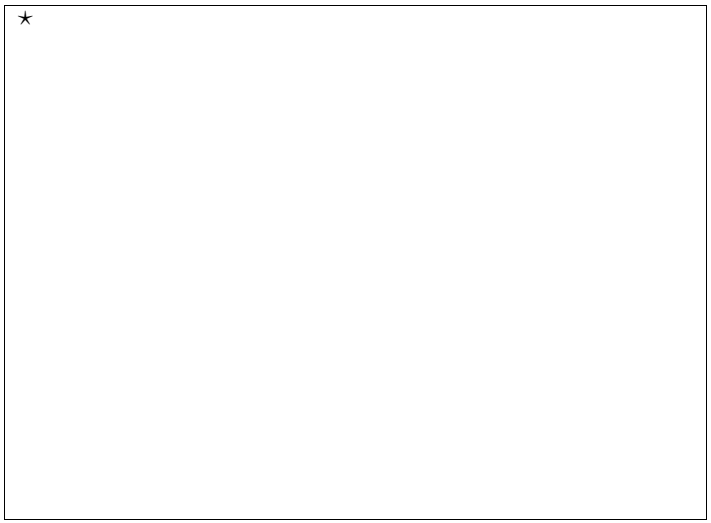
Reductions

Suppose we have a procedure f that transforms any instance α of A into some instance β of B with the following properties:

1. f takes polynomial time.
2. The answers are the same. That is, $\alpha = 1$ if and only if $\beta = 1$.

Then f is a polynomial time **reduction algorithm**

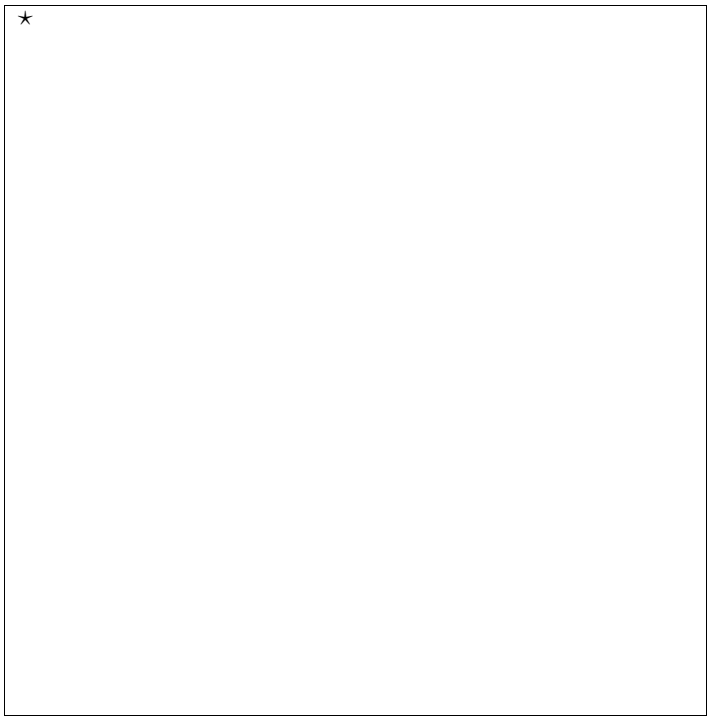
A problem Q reduces to another problem Q' if Q is "no harder to solve" than Q'.



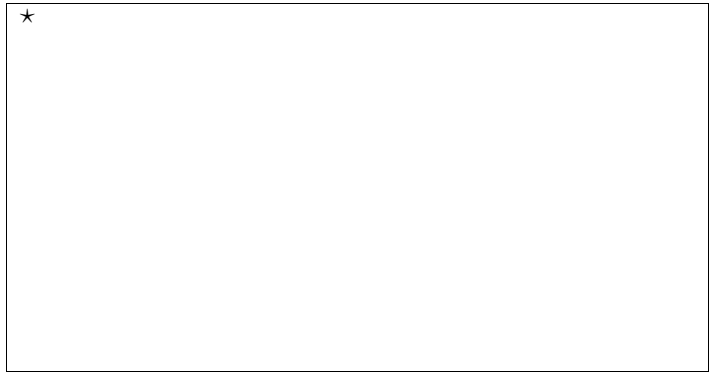
Definition: Let L_1, L_2 be languages. We say $L_1 \leq_p L_2$ ("L_1 is polynomial time reducible to L_2") if there is a polynomial time computable function f mapping binary strings to binary strings, such that

$$x \in L_1 \text{ if and only if } f(x) \in L_2.$$

If we want to decide if an instance x is in the language L_1, we can first "reduce" this problem to the problem of deciding if $f(x)$ is in L_2, and use the answer to determine whether x is in L_1.



The Hamiltonian Cycle problem can be reduced to the Traveling Salesman Problem, (HAM CYCLE $\leq_p$ TSP). We need to describe an efficient algorithm that, given an instance G =(V,E) of the Hamiltonian Cycle Problem, outputs an instance I of the Traveling Salesman Problem.



Furthermore, it should be that $G \in$ Ham-Cycle if and only if $I \in$ TSP. To do this, we must specify (i) what are the cities of the instance I, (ii) what are the costs between each pair of cities, and (iii) what is the bound k (recall the TSP problem asks whether there is a tour of cost at most k).

Let the nodes of G be $1, 2, \ldots, n$. The algorithm constructs I as follows. First, the cities are the nodes of G, namely $1, 2, \ldots n$. Second, the distance d_{ij} from city i to city j is defined as follows:

$$d_{ij} = \begin{cases} 1, & \text{if } \{i, j\} \text{ is in } E \\ 2, & \text{otherwise} \end{cases}$$

Finally, the bound k is defined to be n.

⋆ HAM-CYCLE $\leq_p$ TSP.

⋆ HAM-CYCLE $\leq_p$ TSP.

NP-Completeness

The NP-complete languages are intuitively the hardest problems in NP. Formally,

Definition 2 *L is NP-Complete if:*

1. *L is in NP, and*
2. *for all L' in NP, $L' \leq_P L$.*

Let NPC be the class of NP-Complete languages. If a language L satisfies property 2, bu not necessarily property 1, we say that L is **NP-hard**.

Claim 10 *If L is in NPC and L is in P, then $NP = P$.*

⋆ Proof:

Thus, proving that a problem is NP-complete provides strong evidence that the problem does not does not have an efficient algorithm.

Cook proved that a logic problem called 3-CNF-SAT is NP-complete. We will define the 3-CNF-SAT shortly. Given Cook's result, the following claim provides us with a powerful method by which we can show that other problems are also NPC.

Claim 11 *: If language L_1 is NPC, $L_1 \leq_p L_2$, and L_2 is in NP then L_2 is NPC.*

⋆ Proof:

Thus, to prove that a problem L is NP-complete, it is sufficient to show that

1. the problem is in NP (this is usually the easy part; you just need to find an efficient verification algorithm for the problem),
2. show that some problem L' already known to be NP-complete is reducible to L.

3-CNF-SAT

A famous NP-complete problem is the 3-CNF-SAT problem, which is defined as follows. Given a boolean formula of the form:

$$(x_1 \vee x_5 \vee \bar{x}_{10}) \wedge (x_3 \vee \bar{x}_7 \vee \bar{x}_2) \wedge (x_2 \vee x_4 \vee x_{10}) \ldots$$

where x_i is a variable that can be either true or false, can we assign the variables true/false values such that the formula evaluates to true (is satisfied)?

⋆ 3-CNF-SAT

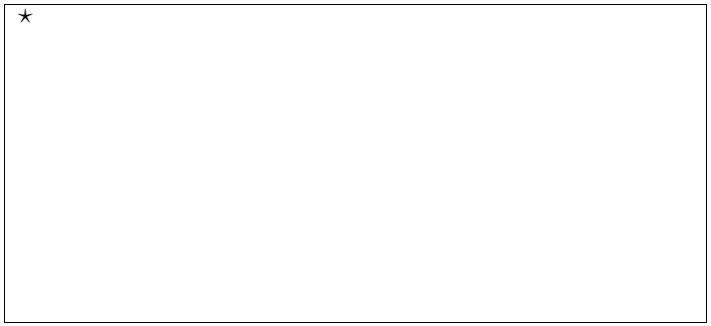
More generally, a 3-CNF-SAT formula is the conjunction ("and," or "∧") of the clauses, each of which is a disjunction ("or," or "v") of three *literals*, where a literal is a variable or its complement.

A naive algorithm that solves this problem is to enumerate all of the 2^n possible truth assignments and for each, test if the formula is true. Output "yes" if and only if a truth assignment is found that satisfies the formula. Corresponding to this algorithm is a natural polynomial time verification algorithm for 3-CNF-SAT. The verification algorithm takes two inputs: a boolean formula ϕ that is an instance of 3-CNF-SAT and a truth assignment T to the variables of ϕ. The algorithm tests whether T satisfies the formula and outputs "yes" if so.

The Clique Problem

We now introduce a new problem on graphs, namely the Clique problem. We will show that the Clique problem is NP-complete via a reduction from 3-CNF-SAT. Thus, the Clique problem is the first problem we will prove is NP-complete.

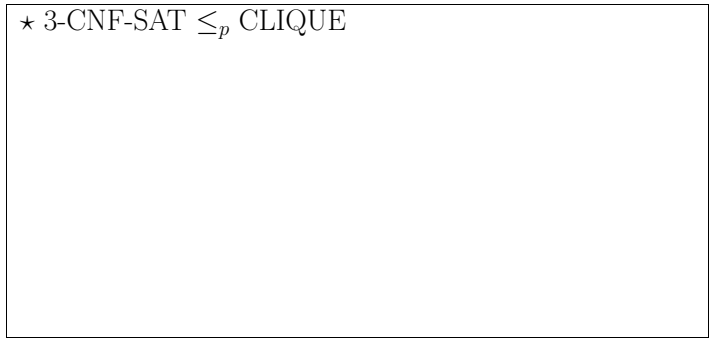
The Clique Problem is as follows: Given an undirected graph $G = (V, E)$, and a number k, does G have a clique of size greater than or equal to k? A clique is a subset V' of V such that every pair of nodes in V' is connected by an edge of E.



Again, this is an example of a problem for which there is no known efficient algorithm; the best known algorithm requires exponential time. A verification algorithm is as follows: the two inputs are an instance (G, k) of the clique problem and subset V' of the nodes of G. The algorithm checks that the size is at least k and also that every pair of nodes in V' is connected by an edge of E.

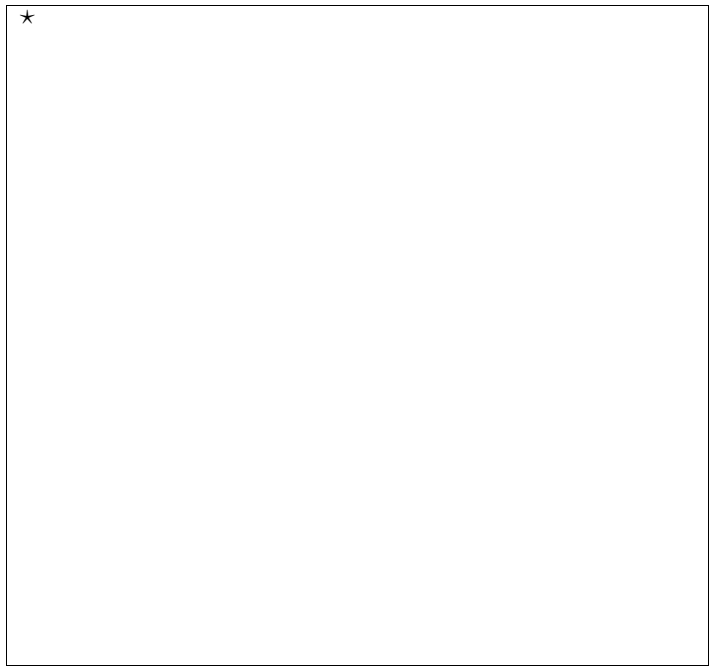
A Reduction from 3-CNF-SAT to CLIQUE

We now describe a reduction from the 3-CNF-SAT problem to the CLIQUE (3-CNF-SAT $\leq_p$ CLIQUE).



Let ϕ be an instance of 3-CNF-SAT, $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, where each C_i is of the form $C_i = l_1^i \vee l_2^i \vee l_3^i$. Each literal l_j^i is a variable from the set $\{x_1, x_2, \ldots, x_n\}$ or its complement. We describe an efficient algorithm that reduces ϕ to an instance $(G = (V, E), K)$ of CLIQUE. That is, the algorithm takes ϕ, an instance of the 3-CNF-SAT problem, and outputs a corresponding graph G and bound K, such that ϕ is satisfiable if and only if G has a clique of size K.

For each clause C_i there are three nodes in the graph, one per literal in the clause. Also, there is an edge between node l_i^r and l_j^s iff $r \neq s$ and $l_i^r \neq \bar{l}_j^s$. The graph $G = (V, E)$ has now been described. The bound K is chosen to be equal to m, the number of clauses of ϕ.



This reduction algorithm can map an instance of 3-CNF-SAT to an instance of CLIQUE in polynomial time. It remains to show that ϕ is in 3-CNF-SAT (i.e. that ϕ is a "yes"-instance of 3-CNF-SAT) iff $(G = (V, E), K)$ is in CLIQUE.

We first prove that ϕ is in 3-CNF-SAT implies $(G = (V, E), K)$ is in CLIQUE. We need to show that G has a clique of size K. Since ϕ is in 3-CNF-SAT, there is a truth assignment to the variables that sets at least one variable in each clause to true. Pick exactly one true literal per clause, and consider the m nodes correspponding to these literals in the graph. These nodes form a clique, because all chosen literals are true, hence not complements of each other, and are in different clauses. All of these literals are nodes in the graph that are connected to each other.

Secondly, we need to prove implication to the left, i.e. that $(G = (V, E), K)$ is in CLIQUE implies ϕ is in 3-CNF-SAT. Let V' be a clique of size K in G. We need to show that ϕ has a satisfying truth assignment. We know that exactly one node of V' corresponds to each clause of ϕ. If l_i^s is in V' then if $l_i^s = x_r$, set x_r to true. If $l_i^s = \bar{x}_r$ then set x_r to false. This defines a consistent truth assignment that sets at least one literal in each clause to true. Therefore, ϕ is satisfiable.

Proving NP-completeness

Summarizing from previous lectures, here are the steps needed to prove that a decision problem Π is NP-complete.

1. Show that Π is in NP, i.e. describe a verification algorithm for the problem Π, explain why your algorithm is correct, and show that your verification algorithm runs in polynomial time. (This is usually the "easy" part of the process, so it is easy to forget to do it!)

2. Pick a problem Π' already known to be NP-complete, and show that $\Pi' \leq_p \Pi$. That is, describe a reduction, or mapping, f that maps instances of Π' to instances of Π. Things to be careful about:

   (a) Make sure your reduction is going in the right direction. It is not correct to reduce your problem Π to a problem that is already known to be NP-complete. (If you do this, you are showing that your problem can be solved by mapping it to a problem already

known to be hard. This does not show that your problem is hard, because it does not rule out the possibility that there is a more simple way to solve your problem.)

(b) Show that your reduction runs in polynomial time. In particular, make sure that your reduction does not depend on knowing the solution to the instances involved.

(c) Show that your reduction is correct. That is, show that for each instance x of Π', (i) if x is a "yes"-instance of Π', then $f(x)$ is a "yes"-instance of Π, and (ii) if $f(x)$ is a "yes"-instance of Π, then x is a "yes"-instance of Π'. Remember to do both of these parts.

THE END

"We shall not cease from exploration
And the end of all our exploring
Will be to arrive where we started
And know the place for the first time."

T.S. Eliot

THE BEGINNING

"A new breed of explorers are now trekking the continent of analysis, bringing with them new tools and new techniques. To progress they have jettisoned some of the old ways and the old ideas. They keep secrets; they prove things without giving proofs; and they use randomness profusely. Having discovered a mountain in the way of progress they've lightened their conceptual burdens; they've dropped some of the classical ideas about algorithms to get fast solutions. These brave souls are prepared to have their algorithms sometimes fail, sometimes lie, and sometimes never return! It's an exciting time. Welcome to the beginning."

Gregory Rawlins