NP-Hardness, Part 1

Decision Problems, P, Reductions, NP

Easy or Hard?

So far: problems with efficient algorithms. Small changes to a problem can make it seem far harder.

we know a fast algorithm	no fast algorithm known
shortest simple path	longest simple path
Euler tour (every edge once)	Hamiltonian cycle (every vertex once)
2-coloring a graph	3-coloring a graph
minimum spanning tree	traveling salesman tour

Easy or Hard?

So far: problems with efficient algorithms. Small changes to a problem can make it seem far harder.

we know a fast algorithm	no fast algorithm known
shortest simple path	longest simple path
Euler tour (every edge once)	Hamiltonian cycle (every vertex once)
2-coloring a graph	3-coloring a graph
minimum spanning tree	traveling salesman tour

This part of the course: how to give evidence that a problem is hard, so we can stop looking for an efficient algorithm.

Decision Problems

Decision Problems

Definition

A decision problem is a problem whose answer is **yes** or **no**.

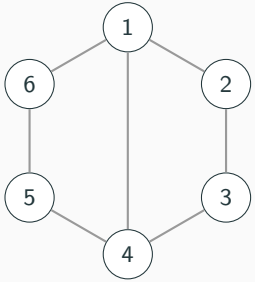
Every input is either a yes-instance or a no-instance.

Three running examples:

- **Independent Set**: given G and k, is there a set of $\geq k$ vertices with no edge between any two of them?
- **Subset Sum**: given integers S and a target t, does some subset of S sum to exactly t?
- **Graph Coloring**: given G and k, can the vertices be colored with at most k colors so that adjacent vertices differ?

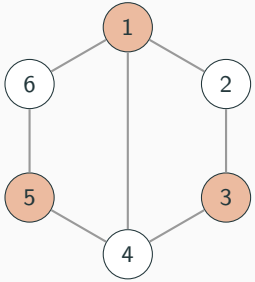
Example: Independent Set

Is there a set of $\geq k$ vertices, no two of them adjacent?



Example: Independent Set

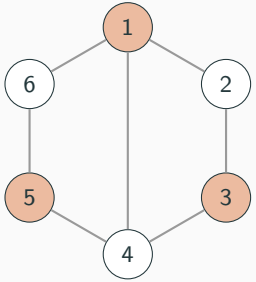
Is there a set of $\geq k$ vertices, no two of them adjacent?



- $k = 3$: yes. The set $\{1, 3, 5\}$ has no edge inside.

Example: Independent Set

Is there a set of $\geq k$ vertices, no two of them adjacent?

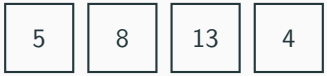


- $k = 3$: yes. The set $\{1, 3, 5\}$ has no edge inside.
- $k = 4$: no. Any 4 of the 6 vertices include two neighbors on the hexagon.

Same graph, different k: a different answer. The *instance* is the pair (G, k).

Example: Subset Sum

$S = \{5, 8, 13, 4\}$. Does some subset sum to exactly t?



Example: Subset Sum

$S = \{5, 8, 13, 4\}$. Does some subset sum to exactly t?



- $t = 17$: yes, since $13 + 4 = 17$.

Example: Subset Sum

$S = \{5, 8, 13, 4\}$. Does some subset sum to exactly t?

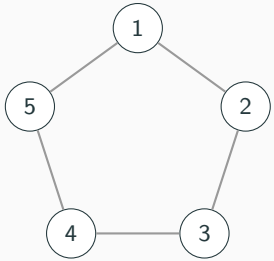


- $t = 17$: yes, since $13 + 4 = 17$.
- $t = 11$: no. The 15 nonempty subsets give the sums $4, 5, 8, 9, 12, 13, 17, 18, 21, 22, 25, 26, 30$, and 11 is not among them.

To answer no, we checked every subset. With n numbers, that is 2^n of them.

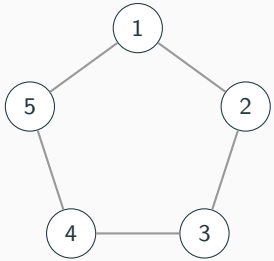
Example: Graph Coloring

Can G be colored with $\leq k$ colors so that every edge has two different colors?



Example: Graph Coloring

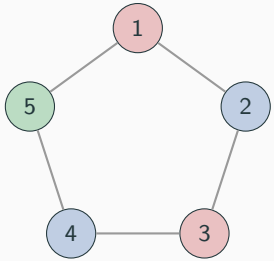
Can G be colored with $\leq k$ colors so that every edge has two different colors?



- $k = 2$: no. Colors must alternate around the cycle, but it has odd length 5, so vertices 5 and 1 would clash.

Example: Graph Coloring

Can G be colored with $\leq k$ colors so that every edge has two different colors?



- $k = 2$: no. Colors must alternate around the cycle, but it has odd length 5, so vertices 5 and 1 would clash.
- $k = 3$: yes. Alternate two colors and give vertex 5 a third.

From Optimization to Decision

Turning Optimization into Decision

Add a threshold k to the objective. Minimize becomes "is there a solution of cost $\leq k$?"; maximize becomes "is there one of value $\geq k$?".

optimization	decision
find a shortest s–t path	is there an s–t path of length $\leq k$?
find a minimum spanning tree	is there a spanning tree of weight $\leq k$?
find a shortest tour of all cities	is there a tour of length $\leq k$?
find a largest independent set	is there an independent set of size $\geq k$?
color G with the fewest colors	can G be colored with $\leq k$ colors?
knapsack: most value within weight W	is there a set of weight $\leq W$ and value $\geq k$?

Decision Is Just as Hard

- The decision version is no harder: solve the optimization problem, then compare the optimum with k.
- It is usually not easier either: binary search over k recovers the optimal value.

Decision Is Just as Hard

- The decision version is no harder: solve the optimization problem, then compare the optimum with k.
- It is usually not easier either: binary search over k recovers the optimal value.

Example. The fewest colors for G is between 1 and n.

- Ask "$\leq n/2$ colors?", then halve the remaining range based on the answer.
- After $\lceil \log_2 n \rceil$ questions we know the exact minimum.

A fast algorithm for the decision version would give a fast algorithm for the optimization version. So it is enough to study decision problems.

The Class P

Polynomial Time and the Class P

An algorithm runs in polynomial time if its worst-case running time is $O(n^c)$ for a constant c, where n is the input size.

Definition

P is the class of decision problems that can be solved in polynomial time.

- **Shortest path** (no negative cycles): is there an s–t path of length $\leq k$? Dijkstra or Bellman-Ford.
- **Strongly connected components**: does G have at most k SCCs? Two DFS passes, $O(n + m)$.
- **Minimum spanning tree**: is there a spanning tree of weight $\leq k$? Prim or Kruskal.

Why Polynomial?

n	n^2	n^3	2^n
10	100	1,000	1,024
50	2,500	125,000	1.1×10^{15}
100	10,000	10^6	1.3×10^{30}

- At 10^9 steps per second, 2^{100} steps take about 4×10^{13} years.
- A faster computer barely helps an exponential algorithm: doubling the speed only adds 1 to the n we can handle.
- Polynomial is a coarse but robust notion of "efficient": it is closed under composition.

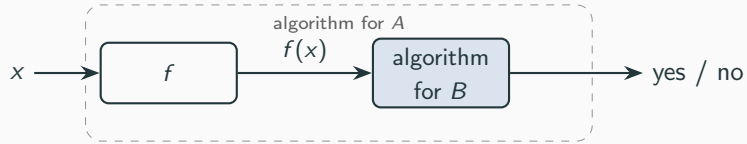
Reductions

Reductions

Definition

A polynomial-time reduces to B, written $A \leq_p B$, if there is a polynomial-time algorithm f turning each instance x of A into an instance $f(x)$ of B such that

$$x \text{ is a yes-instance of } A \iff f(x) \text{ is a yes-instance of } B.$$



Translate the question, then ask the B-solver. The answer is preserved.

Example: Scheduling $\leq_p$ Graph Coloring

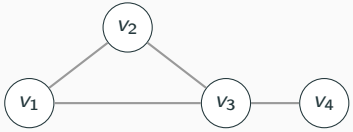
Scheduling: n tasks, k time slots, and a list of conflicting pairs. Can every task get a slot so that conflicting tasks differ?

Tasks t_1, t_2, t_3, t_4

Conflicts: $\{t_1, t_2\}$, $\{t_1, t_3\}$, $\{t_2, t_3\}$, $\{t_3, t_4\}$

$k = 3$ slots

$\xrightarrow{f}$



- One vertex per task, one edge per conflicting pair, same k. Time $O(n^2)$.

Example: Scheduling $\leq_p$ Graph Coloring

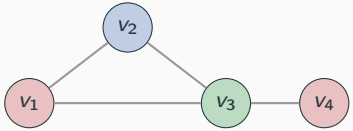
Scheduling: n tasks, k time slots, and a list of conflicting pairs. Can every task get a slot so that conflicting tasks differ?

Tasks t_1, t_2, t_3, t_4

Conflicts: $\{t_1, t_2\}$, $\{t_1, t_3\}$, $\{t_2, t_3\}$, $\{t_3, t_4\}$

$k = 3$ slots

$\xrightarrow{f}$



- One vertex per task, one edge per conflicting pair, same k. Time $O(n^2)$.
- Slot = color: a schedule with k slots is exactly a proper k-coloring, in both directions.

Reductions Transfer Algorithms

Claim

If $A \leq_p B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.

Proof. On input x of size n:

1. Compute $f(x)$: time $O(n^c)$, so $|f(x)| = O(n^c)$ too (it cannot write more than it has time for).
2. Run B's algorithm on $f(x)$: time $O(|f(x)|^d) = O(n^{cd})$.

Total time is polynomial, and the answer is correct by the definition of a reduction. □

Reductions Transfer Algorithms

Claim

If $A \leq_p B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.

Proof. On input x of size n:

1. Compute $f(x)$: time $O(n^c)$, so $|f(x)| = O(n^c)$ too (it cannot write more than it has time for).
2. Run B's algorithm on $f(x)$: time $O(|f(x)|^d) = O(n^{cd})$.

Total time is polynomial, and the answer is correct by the definition of a reduction. □

Contrapositive: hardness flows forward

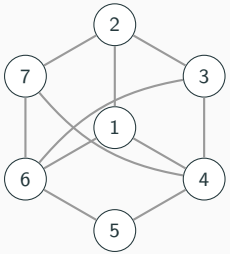
If $A \leq_p B$ and A has no polynomial-time algorithm, then neither does B.

To show B is hard, reduce from a known hard problem to B.

The Class NP

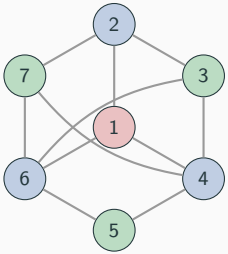
Solving vs. Verifying

We know no polynomial-time algorithm for 3-coloring. But if someone hands us a coloring, checking it is easy: look at every edge.



Is it 3-colorable?

certificate



Every edge has two colors: yes.

The Class NP

Definition

NP is the class of decision problems where every yes-instance has a certificate (a short proof) that can be verified in polynomial time.

Think of a game between Alice and Bob.

- Alice knows the answer is yes and wants to convince Bob.
- Bob only runs polynomial-time algorithms, so he cannot search for a solution himself.
- Alice sends a certificate; Bob checks it.

No-instances need no certificate: whatever Alice sends, Bob must not be fooled.

The Class NP

Definition

NP is the class of decision problems where every yes-instance has a certificate (a short proof) that can be verified in polynomial time.

Think of a game between Alice and Bob.

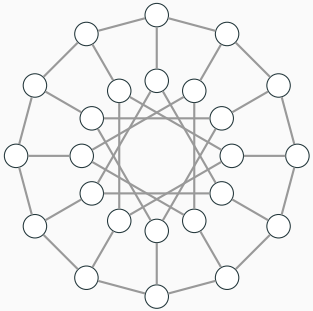
- Alice knows the answer is yes and wants to convince Bob.
- Bob only runs polynomial-time algorithms, so he cannot search for a solution himself.
- Alice sends a certificate; Bob checks it.

No-instances need no certificate: whatever Alice sends, Bob must not be fooled.

NP stands for *nondeterministic polynomial time*, not "not polynomial".

Example: Independent Set $\in$ NP

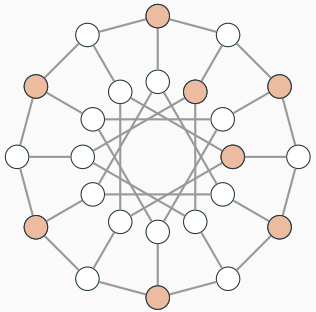
An independent set is a set S of vertices with no edge inside. Given G and k: is there one with $|S| \geq k$?



Is there an independent set of size 8? Not obvious.

Example: Independent Set $\in$ NP

An independent set is a set S of vertices with no edge inside. Given G and k: is there one with $|S| \geq k$?



Is there an independent set of size 8? Not obvious.

Certificate: the set S itself (orange).

Verify: $|S| \geq 8$, and no edge has both ends in S. Time $O(n + m)$.

Example: Subset Sum $\in$ NP

Given $S = \{3, 1, 7, 11, 4, 9, 2, 6\}$ and $t = 20$: does some subset sum to 20?



- Trying all 2^n subsets is too slow in general.

Example: Subset Sum $\in$ NP

Given $S = \{3, 1, 7, 11, 4, 9, 2, 6\}$ and $t = 20$: does some subset sum to 20?



- Trying all 2^n subsets is too slow in general.
- **Certificate**: the subset $\{3, 7, 4, 6\}$.
- **Verify**: $3 + 7 + 4 + 6 = 20 = t$. Time $O(n)$.

$\mathsf{P} \subseteq \mathsf{NP}$

Claim

$\mathsf{P} \subseteq \mathsf{NP}$.

Proof. Let $L \in \mathsf{P}$ with polynomial-time algorithm $\mathcal{A}$. Verifier: on input x and any certificate c, ignore c and run $\mathcal{A}(x)$. It runs in polynomial time and accepts exactly the yes-instances. $\square$

$\mathsf{P} \subseteq \mathsf{NP}$

Claim

$\mathsf{P} \subseteq \mathsf{NP}$.

Proof. Let $L \in \mathsf{P}$ with polynomial-time algorithm $\mathcal{A}$. Verifier: on input x and any certificate c, ignore c and run $\mathcal{A}(x)$. It runs in polynomial time and accepts exactly the yes-instances. □

The big open question

Is $\mathsf{P} = \mathsf{NP}$? Can every problem whose solutions are easy to *check* also be *solved* easily?

Most researchers believe $\mathsf{P} \neq \mathsf{NP}$. It is one of the Clay Millennium Prize Problems ($1,000,000).

Exercise

1. Show that Hamiltonian Cycle is in NP: what is the certificate, and how is it checked?
2. Someone shows 3-Coloring $\leq_p X$ and also finds a polynomial-time algorithm for X. What follows?
3. Someone shows $X \leq_p$ Shortest Path. Does that say anything about how hard X is?
4. Is it clear that "G is not 3-colorable" is in NP?

Solution

1. Certificate: the cycle, as an ordering $v_1, \ldots, v_n$ of the vertices. Check that every vertex appears exactly once and that $\{v_i, v_{i+1}\}$ and $\{v_n, v_1\}$ are edges. Time $O(n + m)$.
2. 3-Coloring $\in$ P by the claim. 3-Coloring is known to be NP-complete (next lecture's term), so this would even give P = NP.
3. Yes, an upper bound: $X \in$ P, because Shortest Path is in P. It says nothing about X being hard.
4. No. A coloring proves colorability, but there is no known short proof that *no* coloring exists. Such "complement" problems form the class co-NP.

Summary

- **Decision problems**: yes/no answers; optimization problems have decision versions of the same difficulty.
- P: decision problems solvable in polynomial time.
- NP: decision problems whose yes-instances have certificates verifiable in polynomial time.
- $A \leq_p B$: a polynomial-time translation preserving the answer. Algorithms flow backwards (B easy $\Rightarrow$ A easy), hardness flows forwards (A hard $\Rightarrow$ B hard).
- P $\subseteq$ NP; whether they are equal is open.