COMPSCI 330: Design and Analysis of Algorithms

Lecture 22: More Reductions

Lecturer: Rong Ge *Scribe: Mo Zhou*

Overview

In this lecture, we review the concept of NP, NP-hard and NP-complete, and introduce a prototypical NP-Complete problem, CIRCUIT-SAT. We then prove the NP-completeness of INDEPENDENT SET, CLIQUE and 3-SAT by reduction.

22.1 NP, NP-hard and NP-complete

Recall that we say a problem A reduces to B in polynomial time if there is a polynomial time algorithm that can transform an instance X of problem A to an instance Y of problem B in polynomial time, such that if the answer to X is YES, answer to Y is also YES, and if answer to X is NO, answer to Y is also NO. Such an algorithm is called a polynomial-time reduction. Then, we have if A reduces to B in polynomial time and B can be solved in polynomial time, then A can also be solved in polynomial time.

Definition 22.1 (NP) *A problem is in NP if its solution can be verified in poly-time.*

In computational complexity theory, NP is the class of (decision) problems whose solution can be verified in poly-time. Actually, there are harder problems that are not in NP, such as Halting Problem and Playing Chess. Halting Problem is that given the code of a program, check if it will terminate or not. This problem is not even solvable; Playing chess is that given a chess board with $n \times n$ size, with $4n$ chess pieces, decide whether the first player can always win. This is believed to be PSPACE complete, and very unlikely to be in NP.

Definition 22.2 (NP-hard) *A problem is NP-hard if all problems in NP can be reduced to it in poly-time.*

We can see that NP-hard problems are "harder" than all problems in NP. By reduction, or more specifically reducing problem B to problem A, we mean that given a "blackbox" solver that solves A, we can also solve B by transforming the instance of B to an instance of A, and then transform the solver's solution to the instance of A to a solution to the instance of B. Observe that if A can be reduced to B and B can be reduced to C, then A can be reduced to C.

Definition 22.3 (NP-complete) *A problem is NP-complete if it is in NP and is NP-hard.*

We can see that NP-complete problems are the hardest problem in NP. The reason is that if A is in NP, and B is a NP-complete problem, then A can be reduced to B. Therefore, if any NP-complete problem has a polynomial time algorithm, then P = NP. See Figure 22.1 for the relation between P,NP,NP-complete and NP-hard.

From here, the road map to prove the NP-completeness of a problem is almost clear: we want to 1) show that it is in NP and 2) show that all problems in NP can be reduced to it. The latter step seems unfeasible at the moment, because there could be infinitely many problems in NP. In fact, we may instead prove that

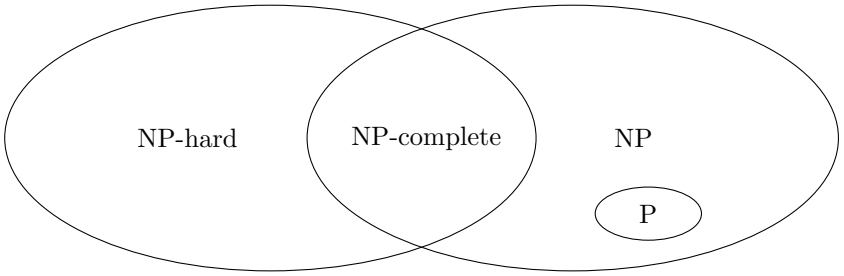


Figure 22.1: Relationships between P,NP,NP-complete and NP-hard (When P≠NP)

an NP-complete problem can be reduced to it; if we can do that, we know all problems in NP can be reduced to it by the definition of NP-hard. Which NP-complete problem, then, should be reduced to it? Fortunately, as we will demonstrate later, the following claim or observation says we don't have to worry about this issue.

Claim 22.4 *All NP-complete problems can be reduced to each other.*

All NP-complete problems are equally "hard". Hence, to prove the NP-completeness of a problem, all we have to do is to 1) show that it is in NP and 2) show that *an* NP-complete problem can be reduced to it.

22.2 CIRCUIT-SAT: The First NP-Complete Problem

CIRCUIT-SAT is a decision problem that asks the following: Given a *Boolean circuit* with n inputs and 1 output, is there a possible input (represented by an n-bit binary string) that makes the output 1?

Theorem 22.5 (Cook-Levin) *CIRCUIT-SAT is NP-compete*

The proof of this theorem is beyond the scope of this course, but we will give you an (hopefully intuitive) idea here. First of all, it is in NP because, given an n-bit string, we can simply follow the circuit and compute the output in poly-time, and check if the output is indeed 1. As for the reduction, there is no better solver for a CIRCUIT-SAT instance than (the circuit of) the computer in front of you. Given any problem in NP, we can write a piece of program for it. (pick your favorite language) The compiler will turn it into a binary machine code, essentially an instance of CIRCUIT-SAT, execute it, and print the output of your program. This shows that all problems in NP can be reduced to CIRCUIT-SAT.

22.3 Reductions

In this section, we prove the NP-completeness of more problems by reduction. We first discuss the general recipe for reductions:

22.3.1 General Recipe for Reductions

In order to prove B is NP-hard, given that we know A is NP hard, we want to reduce A to B. A complete proof of reduction follows these steps:

1. Given an instance X of A, construct an instance Y of B.
2. Prove that if answer to X is YES, then answer to Y is also YES.
3. Prove that if answer to X is NO, then answer to Y is also NO. (Usually prove the contrapositive: if answer to Y is YES, answer to X is also YES)

We demonstrate this technique with the following examples

22.3.2 Reduction from INDEPENDENT SET to CLIQUE

Definition 22.6 (Independent set) *In an undirected graph, an independent set is a set of vertices such that no pair of vertices are connected by an edge.*

The decision problem INDEPENDENT SET asks the following question: given a graph G and a number k, does there exist an independent set of size k?

Definition 22.7 (Clique) *In an undirected graph, a clique is a set of vertices such that all pairs of vertices are connected by an edge.*

The decision problem CLIQUE asks the following question: given a graph G and a number k, does there exist a CLIQUE of size k?

We want to show that INDEPENDENT SET can be reduced to CLIQUE. Given an instance (G, k) of INDEPENDENT SET, we want to construct an instance (G', k') of CLIQUE. Note that in INDEPENDENT SET, we want to find vertices such that they are not connected. While in CLIQUE, we want to find vertices such that they are connected. The idea comes from the observations that if a set S is an independent set in G, then S is a clique in G', where G' is the *complement* of G. The complement of a graph G is a graph G' on the same vertices such that two distinct vertices of G' are adjacent if and only if they are not adjacent in G.

By definition of the complement graph, u, v is connected in G' iff u, v is not connected in G. Hence, S is an independent set in G iff S is a clique in G'. Hence, if the answer to INDEPENDENT SET is YES iff the answer to CLIQUE is YES. Hence, INDEPENDENT SET reduces to CLIQUE. You can see that the proof that CLIQUE reduces to INDEPENDENT SET will be largely the same, only the direction of the argument differs.

22.3.3 Reduction from 3-SAT to INDEPENDENT SET

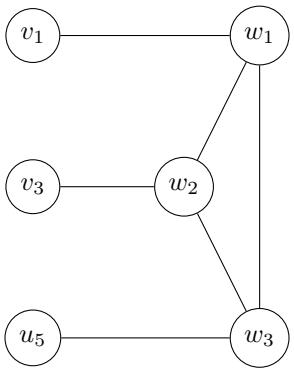
The decision problem 3-SAT asks the following question: given a a *Boolean formula* in conjunctive normal form where each clause contains at most three literals (ANDs of 3 ORs), is there a value (TRUE or FALSE) of variables so that the formula is *satisfied*? An example of input to 3-SAT is $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$. A solution to the problem assigns TRUE and FALSE to the variables. A clause is satisfied if one of its literals is satisfied (evaluates to TRUE). The formula is satisfied if all clauses are satisfied. When $x_1 = x_2 = x_3 = true$ and arbitrary x_4, the above formula is satisfied, thus the answer is yes. When the input is $(x_1 \vee x_2) \wedge (\bar{x}_1) \wedge (\bar{x}_2)$, the formula cannot be satisfied by any assignments of x_1, x_2. Thus, the answer to this input is no.

We want to show that 3-SAT can be reduced to INDEPENDENT SET. Given such Boolean formula, we want to construct an instance (G, k) of INDEPENDENT SET. The two problems look totally different at first glance, but in fact there are many way to construct. Our idea is to construct "Gadgets": for each concept in 3-SAT (variables, clauses), construct a part of instance of INDEPENDENT SET (some vertices/edges) so that we can connect the two problems.

Figure 22.2: Variable Gadget

For every variable x_i, construct two vertices u_i, v_i that connect to each other. We hope that when x_i is set to TRUE, u_i is in the independent set; x_i is set to FALSE, v_i is in the independent set. We call this a "variable gadget" (see Figure 22.2).

For every clause $C_j = (x_{i1} \vee x_{i2} \vee \bar{x}_{i3})$, construct a gadget consists of three vertices w_{j1}, w_{j2}, w_{j3} connecting to each other. We then connect (v_{i1}, w_{j1}), (v_{i2}, w_{j2}) and (u_{i3}, w_{j1}). We call this a "clause gadget." For example, for clause $(x_1 \vee x_3 \vee \bar{x_5})$, we construct



where the variable gadgets on the left also connect to other clause gadgets. With the help of this gadget, when the clause is not satisfied, the solver for INDEPENDENT SET cannot select any vertex in gadget. When clause is satisfied, the solver can select exactly one vertex in the gadget. This encourages all the clauses to be satisfied. More formally, let n denote the number of variables and the m the number of clauses. We will prove that a 3-SAT instance is satisfiable iff the corresponding graph constructed above has an independent set of size $k = n + m$.

If the 3-SAT instance is satisfiable, then we choose our independent set in the following way: 1) for each variable, if x_i is set to TRUE, take u_i; if x_i is set to FALSE, take v_i. This gives us n vertices that are guaranteed not connected. 2) for each clause, take the w vertex that corresponds to one of the satisfied literals. This gives us m vertices that are guaranteed not connected.

If there is an INDEPENDENT SET of size k in the graph constructed, by construction there can be at most 1 vertex from each variable gadget and 1 vertex from each clause gadget in the set. This means we must have exactly one vertex selected in each gadget. Setting x_i to TRUE if u_i is in the independent set and x_i to FALSE if v_i is in the independent set achieves a satisfying assignment: every variable is set to either TRUE or FALSE and every clause has one satisfied literal.

22.3.4 Summary

It is also known that CIRCUIT-SAT can be reduced to 3-SAT, therefore 3-SAT is also an NP-complete problem. Combine that with the reductions above, we have a complete picture of all the NP-complete problems we know so far. Since all of them are in NP:

- CIRCUIT-SAT is NP-complete (by Cook-Levin Theorem.)
- 3-SAT is NP-complete (because CIRCUIT-SAT can be reduced to it.)
- INDEPENDENT SET is NP-complete (because 3-SAT can be reduced to it.)

- CLIQUE is NP-complete (because INDEPENDENT SET can be reduced to it.)