CMSC 451: Lecture 20
NP-Completeness: 3SAT and Independent Set

Tuesday, Nov 28, 2017

Reading: DPV Sect. 8.3 and KT Sect. 8.2.

Recap: Recall the following definitions, which were given in earlier lectures.

P: is the set of decisions problems solvable in polynomial time, or equivalently, the set of languages for which membership can be determined in polynomial time.

NP: is the set of languages that can be *verified* in polynomial time, or equivalently, that can be solved in polynomial time by a "guessing computer", whose guesses are guaranteed to produce an output of "yes" if at all possible.

Polynomial reduction: $L_1 \leq_P L_2$ means that there is a polynomial time computable function f such that $x \in L_1$ if and only if $f(x) \in L_2$. A more intuitive way to think about this is that if we had a subroutine to solve L_2 in polynomial time, then we could use it to solve L_1 in polynomial time. Polynomial reductions are *transitive*, that is, $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$ implies $L_1 \leq_P L_3$.

NP-Hard: L is NP-hard if for all $L' \in$ NP, $L' \leq_P L$. By transitivity of $\leq_P$, we can say that L is NP-hard if $L' \leq_P L$ for some known NP-hard problem L'.

NP-Complete: L is NP-complete if (1) $L \in$ NP and (2) L is NP-hard.

It follows from these definitions that:

- If *any* NP-hard problems is solvable in polynomial time, then *every* NP-complete problem (in fact, every problem in NP) is also solvable in polynomial time.
- If *any* NP-complete problem cannot be solved in polynomial time, then *every* NP-complete problem (in fact, every NP-hard problem) cannot be solved in polynomial time.

Thus all NP-complete problems are equivalent to one another (in that they are either all solvable in polynomial time, or none are).

Cook's Theorem: To get the ball rolling, we need to prove that there is *at least one* NP-complete problem. Stephen Cook achieved this task. This first NP-complete problem involves boolean formulas. A boolean formula consists of variables (say x, y, and z) and the logical operations *not* (denoted $\overline{x}$), *and* (denoted $x \wedge y$), and *or* (denoted $x \vee y$).

Given a boolean formula, we say that it is *satisfiable* if there is a way to assign truth values (0 or 1) to the variables such that it evaluates to 1. (As opposed to the case where every variable assignment results in 0.) For example, consider the following formula:

$$F_1(x, y, z) \;=\; (x \wedge (y \vee \overline{z})) \wedge ((\overline{y} \wedge \overline{z}) \vee \overline{x}).$$

F_1 is satisfiable, by the assignment $x = 1$ and $y = z = 0$. On the other hand, the formula

$$F_2(x, y) \;=\; (\overline{z} \vee x) \wedge (z \vee y) \wedge (\overline{x} \wedge (\overline{y})$$

is not satisfiable since every possible assignment of 0-1 values to x, y, and z evaluates to 0.

The *boolean satisfiability problem* (SAT) is as follows: given a boolean formula F, is it possible to assign truth values (0/1, true/false) to F's variables, so that it evaluates to true?

Cook's Theorem: SAT is NP-complete.

A complete proof would take about a full lecture (not counting the week or so of background on nondeterminism and Turing machines). Here is an intuitive justification.

SAT is in NP: We nondeterministically guess truth values to the variables. (In the context of verification, the certificate consists of the assignment of values to the variables.) We then plug the values into the formula and evaluate it. Clearly, this can be done in polynomial time.

SAT is NP-Hard: To show that the 3SAT is NP-hard, Cook reasoned as follows. First, every NP-problem can be encoded as a program that runs in polynomial time on a given input, subject to a number of nondeterministic guesses. Since the program runs in polynomial time, we can express its execution on a specific input as straight-line program (that is, one containing no loops or function calls) that contains a polynomial number of lines of code in your favorite programming language. We then compile each line of code into machine code, and convert each machine code instruction into an equivalent boolean circuit. Finally, we can express each of these circuits equivalently as a boolean formula.

The nondeterministic choices can be implemented as boolean variables in this formula, whose values take on the possible values of 0 and 1. By definition of nondeterminism, the program answers "yes" if there is some choice of decisions that leads to an output of "yes". In our context, this means that there is some way of assigning 0-1 values to the variables so that our circuit produces an output of 1, that is, if the associated boolean formula is satisfied.

Therefore, if you *could* determine the satisfiability of this formula in polynomial time, you could determine whether the original nondeterministic program output "yes" in polynomial time.

Cook proved that satisfiability in NP-hard even for boolean formulas of a special form. To define this form, we start by defining a *literal* to be either a variable or its negation, that is, x or $\overline{x}$. A formula is said to be in *3-conjunctive normal form* (3-CNF) if it is the boolean-and of clauses where each clause is the boolean-or of exactly three literals. For example

$$(x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \overline{x}_3 \vee \overline{x}_4)$$

is in 3-CNF form. The *3-CNF satisfiability problem* (3SAT) is the problem of determining whether a 3-CNF[1] boolean formula is satisfiable.

NP-completeness proofs: Now that we know that 3SAT is NP-complete, we can use this fact to prove that other problems are NP-complete. We will start with the independent set problem.

[1]Is there something special about the number 3? 1SAT is trivial to solve. 2SAT is trickier, but it can be solved in polynomial time (by reduction to DFS on an appropriate directed graph). kSAT is NP-complete for any $k \geq 3$.

Independent Set (IS): Given an undirected graph $G = (V, E)$ and an integer k does G contain a subset V' of k vertices such that no two vertices in V' are adjacent to one another.

For example, the graph G shown in Fig. 1 has an independent set (shown with shaded nodes) of size 4, but there is no independent set of size 5. Therefore $(G, 4) \in \text{IS}$ but $(G, 5) \notin \text{IS}$. The independent set problem arises when there is some sort of selection problem, but there are mutual restrictions pairs that cannot both be selected. (For example, you want to invite as many of your friends to your party, but many pairs do not get along, represented by edges between them, and you do not want to invite two enemies.)

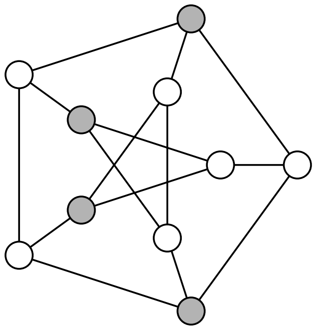

Fig. 1: A graph with an independent set of size $k = 4$.

Claim: IS is NP-complete.

Proof: As with all NP-completeness proofs, there are two parts.

IS is in NP: Recall that this means that it is possible to present a certificate that proves when a given instance has an independent set. (If the instance does not have an independent set, then we don't care what the certificate contains.) In this case, the certificate consists of the k vertices of V'. In polynomial time we can verify that, for each pair of vertices $u, v \in V'$, there is no edge between them. (In particular, if G is given as an adjacency matrix, this can be done in $O(n^2)$ time.)

IS is NP hard: It suffices to show that some known NP-complete problem (3SAT) is polynomially reducible to IS, that is, $3\text{SAT} \leq_P \text{IS}$. Let F be a boolean formula in 3-CNF form. We wish to find a polynomial time computable function f that maps F into a input for the IS problem, a graph G and integer k. (This is shown schematically in Fig. 2.) That is, $f(F) = (G, k)$, such that F is satisfiable if and only if G has an independent set of size k.

This will imply that if we could solve the independent set problem for G and k in polynomial time, then we would be able to solve 3SAT in polynomial time. The rest of this section presents this reduction in detail.

Since this is the first nontrivial reduction we will do, let's take a moment to think about the process by which we develop a reduction. An important aspect to reductions is that we *do not* know whether the formula is satisfiable, we *don't know* which variables should be true

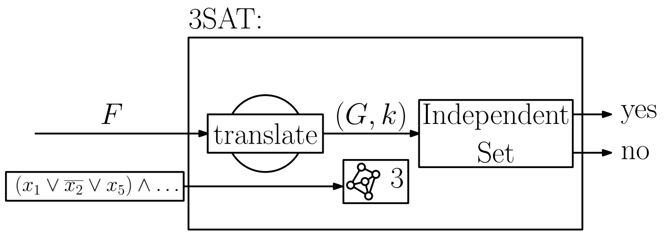

Fig. 2: Reduction of 3-SAT to IS.

or false, and we *don't have time* to determine this. (Remember: It is NP-complete!) The translation function f must operate without knowledge of the answer.

What is to be selected?

3SAT: Which variables are assigned to be true. Equivalently, which literals are assigned true.

IS: Which vertices are to be placed in V'.

Idea: Let's create a vertex in G for each literal in each clause. A natural approach would be that if a literal is true, then it will correspond to putting the vertex in the independent set. Unfortunately, this will not quite work. Instead, we observe that *at least one* vertex of each clause must be true. We will take *exactly one* such literal from each clause to put into our independent set.

Requirements:

3SAT: Each clause must contain at least one literal whose value it true.

IS: V' must contain at least k vertices.

Idea: Let's group vertices into groups of three, one group per clause. As mentioned above, exactly one vertex of each group must be in any independent set. We'll set k equal to the number of clauses to enforce this condition.

Restrictions:

3SAT: If x_i is assigned true, then $\overline{x}_i$ must be false, and vice versa.

IS: If u and v are adjacent, then both u and v cannot be in the independent set.

Conclusion: We'll put an edge between two vertices if they correspond to complimentary literals.

In summary, our strategy will be to create clusters of three vertices, one for each literal in each clause. We call these *clause clusters*(see Fig. 3). Since each clause must have at least one true literal, we will model this by forcing the IS algorithm to select one (and only one) vertex per clause cluster. Let's set k to the number of clauses. But, this does not force us to select one true literal from each clause, since we might take two from some clause cluster and zero from another. To prevent this, we will connect all the vertices within each clause cluster to each other. At most one can be taken to be in any independent set. Since we need to select k vertices, this will force us to pick exactly one from each cluster.

To enforce the restriction that only one of x and $\overline{x}$ can be set to 1, we create edges between all vertices associated with x to all vertices associated with $\overline{x}$. We call these *conflict links*.

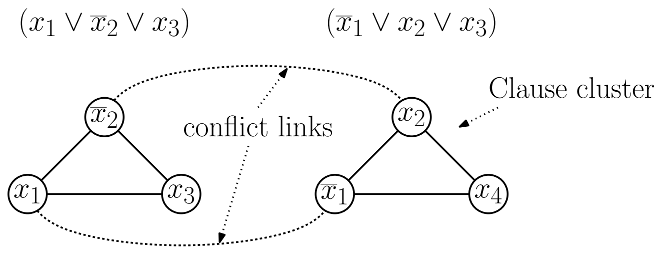

Fig. 3: Clause clusters for the clauses $(x_1 \vee \overline{x}_2 \vee x_3)$ and $(\overline{x}_1 \vee x_2 \vee x_5)$.

A formal description of the reduction is given below. The input is a boolean formula F in 3-CNF, and the output is a graph G and integer k.

3SAT to IS reduction

```
$k \leftarrow$ number of clauses in $F$
**for each** (clause $(x_1 \vee x_2 \vee x_3)$ in $F$)
    create a clause cluster consisting of three vertices labeled $x_1$, $x_2$, and $x_3$
    create edges $(x_1, x_2)$, $(x_2, x_3)$, $(x_3, x_1)$ between all pairs of vertices in the cluster
**for each** (vertex $x_i$)
    create edges between $x_i$ and all its complement vertices $\overline{x}_i$ (conflict links)
**return** $(G, k)$
```

Given any reasonable encoding of F, it is an easy programming exercise to create G in polynomial time. As an example, suppose that we are given the 3-CNF formula:

$$F = (x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee x_3).$$

The reduction produces the graph shown in Fig. 4. The clauses clusters appear in clockwise order starting from the top.

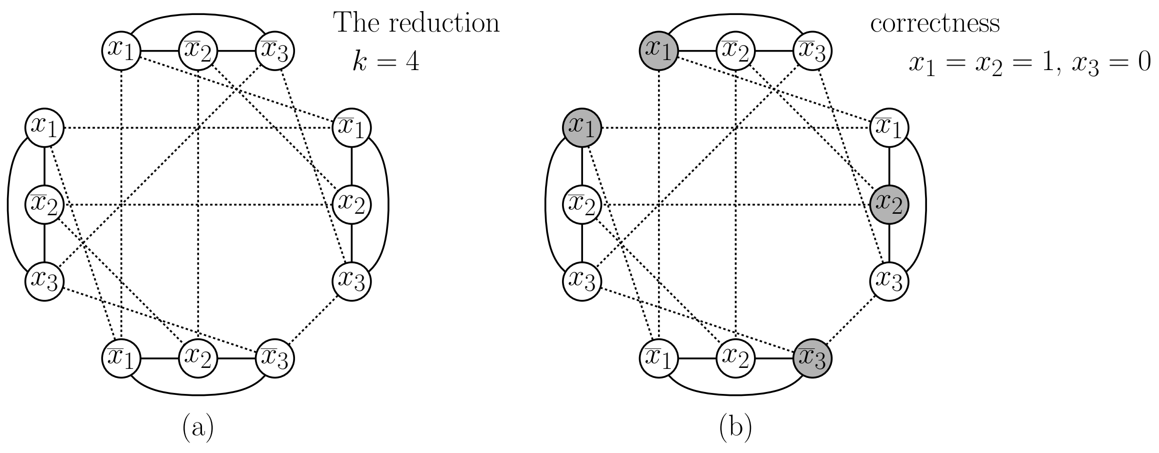

Fig. 4: 3SAT to IS reduction.

In our example, the formula is satisfied by the assignment $x_1 = 1$, $x_2 = 1$, and $x_3 = 0$. Note that the literal x_1 satisfies the first and last clauses, x_2 satisfies the second, and $\overline{x}_3$ satifies the third. Observe that by selecting the corresponding vertices from the clusters, we obtain an independent set of size $k = 4$.

Correctness: We'll show that F is satisfiable if and only if G has an independent set of size k.

($\Rightarrow$) : If F is satisfiable, then each of the k clauses of F must have at least one true literal. Select such a literal from each clause. Let V' denote the corresponding vertices from each of the clause clusters (one from each cluster). We claim that V' is an independent set of size k. Since there are k clauses, clearly $|V'| = k$. We only take one vertex from each clause cluster, and we cannot take two conflicting literals to be in V'. For each edge of G, both of its endpoints cannot be in V'. Therefore V' is an independent set of size k.

($\Leftarrow$) : Suppose that G has an independent set V' of size k. We cannot select two vertices from a clause cluster, and since there are k clusters, V' has exactly one vertex from each clause cluster. Note that if a vertex labeled x is in V' then the adjacent vertex $\overline{x}$ cannot also be in V'. Therefore, there exists an assignment in which every literal corresponding to a vertex appearing in V' is set to 1. Such an assignment satisfies one literal in each clause, and therefore the entire formula is satisfied.

Let us emphasize a few things about this reduction:

- Every NP-complete problem has three similar elements: (a) something is being selected, (b) something is forcing us to select a sufficient number of such things (requirements), and (c) something is limiting our ability to select these things (restrictions). A reduction's job is to determine how to map these similar elements to each other.
- Our reduction did not attempt to solve the 3SAT problem. (As a sign of this, observe that whatever we did for one literal, we did for all.) Remember this rule! If your reduction treats some entities different other, based on what you think the final answer may be, you are very likely making a mistake. Remember, these problems are NP-complete!