CS 170 HW 9

Due on 2018-04-02, at 11:59 pm

1 (★) Study Group

List the names and SIDs of the members in your study group.

2 (★) An introduction to 3-SAT

In this homework, we are going to delve deep into the properties of one of the integral problems in computer science, 3-**SAT**. Also, known as 3-*conjunctive normal form satisfiability*, 3-**SAT** is the decision problem of determining if a formula of a specific type has a solution.

A clause is a disjunction (an or) of literals (or their negations). A 3-**SAT** formula must be a conjunction (an and) of clauses with each clause containing *at most* 3 literals.

For example, the following is a 3-**SAT** clause.

$$\varphi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2).$$

It is satisfiable because $x_1 = 1, x_2 = 0, x_3 = 0$ is *a* satisfying assignment. It is not necessarily unique.

Show that 3-**SAT** is in NP.

Solution: A sufficient witness is an assignment for each of the literals $x_i = \alpha_i$. Then, it suffices to check if each of the clauses is satisfied, which can be done in a linear time in the number of clauses. Furthermore the number of clauses is at most $O(n^3)$ where n is the number of variables.

3 (★★★) Reduction to (3,3)-SAT

As it turns out 3-**SAT** is also NP-complete, meaning that all NP problems have a polynomial time reduction to 3-**SAT**. Let's first explore a simple one, a toy problem called (3,3)-**SAT**. (3,3)-**SAT** is the decision problem of determining whether there is a satisfying assignment for a 3-**SAT** formula in which each literal or its negation appears *at most* 3 times across the entire formula. Notice that (3,3)-**SAT** is reducible to 3-**SAT** because every formula that satisfies the (3,3)-**SAT** constraints satisfies those for 3-**SAT**. We are interested in the other direction.

Show that 3-**SAT** is reducible to (3,3)-**SAT**. By doing so, we will have eliminated the notion that the "hardness" of 3-**SAT** was in the repetition of variables across the formula.

Describe your reduction precisely. Assert the correctness and argue that the reduction can be computed in polynomial time. No pseudocode.

Solution: Assume that the variable x_i or $\neg x_i$ appears $k > 3$ times. Let us replace the k occurances of x_i with $x_i^{(j)}$ for $j = 1, \ldots, k$. We now have used each literal $x_i^{(j)}$ exactly once. We additionally add the following two clauses for each $j = 1, \ldots, k-1$:

$$x_i^{(j)} \vee \neg x_i^{(j+1)}, \qquad \neg x_i^{(j)} \vee x_i^{(j+1)}$$

Notice that both clauses are only satisfied if $x_i^{(j)} = x_i^{(j+1)}$. Adding all such clauses enforces that in any satisfying assignment, $x_i^{(1)} = x_i^{(2)} = \ldots = x_i^{(k)}$.

Make the replacement for each literal that occurs more than 3 times ensures that all literals in the new formula occur at most 3 times. This is only a polynomial increase in the size of the problem description.

A satisfying assignment $\{x_i = \alpha_i\}$ for the original formula can be translated to an assignment for the new formula by setting $x_i^{(j)} = \alpha_i$ by the previous argument. For the other direction, a satisfying assignment for the new formula must have $x_i^{(1)} = x_i^{(2)} = \ldots = x_i^{(k)} = \alpha_i$, so we can set $x_i = \alpha_i$ in the original formula.

4 (★★★★) Equivalence of Decision and Search

A powerful property of the class NP is the equivalence of decision and search. In the context of 3-**SAT**, it says that *finding* a satisfing assignment for a 3-**SAT** formula is polynomial-time reducible to *deciding* if one exists. (Convince yourself that the opposite direction is definitionally true.)

Formally, show that Search 3-**SAT** is reducible to Decision 3-**SAT**.

- Search 3-**SAT**. Input: 3-**SAT** formula φ on variables $x_1, x_2, \ldots, x_n$. Output: A satisfying assignment (i.e. $x_1 = \alpha_1, x_2 = \alpha_2, \ldots$) if one exists. Othewise, $\perp$.
- Decision 3-**SAT**. Input: 3-**SAT** formula φ on variables $x_1, x_2, \ldots, x_n$. Output: 1 if a satisfying assignment exists and otherwise, 0.

Describe your reduction precisely. Assert the correctness and argue that the reduction can be computed in polynomial time. No pseudocode.

Hint: Assume you have an algorithm $\mathcal{O}$ that can solve Decision 3-**SAT**. Then given an input to Search 3-**SAT**, make $O(n)$ black-box queries to $\mathcal{O}$.

Solution: The idea is that we will use $\mathcal{O}$ to *binary search* through the exponential set of possible assignments.

If we ran the oracle $\mathcal{O}$ on the original formula φ and outputted 0, then there is no satisfying assignment so we can output $\perp$. However, if $\mathcal{O}$ outputs 1, then there must exists a solution. Notice that in the solution $x_1 = 0$ or $x_1 = 1$; this is an exhaustive search.

We can determine which of these two possibilities is true in the following manner: Take the formula φ and replace any instance of x_1 with 0. If the formula simplifies (ex. $0 \vee x_j \vee \neg x_k \Leftrightarrow x_j \vee \neg x_k$ or $1 \vee x_\ell \vee x_m \Leftrightarrow 1$), make such simplifications. Call this new formula φ'. Now run $\mathcal{O}$ on φ'. If it outputs 1, then there is a satisfying assignment that has $x_1 = 0$. Otherwise,

all satisfying assignments have $x_1 = 1$. Store the assignment α_1 that produced a satisfying assignment.

Now, we notice that if we replace any instance of x_1 with α_1 and simplify for a new formula φ_1, this formula has a satisfying assignment and acts on $n-1$ literals. Then we can recursively repeat this procedure to produce an assignment $\alpha_2, \ldots, \alpha_n$.

Notice, that the overall runtime then requires $O(n)$ calls to the algorihtm $\mathcal{O}$ in addition to at most a polynomial time overhead. This completes the reduction.

5 (★★★★★) Reduction to 3-Coloring

One of the beautiful properties of NP-complete problems is that they span most branches of theoretical computer science. For example, 3-**SAT**, as we have seen, is about boolean formulas; Traveling Salesman is about paths in graphs; Integer Programming is about optimization; the list goes on. Richard Karp's 1972 paper, *Reducability among Combinatorial Problems* laid the groundwork for this field by demonstrating 21 different problems which were all polynomial time reducible to 3-**SAT**. Today, we will explore one of these, namely 3-Coloring. The problem is stated as follows:

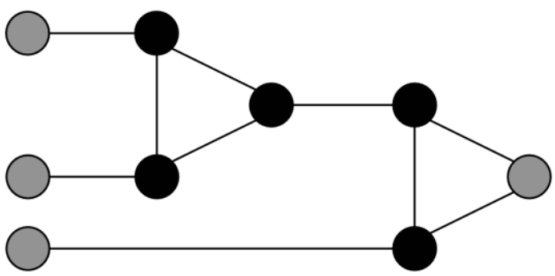
Input: A graph $G = (V, E)$. Output: 1 if there exists an assignment $\mathcal{C} : V \to \{0, 1, 2\}$ of one of three colors to each of the vertices such that no adjacent vertices are the same color if such an assignment exists. Otherwise, 0.

Recall, that vertices are adjacent if they share an edge. Provide a reduction from 3-**SAT** to 3-Coloring.

Describe your reduction precisely. Assert the correctness and argue that the reduction can be computed in polynomial time. No pseudocode.

Hint: Given a 3-**SAT** formula φ on literals $x_1, \ldots, x_n$, the goal is to produce a graph G. Your graph will consist of $\geq 2n + 3$ vertices where the $2n + 3$ vertices are: one vertex corresponding to each literal x_i and one corresponding to each literal $\neg x_i$ and three additional vertices that will form a triangle. Since, they form a triangle, they must each be colored differently if a 3-Coloring exists. Furthermore, one of the colors will represent true, i.e. 1. Another color will represent false, i.e. 0.

In addition, the following "gadget" will be useful. In the following graph,



if each of the grey nodes are colored with either 0 or 1, then it is possible to 3-Color the entire graph if and only if at least one of the grey nodes on the left has the same color as the one on the right. [In your proof, you make take this statement for granted without proving it explicitly.]

Solution: Let φ be a formula on literals $x_1, \ldots, x_n$. Produce the graph as follows. Generate 3 vertices called A, B, C and add edges between them to form a triangle. Generate $2n$ vertices suggestively called x_i and $\neg x_i$ for $i = 1, \ldots, n$. Add a triangle $A, x_i, \neg x_i$ for each i. Then for every clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ attach a version of the gadget where the three grey nodes on the left are ℓ_1, ℓ_2, ℓ_3 and the grey node on the right is B.

We notice the following properties of the graph. In any 3-Coloring of the graph, the three vertices A, B, C must be colored different. Without loss of generality, let A be colored with 2, B colored with 1, and C colored with 0. As suggested in the hint, the colors 1 and 0 correspond to true and false, respectively. Furthermore, since $A, x_i, \neg x_i$ is a triangle, then x_i and $\neg x_i$ must be colored with 0 and 1 and they must be different colors. This corresponds to an assignment of 1 and 0 to the variable x_i. Lastly, we notice that by the hint given, the gadget corresponding to clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ can only be 3-Colored if one of the nodes corresponding to ℓ_1, ℓ_2 or ℓ_3 is colored with 1 since the right node of the gadget is B which is colored with 1.

We now prove the first direction. Assume the formula has a satisfying assignment $\{x_i = \alpha_i\}$. Then, we claim that that by coloring the vertices $x_i = \alpha_i$ and $\neg x_i = \neg\alpha_i$, then we can 3-color each gadget because each clause will include at least 1 true literal and therefore at least one of the left grey nodes of the gadget must be colored with 1.

For the other direction, we can simply take the color of each vertex x_i as the assignment of x_i in the formula. Each clause is satisfied because each gadget must include at least 1 left grey node colored with 1.

It is not difficult to see that the reduction can be constructed in polynomial time.