Craig Silverstein
Gates 470
Phone: (72)3-1787
Office Hours: Wed 4:30–5:30, Fri 1–3, Gates 470
Section: Monday 2:15–3:15, 380-380W

Computer Science 154
School of Engineering
Stanford University

**Section Handout #8**

# $P$ and $NP$ and $NP$-completeness

The preferred way to think about things: A problem is in $P$ if it can be solved by a DTM in polynomial time. A problem is in $NP$ if a solution can be verified in polynomial time. (As in problem 1.3 in Problem Set 6, this implies the solution must be expressible in polynomial space.) Here's an example: COMPOSITE (if a number is not prime) is in $NP$: you give me two factors as a witness of compositeness, and in polynomial time I multiply them together to see if they really equal the input number. Another example: SAT is in $NP$ because you give me an assignment to all the variables, and I substitute it into the formula to make sure the formula evaluates to 1.

A problem is **NP-complete** if we can modify every problem in NP so it becomes a special case of our NP-complete problem. NP-complete problems are thus the "hardest" problems in NP, because if we could solve them in polynomial time we could solve any other problem in polynomial time. We take the time to change our problem into the NP-complete problem and use our fast algorithm on the result. Of course, our modification must be doable in polynomial time or the total time may be exponential. This process of changing problem A into a special case of problem B is called **reducing** problem A to B.

To show a problem A is NP-complete, we must show that we can change another problem B, already known to be NP-complete, into our problem. We thus get the following chain:

| | |
|---|---|
| since A is known to be NP complete: | everything reduces to A |
| we try to show: | A reduces to B |
| then, by transitivity: | everything reduces to B |

Don't forget, doing the reduction just shows the problem is $NP$-hard. You also must show it's in $NP$ to prove it's $NP$-complete!

# Reductions

Just like in the decidability case, we try to show that if we had an oracle for our problem, we could solve a known-to-be-hard problem. But in this case, the oracle isn't a decider, it's a poly-time-solver. That is, we prove that if we could magically solve our problem in polynomial time, that would mean we could solve some known-to-be-slow problem in polynomial time as well. By "known-to-be-slow," we mean $NP$-complete. (Really it's a misnomer: it should be "strongly-suspected-of-being-slow.") As in the decidability-reduction case, we start by transforming the input of the known-to-be-hard problem into input for our problem. Remember, though, that in this case the transformation must be doable in polynomial time!

Here's an example: Prove 3-CNF is $NP$-complete. We reduce from CNF. Remember that CNF means that the formula is an AND of OR clauses; 3-CNF means that all OR clauses have exactly three literals in it.

We start by transforming input from CNF to 3-CNF. We do this by making all clauses have three literals. If a clause has less than 3, we just repeat the first literal; obviously $(a \vee a \vee a)$ is true iff $(a)$ is true. If a clause has more than 3 literals, we do the following to reduce the number of literals by 1: replace $(a \vee b \vee c \vee d \vee \ldots)$ by $(a \vee b \vee s_1) \wedge (\overline{s_1} \vee d \vee s_2) \wedge (\overline{s_2} \vee e \vee s_3) \wedge \ldots$. We introduce new $s$ variables that connect the clauses (each clause introduces a new $s$ which takes on the form $\overline{s}$ in the next clause). Verify the two forms are equal, and also that it only takes linear time to convert from one form to another.

As a result, 3-CNF must be $NP$-hard. If we could solve it fast, we could, quickly, convert a CNF formula to a 3-CNF formulat and solve it quickly. The answer to this quick procedure would be the answer to the CNF problem, which we know is $NP$-complete. Thus our problem is $NP$-hard.

Our problem is $NP$-complete because it's in $NP$: give me a satisfying assignment and I can verify it in polynomial time.

How do we do reductions? In general, it's hard, but certain trends emerge. When reducing from one logic formula to another, we generally have to introduce new variables, and use their $a$ and $\overline{a}$ forms to force behavior we want. When reducing from one graph to another, we often add or delete edges or duplicate graphs. Changing between formulas and graphs and other types of problems usually involve gadgets that encode the new form we're trying to change into. This is kinda like the formulas that encoded Turing Machines in Cook's theorem.

## Hints on Homework 7

**(1)** This differs from vertex cover because it's ok if there's an edge between two vertices not in $S$. The point is not that every edge touches a vertex in our set, but rather that every vertex is connected to a vertex in our set via an edge. If the graph is a triangle, each vertex is itself dominating set but is not a vertex cover.

**(2)** What's the complement?

**(3)** Here the verifier is verifying that the certificate does *not* fit the language. For instance, PRIME is in $coNP$ with the verifier taking two numbers and verifying that their product is not the (supposedly prime) input (unless one of the numbers is 1).