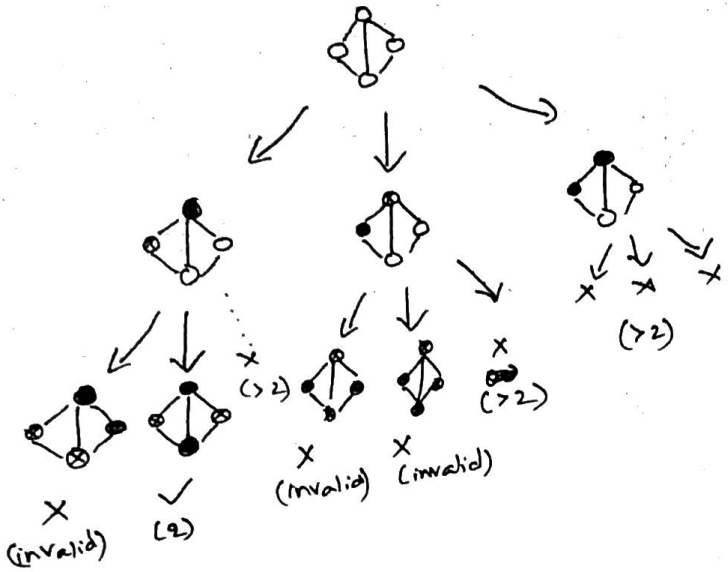
instead of 16, we explore only 9.

the speedup is highly dependant on input structure.

we stop at two situations:
→ we found a soln.
→ we found an edge left out

improving the search tree



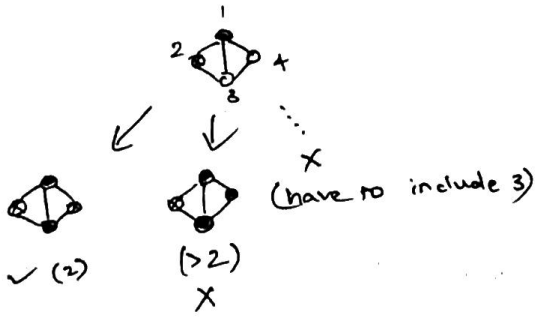


⇒ always finds smallest V.C

⇒ at each level determines assignment of atleast 1 vertex

one more insight ⇒ for any edge if one vertex is marked 0, the other vertex has to be 1 for a valid soln.

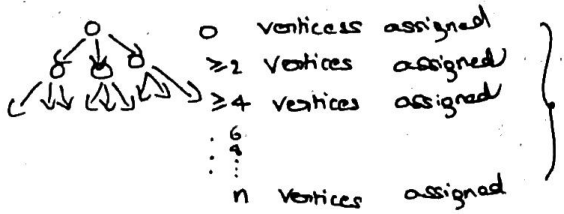
⇒ search tree is shorten



Analysis:

brute force ← 2^n assignments

Search tree ←



depth = $\frac{n}{2}$ branching factor = 3

~~At the lowest level, we have~~

This tree has $3^{n/2}$ leaves. ⇒ # distinct ~~possible~~ assignments considered by the search tree

$3^{n/2} = (1.732)^n$ (better than 2^n)

↳ worst case analysis

$n = 50$

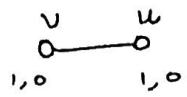
$2^{50} \Rightarrow$ to 1.13×10^{15} (NOPE!)

$(1.732)^{50} \Rightarrow$ 3.71×10^{11} (1000 times faster in worst case)

HOPE!!!

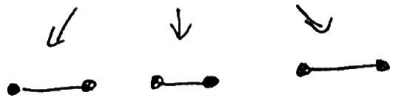
What about clique & independent set?

independent set

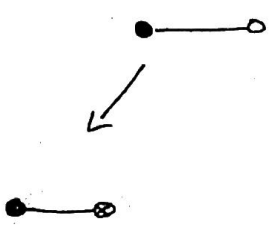


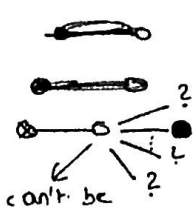
→ at most one of {v, u} can be in the i.s.

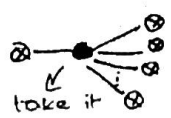
→ both could be 0.

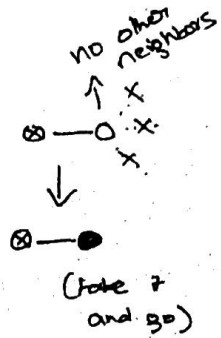


also, if a vertex is marked 1 ⇒ mark all its neighbors 0.









Similar to V.C. we get a branching factor of 3 and assign all-cast 2 vertices at each level.

Size $\Rightarrow$ 1.732^n

Same for clique as well.

How faster can it get?

independent set : 1.189^n (or) 1.211^n (Search tree size)

clique : (same)

3SAT : 1.496^n

TSP : 2^n

Maybe someday we get $(1.001)^n$

We don't know

Max. problem size we can handle if we have a limit of 1 billion soln.?

$1.189^n \Rightarrow n \leq 119$

$2^n \Rightarrow n \leq 29$

$1.1^n \Rightarrow n \leq 217$

$1.211^n \Rightarrow n \leq 108$

$1.496^n \Rightarrow n \leq 51$

(worst case views)

In practice, we solve instances of size about 1000 for independent set & clique.

Most inputs seem friendly.