

CSCI3390-Assignment 5.

due November 29

Once again, a lot of questions, this time about polynomial-time reductions and **NP**-completeness. And, again, you don't have to do all of them. A 'perfect' paper is 100 points. Every problem is worth 20 points.

1 Boolean satisfiability

1. Consider the propositional formula

$$(\bar{p} \vee \bar{q}) \wedge (p \vee r) \wedge (p \vee \bar{r}) \wedge (q \vee r).$$

Use the polynomial-time algorithm for 2-SAT to find a satisfying assignment for this formula. How many such assignments are there? (It is easy enough to write down an assignment—but I want you to apply the algorithm here and construct the graph.) What happens if we add the clause $\neg p \vee q$?

2. *In which I prove $\mathbf{P} = \mathbf{NP}$ and win a million dollars.* Just as every propositional formula is equivalent to a formula in conjunctive normal form, every propositional formula is also equivalent to a formula in *disjunctive* normal form. This is an OR of ANDs, where CNF is an AND of ORs. For example,

$$(p \wedge q \wedge \neg r) \vee (\neg p \wedge \neg q \wedge r)$$

is in DNF.

(a) Let DNF-SAT be the problem where the input is a formula in disjunctive normal form, and the output is yes if the formula is satisfiable, no otherwise. Show that DNF-SAT is in **P**—the algorithm is really simple.

(b) I shall now prove that $\mathbf{P}=\mathbf{NP}$. Given a formula in CNF, I convert it to DNF and apply the algorithm of part (a). This is now a polynomial-time algorithm for SAT. You're welcome. I'll take my million dollars now.

Surely something is wrong here...

3. *Boolean satisfiability for general formulas.* Suppose we ask the more general question: given a propositional formula ϕ , not necessarily in CNF, is it satisfiable? We know that this problem (which we will call CIRCUI-T-SAT, for reasons to be explained shortly) is in \mathbf{NP} , and since we know SAT, and even 3-SAT, are \mathbf{NP} -hard, we can conclude

$$\text{CIRCUI-T-SAT} \leq_P \text{3-SAT}.$$

The goal of this problem is to demonstrate this reduction without recourse to the Cook-Levin theorem, but by directly constructing a 3-CNF formula from ϕ . As the preceding problem might tip you off, it is not good enough to simply convert ϕ to CNF!

Here is the idea: We can represent any formula by a directed acyclic graph in which the source nodes are literals, and the interior nodes are labeled \vee, \wedge, \neg . This graph is called a *circuit*, because it is exactly like one of those digital circuits built from AND, OR and NOT gates. (A circuit is more general and usually more succinct than a formula, because we can use the same subcircuit's output several times without adding to the size.)

For example, the formula

$$(p \vee (q \wedge \bar{r})) \wedge (\bar{p} \vee q \vee \bar{r})$$

is represented by the circuit shown in Figure 1. Observe that an assignment of truth values to the variables of the formula propagates to an assignment of truth values to the gates; the original assignment is satisfying if the final gate (the one at the right) gets the value True.

(a) Show how to turn the circuit of Figure 1 into a 3-CNF formula that is satisfiable if and only if the original formula is. Your new formula should have the original variables p, q, r , as well as variables corresponding to each of the AND and OR gates. The clauses will give the relations that must hold between these gate variables. You should check that the satisfying assignments match up.

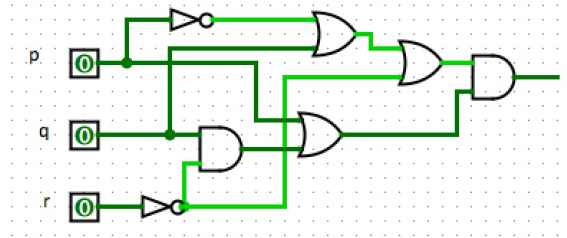


Figure 1: Circuit corresponding to the formula $(p \vee (q \wedge \bar{r})) \wedge (\bar{p} \vee q \vee \bar{r})$

(b) Describe the general procedure for turning a circuit into a 3-CNF formula. Why is this a polynomial-time reduction to 3-SAT?

2 Graph coloring

Just a reminder, k -COLORABLE is the problem whose input is a graph, and whose output is ‘Yes’ if and only if the graph can be colored using k colors, in such a manner that adjacent vertices do not get the same color.

4. Show that

$$3\text{-COLORABLE} \leq_P 4\text{-COLORABLE}.$$

Remember, you have to show, given a graph G , how to construct a graph G' such that G is 3-colorable if and only if G' is 4-colorable. (This is a very simple construction!) Make sure you explain why the construction can be carried out in polynomial time.

5. Suppose that we have a polynomial-time algorithm for 3-colorability. Does this actually imply the existence of a polynomial-time algorithm to find a coloring? The answer is yes, and we will give two approaches to this. First, use the fact stated in class, but not proved, that 3-COLORABLE is **NP**-complete, so if there is such an algorithm for colorability, then SAT is in **P**. Now apply some results we proved in class...
6. Here, and in the next problem, you will show how to find a coloring from an algorithm for existence of a coloring without appealing to the **NP**-completeness of 3-colorability. Consider the following problem. The input is a graph G that has been partially colored with three colors—that is, some subset of the

vertices have been colored blue, green, and red. The output is yes if and only if this partial coloring can be extended to a 3-coloring of all the vertices. Show that this problem is polynomial-time reducible to 3-colorability. (HINT: Add a little triangle gadget to to the graph.)

7. Now use the result of 6 to show that a polynomial-time algorithm for 3-COLORABLE implies that there is a polynomial-time algorithm that finds a coloring if one exists. Again, do this without appealing to **NP**-completeness.

3 Hamiltonian paths

8. This is a question about the proof of **NP**-completeness of directed Hamiltonian path: Draw the gadget for $(p \vee q) \wedge \bar{p} \wedge \bar{q}$. What does the graph tell you about the formula, and vice-versa?
9. The notes contain, as a final step of the proof of the **NP**-completeness of Hamiltonian path, a reduction from the directed version of the problem to the undirected version, through the use of little 3-vertex gadgets to represent each vertex of the directed graph. When I first saw this, I thought the middle vertex of these gadgets was unnecessary. But later I realized that if we used 2-vertex gadgets, connected the same way, then Hamiltonian paths in the two graphs would not correspond, so this transformation would not be a reduction from one problem to the other. Give an example that shows this.
10. We now go back to our original version of the problem, let's call it HAM, in which the input is an undirected graph G and a pair of vertices i, j , and the output is Yes if and only if there is a Hamiltonian path from i to j . There is also the *Hamiltonian Circuit* problem, call it HAMCIRCUIT, in which we ask whether a given graph has a Hamiltonian circuit. Show that

$$\text{HAM} \leq_P \text{HAMCIRCUIT},$$

and conclude that HAMCIRCUIT is also **NP**-complete. (This is another very easy reduction, much in the spirit of the very easy reduction in Problem 4.)

11. Since HAM is in **NP**, the Cook-Levin Theorem implies that it is polynomial-time reducible to SAT. Show this directly, by transforming a graph, together with designated start and end vertices i and j , into a propositional formula

that is satisfiable if and only if there is a Hamiltonian path from i to j . (HINT: Use variables $v_{k,\ell}$, where $v_{k,\ell}$ is true if and only if vertex ℓ is the k^{th} vertex on a Hamiltonian path. The clauses will encode the required properties of such a path.) Make sure that the end result is in CNF and that the construction can be carried out in polynomial time.

4 Clique

12. An instance of the CLIQUE problem is the graph G and an integer $k \leq |V(G)|$. The output is yes if and only if there is a clique with k -vertices—that is k vertices each adjacent to the other $k - 1$. Explain carefully why this problem is in **NP**. (You can even supply pseudocode, or real code, that verifies a witness efficiently.)
13. Suppose that instead we fix the value of k , so that it is not a part of the problem instance. For instance, we can set $k = 5$ and pose the problem of whether a given graph has a clique of size 5. Show that this problem is in **P**.
14. Show, what we claimed in class, that CLIQUE is **NP**-hard, by proving the existence of a polynomial-time reduction from SAT to CLIQUE. The idea is this: Each literal in each clause is a vertex of the graph (so, for example, if the literal \bar{p} occurs in two different clauses, then there will be two different vertices labeled \bar{p}). Draw an edge between every pair of vertices that belong to different clauses, *except* if one of the vertices is the negation of the other. Example: If the formula is

$$\bar{p} \wedge (\bar{p} \vee q) \wedge (p \vee q)$$

then the vertices can be named $\bar{p}_1, \bar{p}_2, q_2, p_3, q_3$. The graph will have edges between \bar{p}_1 and \bar{p}_2 , between \bar{p}_1 and q_3 , and many other pairs. But there will be no edge between \bar{p}_2 and q_2 , because they belong to the same clause, and no edge between \bar{p}_1 and p_3 , because they are negations of one another.

Prove that the graph has a k -clique, where k is the number of clauses, if and only if the formula is satisfiable. Use this to conclude (carefully!) that CLIQUE is **NP**-hard, and thus **NP**-complete. Draw pictures!

5 Compositeness

15. Unlike the other problems listed here, testing compositeness—or, what is the same thing, testing primality—*does* have a polynomial-time algorithm. But this does not mean that there is a polynomial-time algorithm for finding a witness to compositeness, that is, a factorization of the given number. In fact it is believed that this problem has no polynomial-time algorithm. Consider instead the following problem: The input consists of integers $k \leq N$, coded in binary or decimal, and the output is Yes if and only if N has a factor $1 < m < k$. Show that if there is a polynomial-time algorithm for *this* problem, then there is a polynomial-time algorithm for factoring an integer into smaller factors. (HINT: Binary search.)
16. Use the above to prove that if $\mathbf{P}=\mathbf{NP}$, then there is a polynomial-time algorithm for factoring. (Many cryptographic protocols depend for their security on factoring being a hard problem, so if $\mathbf{P}=\mathbf{NP}$, these systems fall apart.)