# 15 Polynomial reduction

**Overview.** Not surprisingly, there are many problems that are extremely hard. In particular, they do not admit polynomial time algorithms. However, for many fundamental discrete computational problems, in optimization, AI/ML, combinatorics, logic, etc, we do not know of any polynomial algorithms; on the other hand we cannot prove that no polynomial time algo exists as well.

Later people realized that a large class of problems in this "gray area" has been shown to be "equivalent" – a polynomial-time algorithm for any of them would imply the existence of a polynomial-time algorithm for all of them. These are called NP-complete problem.

In other words, all of these open questions are really a **single** open question, a single type of complexity that we don't yet fully understand.

NP-completeness essentially means "computationally hard for all practical purpose, though we cannot prove it".

Discovering that a problem is NP-complete provides a compelling reason to stop searching for an efficient algorithm with provable guarantees. On the other hand, it somewhat justifies the search of good heuristics (algo without guarantees)

**Polynomial reduction.** We want to say problem $X$ is at least as hard as problem $Y$. We show this by the notion of "reduction" – if we have a blackbox capable of solving $X$, then we could also solve $Y$.

$Y \leq_p X$: arbitrary instance of $Y$ can be solved using a polynomial number of standard computational steps (reduction overhead) + a polynomial number of calls to a blackbox that solves problem $X$.

$Y \leq_p X$ reads "$Y$ is polynomial-time reducible to $X$" or "$X$ is as hard as $Y$ with respect to polynomial time".

**The contrapositive.** $Y \leq_p X$ gives: if $Y$ cannot be solved in polynomial time, then $X$ cannot be solved in polynomial time. *Be aware of the direction*.

**Problems.**

- *Independent Set:* Give a graph $G = (V, E)$, in the **search version**, the problem is to find a subset $S \subseteq V$ of the *maximum* size such that for any pair $u, v \in S$, there is *no* edge between $u$ and $v$.

  In the **decision version**, we also give a parameter $k$ as an input, and ask whether there is a subset $S \subseteq V$ of size at least $k$ such that for any pair $u, v \in S$, there is *no* edge between $u$ and $v$.

  In the following, when we talk about reductions we always use the decision version of the problem (same for other problems).

- *Vertex Cover (decision version):* Give a graph $G = (V, E)$ and a parameter $k$, we ask whether there is a subset $S \subseteq V$ of size at most $k$ such that for any edge $e = (u, v) \in E$, either $u \in S$ or $v \in S$.

- *Set Cover (decision version):* Given a set $U$ of $n$ elements, a collection $S_1, \ldots, S_m$ of subsets of $U$, and a number $k$, does there exist a collection of at most $k$ of these sets whose union is equal to $U$?

- *SAT and* 3*-SAT:* We are given a set $X$ of $n$ Boolean variables $x_1, \ldots, x_n$; each can take the value 0 or 1 (equivalently, "false" or "true"). By a *term* over $X$, we mean one of the variable $x_i$ or its $\bar{x}_i$. Finally, a *clause* is simply a disjunction of distinct terms $C_j = t_1 \vee t_2 \vee \cdots \vee t_\ell$, each $t_i \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$. We say the clause has length $\ell$ if it contains $\ell$ terms. For the SAT problem, we want to determine whether there is a truth assignment for $X$ (that is, to assign a value 0 or 1 to each $x_i$), such that $C_1 \wedge C_2 \wedge \cdots \wedge C_k$ evaluates to be 1.

  For 3-SAT, the only difference is that we require that each clause contains exactly 3 terms.