Reductions Review

Mukul Agarwal

May 4, 2026

P and NP

The complexity class P (for polynomial) is (roughly) the set of problems which can be solved in polynomial time. The complexity class NP (for nondeterministic polynomial) is (roughly) the set of problems which have "certificates" that can be *verified* in polynomial time.

Problem 1. Which complexity class are the following problems in? (pick the smallest one you think is possible).

Problem	Class
Given a graph G, does G have a cycle?	P (do DFS)
Given a graph G, does G have a Hamiltonian cycle?	NP (in fact NP-hard)
Given numbers n and k, does n have a factor in $\{2, ..., k\}$?	NP[1]
Given a formula in CNF, is there a satisfying assignment?	NP (NP-hard)
Given a formula in DNF, is there a satisfying assignment?	P (find a satisfiable term)

We know that $\mathsf{P} \subseteq \mathsf{NP}$. But, is $\mathsf{P} = \mathsf{NP}$?

[1]Here, any factor in $\{2, ..., k\}$ can serve as a certificate.

NP-Hardness

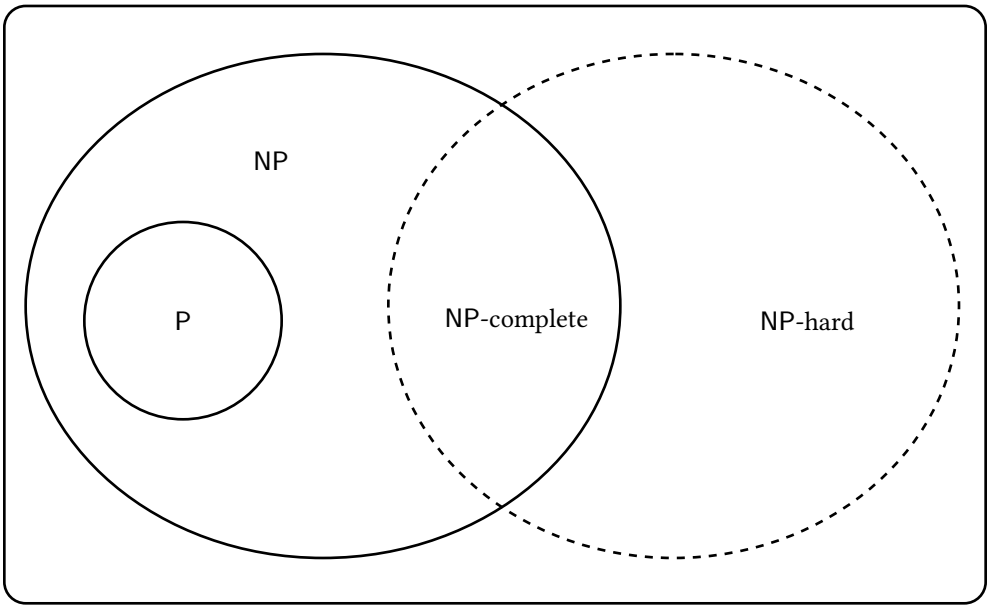
To try to address this problem, we have the idea of NP-hardness. A problem is said to be NP-hard if it can be used to solve problems in NP with polynomial extra work (this is called a reduction).

Theorem (Cook-Levin). The problem SAT is NP-hard.

SAT is particularly interesting because:

1. We can now show *other* things to be NP-hard by doing reductions with SAT.
2. It is also in NP. Other such problems that are in NP $\cap$ NP-hard are called NP-complete.

Here is a diagram of complexity classes (assuming P $\neq$ NP)!



Here are some other NP-hard problems (you will have this list on your exam too).[2]

1. Sat: Given a boolean formula in CNF form, decide if there is a satisfying assignment.
2. 3Sat: Given a boolean formula in CNF form with exactly 3 variables per clause, decide if there is a satisfying assignment.
3. CircuitSat: Given a boolean circuit, decide if there is a satisfying input.
4. MaximumIndSet: Given an undirected graph $G = (V, E)$, find the maximum cardinality set $S \subset V$ such that no two vertices in S share an edge.
5. Clique: Given an undirected graph $G = (V, E)$, find the maximum cardinality set $S \subseteq V$ such that every two vertices in S share an edge.
6. GraphColoring: Given an undirected graph $G = (V, E)$ and an integer k, assign each vertex one of k colors such that no edge has the same color endpoints.
7. HamCycle: Given an undirected (resp. directed) graph G, decide if there is a cycle containing every vertex exactly once.
8. HamPath: Given an undirected (resp. directed) graph G, decide if there is a Hamiltonian path (visiting every vertex exactly once).
9. (s, t)-HamPath: Given an undirected (resp. directed) graph G and two vertices s and t, decide if there is a Hamiltonian path (a path visiting each vertex exactly once) from s to t.
10. TravelingSalesman: Given a (directed or undirected) graph with positive edge lengths, compute a tour (a walk visiting each vertex at least once, starting and ending at the same vertex) of least weight.
11. SubsetSum: given a set X of positive integers and an integer T, decide whether X has a subset whose elements sum to T.
12. PlanarCircuitSat: Given a boolean circuit that can be embedded in the plane so that no two wires cross, is there an input that makes the circuit output true?
13. 1In3Sat: Given a 3-Cnf formula, is there an assignment of values to the variables so that each clause contains *exactly* one True literal?
14. NotAllEqualSat: Given a 3-Cnf formula, is there an assignment of values to the variables so that every clause contains at least one true literal and at least one false literal?

[2] I transliterated this list using Gemini, so it might have mistakes… if you email me I will fix it.

15. Planar3Sat: Given a 3-Cnf boolean formula, consider a bipartite graph whose vertices are the clauses and variables, where an edge indicates that a variable (or its negation) appears in a clause. If this graph is planar, the 3-Cnf formula is also called planar. The Planar 3Sat problem asks, given a planar 3-Cnf formula, whether it has a satisfying assignment.
16. Exact3DMatching: Let X, Y, Z be three finite collections of elements, and let $T \subseteq X \times Y \times Z$ be a subset of triples (x, y, z) consisting of one element x from X, one element y from Y, and one element z from Z. Find the maximum collection of disjoint triplets. (This problem is similar in spirit to bipartite matching, except the vertex set is divided into 3 parts, and there are "3 vertices in each edge with one from each part," so to speak.)
17. Partition: Given a set of n integers S, are there subsets A and B such that $A \cup B = S$, $A \cap B = \emptyset$, and
$$\sum_{a \in A} a = \sum_{b \in B} b?$$
18. 3Partition: Given a set S of $3n$ integers, can it be partition into n disjoint 3-element subsets, such that every subset has exactly the same sum?
19. SetCover: Given a collection of sets $S = \{S_1, ..., S_m\}$, find the smallest sub-collection of S_i's that contains all the elements of $\cup_i S_i$.
20. HittingSet: Given a collection of sets $S = \{S_1, S_2..., S_m\}$, find the minimum number of elements of $\cup_i S_i$ that hit every set in S. This problem is also a generalization of vertex cover.
21. LongestPath: given a (weighted or unweighted/uniformly weighted) graph G (either directed or undirected), what is the longest path (visiting each vertex at most once) in the graph?
22. Longest-(s,t)-Path: given a (weighted or unweighted/uniformly weighted) graph G (either directed or undirected) and two vertices s and t, what is the longest path (visiting each vertex at most once) from s to t in the graph?
23. SteinerTree: Given a weighted, undirected graph G with some of the vertices marked, what is the minimum-weight subtree of G that contains every marked vertex?
24. VertexCover: Let $G = (V, E)$ be an undirected graph. A set of vertices $S \subseteq V$ is a *vertex cover* if every edge $e \in E$ has at least one endpoint in S. Given a graph G and an integer k, is there a vertex cover with at most k vertices?

25. MAX2SAT: Given a Boolean formula in CNF, with exactly two literals per clause, find a variable assignment that maximizes the number of satisfied clauses.
26. MAXCUT: Given an undirected graph $G = (V, E)$, find a subset $S \subset V$ that maximizes the number of edges with exactly one endpoint in S.
27. MULTICOMMODITYFLOW: The input consists of either a directed or undirected graph, possibly with integral capacities, and k source-sink pairs $(s_1, t_1), ..., (s_k, t_k)$ each with an integral demand $d_i \in \mathbb{N}$. The problem is to pack d_i (s_i, t_i)-paths simultaneously for all i.

Problem 2. Recall the problem SetCover:

> SetCover(S): Given a collection of sets $S = \{S_1, ..., S_m\}$, find the smallest sub-collection of S_i's that contains all the elements of $\cup_i S_i$.

Prove that this problem is NP-hard.

Solution. Recall that the VertexCover problem is defined as:

> VertexCover(G): Let $G = (V, E)$ be an undirected graph. A set of vertices $S \subseteq V$ is a *vertex cover* if every edge $e \in E$ has at least one endpoint in S. Given a graph G and an integer k, is there a vertex cover with at most k vertices?

So, in this problem we are covering edges with vertices, and in SetCover we are covering elements with sets! This leads us to the following reduction:

VertexCover$(G = (V, E), k)$

1. Enumerate the vertices $V = \{v_1, ..., v_n\}$.
2. Create a collection of sets $S = \{S_i : i \in [n]\}$.
3. For each edge $e = \{v_i, v_j\} \in E$, add e as an element to S_i and to S_j.
4. Compute Opt $\leftarrow$ SetCover (S)
5. Return True iff $(\text{Opt} \leq k)$

Clearly this reduction is poly-time. It remains to prove correctness. For this, observe that it suffices to show that every set cover of S corresponds exactly to a vertex cover in G which has the same size (and vice versa).

($\Rightarrow$) Suppose we have a vertex cover of G with vertices $v_{i_1}, ..., v_{i_k}$. Then, consider $S' \subseteq S$ given as

$$S' = \{S_{i_1}, ..., S_{i_k}\}.$$

Observe that $\cup_i S_i = E$, so to show that S' is a set cover it suffices to show that each element of E is covered by it. Fix $e \in E$ and note that since e is an edge in G, it is covered by the vertex cover above. So $\exists j \in [k]$ such that v_{i_j} is an endpoint of e. But, by construction this means that $e \in S_{i_j}$ also! Hence, by univ. generalization, S' is a set cover.

($\Leftarrow$) (Very similar to the other direction...)

Now, note that if there were a vertex cover of size at most k then there would also be a set cover of size at most k, and in particular, the *smallest* set cover will have

size $\leq k$. Hence, in this case the algorithm is correct. OTOH if there is no vertex cover of size at most k, then the min-set-cover will correspondingly have size $> k$, and the algorithm will be correct in this case as well.

Problem 3. Consider the following problems.

> ExactDagDirectedPath$(G = (V, E), \ell : E \to \mathbb{R}, W)$: Given a DAG G with weights ℓ, and a weight $W \geq 0$, decide if G has a path of weight exactly W.

For the ExactDagDirectedPath problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that the problem is NP -Hard.

Solution. This problem is NP-hard. We reduce from SubsetSum.

> SubsetSum(X, T): given a set X of positive integers and an integer T, decide whether X has a subset whose elements sum to T.

SubsetSum$(X = [x_1, ..., x_n], T)$

1. Create a graph $G = (V, E)$, weighted by ℓ.
2. Add vertices $v_1, ..., v_n, v_{n+1}$, as well as vertices $c_1, ..., c_n$, and vertices $d_1, ..., d_n$
3. For each x_i add an edge $(v_i \to c_i)$ weighted x_i, and edges $(v_i \to d_i), (c_i \to v_{i+1}), (d_i \to v_{i+1})$, each weighted 0.
4. Return ExactDagDirectedPath(G, ℓ, T)

Clearly, the graph created has $O(n)$ vertices and $O(n)$ edges, and the construction proceeds in polynomial time. So it is only left to prove correctness.

For correctness, we first show that the graph is indeed a DAG. Let us create a function $f : V \to \mathbb{R}$ weighting each vertex s.t. $f(v_i) = i$, and $f(d_i) = f(c_i) = i + 1/2$. Notice clearly that each edge strictly increases in f, enforcing the fact that the graph is acyclic (a cycle would need to have at least *one* edge where the f-weight has to go down, a contradiction).

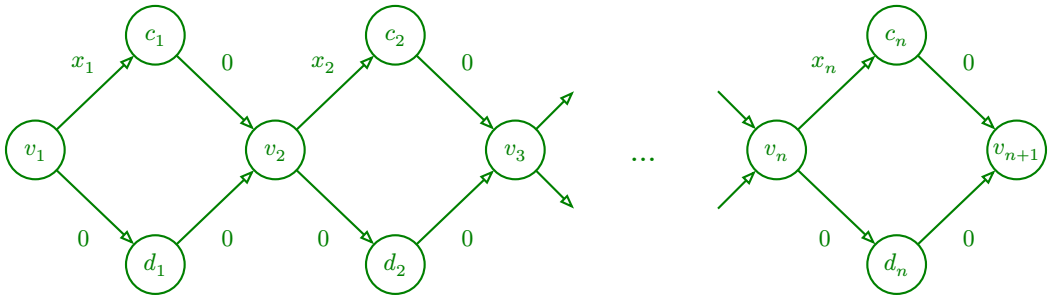
Notice that for a path, nonzero contributions to weight come exactly from edges of the form $v_i \to c_i$, which correspond exactly to x_i. So, if T is a feasible Subset-Sum using a sub(multi)set $A \subseteq X$, then we can construct paths as follows:

$$p_i = \begin{cases} v_i \to c_i \to v_{i+1} & \text{if } x_i \in A \\ v_i \to d_i \to v_{i+1} & \text{otherwise} \end{cases}$$

and create a final path p which is simply the concatentation of paths $p_1 \parallel p_2 \parallel \cdots \parallel p_n$. Observe that p_i contributes x_i to the total weight if $x_i \in A$ and 0 weight otherwise; hence the total weight of the path is $T = W$.

OTOH suppose a path of weight $W = T$ is feasible in G. Then, if we simply consider the edges along the path with nonzero weight, we recover exactly a subset of X having sum T, as desired.

Here is a diagram of the graph we construct, for illustrative purposes. Observe how the choice to "include" (resp. "not include") a particular x_i in a subset[3] corresponds *exactly* to the choice of traversing from $v_i \to v_{i+1}$ using c_i (resp. d_i) as an intermediate vertex.



[3]Remember the HW0/PSO problem we had about this?