1 2 3 4 5 (aside)

\+ − − + +

1.5 3.5

H_1 (avg entropy) H_2

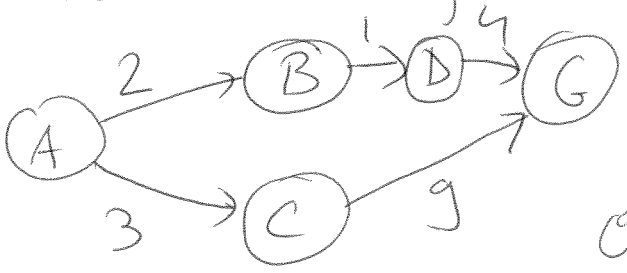
Pick one that gives lowest avg entropy

Computational complexity → more generally

Graph traversal: moving around on a graph
→ shortest path or ask some question about types of paths in the graph.

A ⟶ G shortest

Mimick Breadth-first search

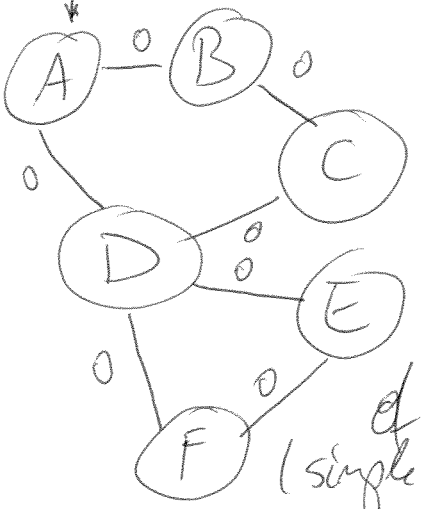


Priority queue to put nodes in the order of cost so far.

B, 2	C, 3	

Nice!

Given a graph is there a cycle that touches every edge exactly once? (Euler cycle)



yes!

↓

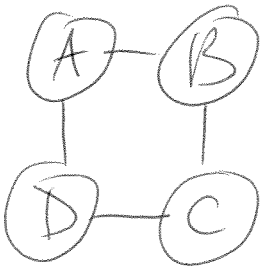
Poly time in no. of edges (simple walk)

Given a graph is there a cycle that touches every node exactly once? (Hamilton cycle)

No Hamilton cycle (D is a "choke point"

Consider permutations of nodes → check if it is a legal cycle

ABCEDFA → not a cycle not edges



Both Hamilton & Euler cycles

$n!$ (n is no of vertices)

→ <u>not</u> poly-time!

↓

Check a proposed solution easily

But no. of possible solutions to check is huge! ($O(b^n)$)

$O(n!)$ → combinatorial problem

n ↑ size of object of interest

Nondeterministic Polynomical (NP) problems

e.g. Hamilton cycle

Large space of possible solutions

Poly-time algorithm for <u>checking</u> if a solution is correct

But in worst-case, we may need to check <u>all</u> sol!

Poly-time algorithms: $O(n^k)$ (P)

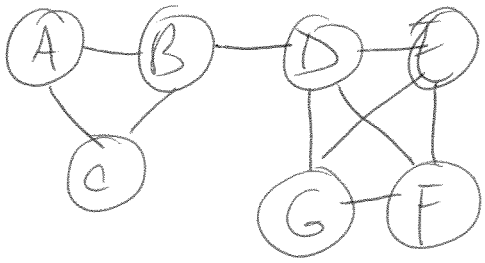
↑ k constant

$P \subseteq NP$

Is $P = NP$? (Probably No)

<u>E.g.</u> Clique-finding in graphs

Clique = set of vertices s.t. each 2 vertices are connected by an edge



Cliques of size 2: all edges

Clique of size 3: ABC

Clique of size 4: DEFG

What is the maximum size clique in a graph?

↳ Is there a clique of size k in the graph (k given) (yes/no, aka decision question)

Guess-and check: pick a subset of size k - check if all nodes are connected

↳ poly-time

Nr of subsets: $\binom{n}{k}$ → "combinatorial"

↳ Heuristics → "special" structure & start the search there

↳ Randomization → travel the solution space randomly (explore it) (control the complexity)

Satisfiability (Cook, 1970s)

$$(x_1 \wedge x_2 \wedge \neg x_3) \vee \underbrace{(\neg x_1 \wedge x_2)}_{\text{clauses}} \vee (x_3 \wedge x_4)$$

n variables

at most k variables in a clause

Is there an assignment of values to vars which makes formula true (satisfying assignment)

$$(x_1 \vee x_2) \wedge \underbrace{(\neg x_1 \vee x_3 \vee x_4)}_{\text{disjunctions as clauses}}$$

↑ conjunctions

$x_1 = t \quad x_2 = f \quad x_3 = t \quad x_4 = f \Rightarrow t$ (poly-time)

Nr. of possible combinations is $O(2^n)$!

NP-complete problem: any other NP problem can be reduced to K-satisfiability ($K=3$)

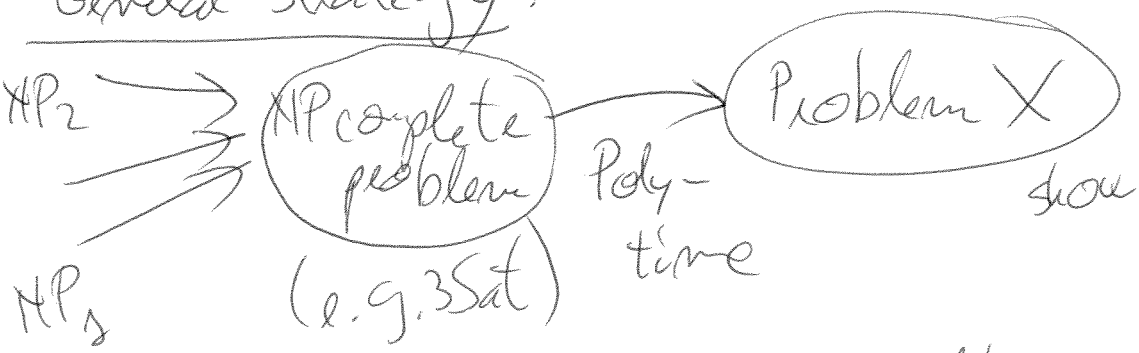
→ Prioritize clauses that are hard to satisfy (assign vals to these first)

→ Prioritize variables that are "active"

(heuristics)

Randomization: given certain values, try at random assignments to the others

General strategy:



Solve X → convert it to another NP problem that can be solved (approximately) well.

$X_1 \ldots X_n$ $\quad X_i \in D_i$ (set of values)

Constraints on groups of X_i's

$X_i \leq X_j$ $\qquad X_i \neq X_j \ \forall i \neq j$ (resource allocation)

Use of randomization to solve NP problems (& other problems)

"Dice" ⟶ Random

seed (number) ⟶ generates a sequence of numbers.

If you know the seed → seq is fully predictable (always the same)

Given a sequence, it's really hard to predict next number (without seed)

pseudo-random number generators

Plant → makes devices faulty / ok

if ok → all devices work

faulty → at least 50% do not work

n devices ⇒ check all devices → answer

⇒ check & stop if n/2 faulty or cannot have n/2 faulty anymore.

Randomized ⇒ pick at random, check

↓

compute conditional prob of faulty, given results so far → stop if ≤ ε

Constraint Satisfaction Problems (CSP):

Find values $X_i \in D_i$ s.t. all constraints are satisfied

↳ "relaxations": minimize no. of constraint violations

Easy problems can be solved efficiently