Reductions and NP-completeness

Last modified: 2026-10-30 13:54:17 (PDT)

A reduction turns every question about one problem into a question about another, so that a fast algorithm for the second gives one for the first. Reductions compare the difficulty of the problems in NP, and single out the hardest of them.

Polynomial-time reductions

Definition 0.1 (Polynomial-time reduction)**.** A decision problem L **reduces in polynomial time** to a decision problem L', written $L \leq_p L'$, if there is a function $f\colon \{0,1\}^* \to \{0,1\}^*$, computable in polynomial time, such that for every $x \in \{0,1\}^*$,

$$x \in L \quad \text{if and only if} \quad f(x) \in L'$$

(Arora and Barak 2009, sec. 2.2).

Theorem 0.1 (Reductions carry polynomial-time algorithms back)**.** *If $L \leq_p L'$ (Definition 0.1) and L' is in P, then L is in P.*

Proof

Proof. Let f be the reduction. A big-O bound holds from some n_0 on, and raising its constant to cover the finitely many lengths below n_0 turns it into a bound for every length. So there are constants $a, b \geq 1$ and $c, d \geq 0$ such that, on inputs of every length $n \geq 0$, computing f takes at most $a(n+1)^c$ steps, and some algorithm decides L' within $b(n+1)^d$ steps.
On input x of length n, compute $f(x)$ and then decide whether $f(x) \in L'$. The answer is correct because $x \in L$ exactly when $f(x) \in L'$.
Each step writes at most some fixed number k of output bits, so $|f(x)| \leq ka(n+1)^c$. Since $(n+1)^c \geq 1$,

$$
\begin{aligned}
|f(x)| + 1 &\le ka(n+1)^c + 1 \\
&\le (ka+1)(n+1)^c.
\end{aligned}
$$

The two stages together take at most

$$
\begin{aligned}
a(n+1)^c + b(|f(x)|+1)^d &\le a(n+1)^c + b[(ka+1)(n+1)^c]^d \\
&= a(n+1)^c + b(ka+1)^d(n+1)^{cd}
\end{aligned}
$$

steps, which is $O(n^{\max(c,cd)})$, a polynomial bound. □

Theorem 0.2 (Reductions compose)**.** *If $L \le_p L'$ and $L' \le_p L''$ (Definition 0.1), then $L \le_p L''$.*

Proof

Proof. Let f reduce L to L' and g reduce L' to L''. For every x, $x \in L$ exactly when $f(x) \in L'$, which is exactly when $g(f(x)) \in L''$. Computing $f(x)$ and then g of the result takes polynomial time, by the same count as in the proof of Theorem 0.1, with the algorithm for g in place of the algorithm for L'. So $x \mapsto g(f(x))$ reduces L to L''. □

NP-completeness

Definition 0.2 (NP-hard and NP-complete)**.** A decision problem L' is **NP-hard** if $L \le_p L'$ (Definition 0.1) for every decision problem L in NP. It is **NP-complete** if it is NP-hard and also in NP (Arora and Barak 2009, sec. 2.2).

Theorem 0.3 (One NP-hard problem in P puts all of NP in P)**.** *If any NP-hard problem (Definition 0.2) is in P, then every problem in NP is in P.*

Proof

Proof. Let L' be NP-hard and in P, and let L be any problem in NP. Then $L \le_p L'$ by Definition 0.2, so L is in P by Theorem 0.1. □

Theorem 0.4 (New NP-complete problems from old)**.** *If L is NP-complete (Definition 0.2), $L \le_p L'$ (Definition 0.1), and L' is in NP, then L' is NP-complete.*

Proof

Proof. L' is in NP by assumption. For any L_0 in NP, $L_0 \leq_p L$ because L is NP-hard, so $L_0 \leq_p L'$ by Theorem 0.2. So L' is NP-hard, and hence NP-complete. □

Definition 0.3 (The satisfiability problem SAT)**.** The decision problem **SAT** asks: given a Boolean formula built from variables with AND, OR and NOT, is there an assignment of true and false to its variables that makes the formula true?

Theorem 0.5 (Cook–Levin theorem)**.** *SAT is NP-complete (Definition 0.2).*

Theorem 0.6 (Independent set is NP-complete)**.** *INDSET is NP-complete (Definition 0.2) (Arora and Barak 2009, sec. 2.4).*

Remark 0.1 (Where NP-completeness comes from). Stephen Cook and Leonid Levin proved Theorem 0.5 independently in the early 1970s; Arora and Barak (2009, sec. 2.3) give a proof. Once one problem is known to be NP-complete, Theorem 0.4 gives others, one reduction at a time. Arora and Barak (2009, sec. 2.4) prove Theorem 0.6 this way, reducing a version of SAT to INDSET.

Worked reductions

Definition 0.4 (The clique problem CLIQUE)**.** A *clique* in a graph is a set of vertices every two of which are joined by an edge. The decision problem **CLIQUE** asks: given a graph G and an integer $k \geq 0$, does G have a clique of k vertices?

Definition 0.5 (Complement of a graph)**.** The **complement** $\bar{G}$ of a graph G has the same vertices as G, and joins two distinct vertices exactly when G does not.

Example 0.1 (From independent sets to cliques)**.** A set S of vertices is independent in G exactly when no two vertices of S are joined in G, which is exactly when every two vertices of S are joined in the complement $\bar{G}$, that is, when S is a clique in $\bar{G}$. So the function $f(G, k) = (\bar{G}, k)$ satisfies

$$(G, k) \in \text{INDSET} \quad \text{if and only if} \quad (\bar{G}, k) \in \text{CLIQUE}.$$

With m vertices, computing $\bar{G}$ flips the $m(m-1)$ off-diagonal entries of the adjacency

matrix, which takes $O(m^2)$ steps. So INDSET $\leq_p$ CLIQUE (Definition 0.1). CLIQUE is in NP, since a clique is its own certificate, as in the INDSET example. INDSET is NP-complete (Theorem 0.6), so CLIQUE is NP-complete too, by Theorem 0.4.

Definition 0.6 (The vertex cover problem VERTEX-COVER)**.** A *vertex cover* of a graph is a set of vertices that contains at least one end of every edge. The decision problem **VERTEX-COVER** asks: given a graph G with m vertices and an integer $k \geq 0$, does G have a vertex cover of at most k vertices?

Exercise 0.1 (From independent sets to vertex covers)**.**

1. Show that a set S of vertices is independent if and only if the set $\bar{S}$ of the vertices not in S is a vertex cover.
2. Use part 1 to give a polynomial-time reduction (Definition 0.1) from INDSET to VERTEX-COVER.

Solution

Solution 0.1.

1. S fails to be independent exactly when some edge has both its ends in S, which is exactly when some edge has no end in $\bar{S}$, that is, when $\bar{S}$ fails to be a vertex cover.
2. Map (G, k) to $(G, m-k)$ when $k \leq m$, and to a fixed no-instance, such as a single edge with $k = 0$, when $k > m$. If G has an independent set S of k vertices, then by part 1, $\bar{S}$ is a vertex cover of $m-k$ vertices. If G has a vertex cover C of at most $m-k$ vertices, then by part 1, $\bar{C}$ is independent with at least k vertices, and any k of them still form an independent set. So $(G, k) \in$ INDSET if and only if $(G, m-k) \in$ VERTEX-COVER. The map copies G and computes $m-k$, which takes polynomial time.

Hardness in data science

Remark 0.2 (Sparse regression is NP-hard). Best-subset selection looks for the regression with the fewest nonzero coefficients that fits the data to a given tolerance. Its decision version asks: given a matrix $\mathbf{X}$, a vector y, a tolerance ϵ and an integer k, is there a vector β with at most k nonzero entries and $\|\mathbf{X}\beta - y\|_2 \leq \epsilon$? Natarajan (1995) showed, by a reduction from an NP-complete problem, that this decision problem is NP-hard

(Definition 0.2). An algorithm that finds a sparsest β would decide it, by checking whether that β has at most k nonzero entries. If such an algorithm ran in polynomial time, Theorem 0.3 would put every problem in NP into P, and since P is contained in NP, P would equal NP. That is one reason practical methods use greedy forward selection or replace the count of nonzero coefficients with a convex penalty, as the lasso does.

References

Arora, Sanjeev, and Boaz Barak. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press. https://doi.org/10.1017/CBO9780511804090.

Natarajan, B. K. 1995. "Sparse Approximate Solutions to Linear Systems." *SIAM Journal on Computing* 24 (2): 227–34. https://doi.org/10.1137/S0097539792240406.