CSE 531: Computational Complexity I **Winter 2016**

Lecture 4: More NP-completeness, NP-search, coNP

January 15, 2016

Lecturer: Paul Beame *Scribe: Paul Beame*

1 More NP-completeness

On Wednesday we showed that $CIRCUIT\text{-}SAT \leq_p 3SAT \leq INDEPENDENT\text{-}SET$ were all NP-complete. We now give a sampler of NP-completeness proofs for other problems.

We first observe that NP-completeness immediately follows for $CNFSAT$ and SAT since $3SAT \leq_p CNFSAT \leq_p SAT$. Note that $3SAT \subset CNFSAT \subset SAT$. The reduction in each case is almost the identity (though of course it can't be since the sets are different). The only issue is that we have to modify satisfiable formulas that are not in 3CNF. For the $3SAT \leq_p CNFSAT$ reduction we simply map any input that is not in 3CNF to some fixed object like $x \wedge \neg x$ and so the same for the $CNFSAT \leq_p SAT$ reduction for formulas not in CNF.

Definition 1.1. *A set $U \subseteq V$ is a* clique *in undirected graph $G = (V, E)$ iff all edges on the vertices in U are present in E. A set $C \subseteq V$ is a* vertex *cover of G iff C intersects every edge of E. The complement $\overline{G}$ of a graph $G = (V, E)$ is given by $\overline{G} = (V, \overline{E})$ where $(u, v) \in \overline{E}$ for $u \neq v \in V$ iff $(u, v) \notin E$.*

Recall that $CLIQUE = \{[G, k] \mid G \text{ has a clique of size } \geq k\}$ and define $VERTEX\text{-}COVER = \{[G, k] \mid G \text{ has a vertex cover of size } \leq k\}$.

Lemma 1.2. *Let $U \subseteq V$ where $G = (V, E)$. U is a clique in $\overline{G}$ $\Leftrightarrow$ U is an independent set in G $\Leftrightarrow$ $V - U$ is a vertex cover of G.*

Proof. The first equivalence is immediate by definition. The second follows by observing that being independent means covering at most one vertex from any edge. □

Corollary 1.3. *$CLIQUE$ and $VERTEX\text{-}COVER$ are* NP*-complete.*

Proof. Clearly both are in NP where the certificate is the set given in the definition. From the lemma, the map $[G, k] \mapsto [\overline{G}, k]$ which is clearly polynomial-time computable is a reduction which shows that $INDEPENDENT\text{-}SET \leq_p CLIQUE$. Also by the lemma, the map $[G, k] \mapsto [G, n-k]$ where $n = |V|$ is a polynomial-time mapping reduction proving $INDEPENDENT\text{-}SET \leq_p VERTEX\text{-}COVER$. □

Components of an NP-completeness proof To show that B is NP-complete:

1. Show that $B \in \mathsf{NP}$:
   (a) What is the form of the guess/certificate/proof/witness?
   (b) How is the witness verified?
   (c) Why is that verification polynomial time?
2. Show that $A \leq_p B$ for some known NP-complete problem A:
   (a) Define the mapping f from inputs of the form suitable to A to those suitable for B.
   (b) Why is f polynomial-time computable?
   (c) Show that $x \in A$ implies $f(x) \in B$
      - Describe how to convert a witness w that $x \in A$ to a witness that $f(x) \in B$.
   (d) Show that $f(x) \in B$ implies $x \in A$
      - Describe how to convert a witness w' that $f(x) \in B$ to a witness that $x \in A$.

So far, after the Cook-Levin Theorem we have seen several very simple reductions, and one somewhat clever one ($3SAT \leq_p INDEPENDENT\text{-}SET$). Reductions like the latter one are often called "gadget reductions" because of the objects present in the reductions. When $3SAT$ is a starting point for a gadget reduction, one will want to have a "Boolean part", something about the problem that reflects independent 0-1 decisions for the truth assignment, as well as a "clause part" that reflects the constraints that each clause imposes. We now give a couple of examples.

Definition 1.4. *A* k-coloring *of a graph* $G = (V, E)$ *is a map* $\chi : V \to \{1, \ldots, k\}$ *such that* $(u, v) \in E$ *implies that* $\chi(u) \neq \chi(v)$.

Let $COLOR = \{[G, k] \mid G \text{ has a } k\text{-coloring}\}$ and
$kCOLOR = \{[G] \mid G \text{ has a } k\text{-coloring}\}$. Clearly both are in NP since the witness is the coloring χ and verification only requires checking each edge of G.

Theorem 1.5. $3COLOR$ *is* NP*-complete.*

Proof. Since we already know that both are in NP, it suffices to show that $3SAT \leq_p 3COLOR$. For the construction of the mapping reduction f that produces a graph G given a $3CNF$ formula ϕ, we begin with a base triangle having vertices labelled T, F, and O, respectively. For any 3-coloring we can identify each of the 3 colors with the label of the vertex of this triangle that is colored with that label.

For every variable x_i, we have a separate edge with endpoints labeled by the literals x_i and $\overline{x_i}$, respectively, both with edges forming a triangle with the O vertex of the base triangle. In this way every 3coloring assigns the color of T or F to node x_i and the complement color to $\overline{x_i}$.

It remains to add gadgets for the clauses. For each clause we have a separate subgraph on 6 vertices, 3 "inner" ones forming a triangle and 3 "outer" ones, each connected to a different inner vertex of the gadget. Each of the 3 outer vertices is connected by an edge to the T vertex in the base triangle. This means that each outer vertex can only be colored either O or F. Each outer vertex of the gadget for each clause C_j is also connected to the corresponding literal node that appears in clause C_j, one per vertex.

This completes the definition of the output of the function f. It is computable in nearly linear time in the size of ϕ.

It remains to argue correctness. First, suppose that ϕ is satisfiable and let α be a satisfying assignment for ϕ. We use this to define a 3-coloring χ by setting the color of node x_i to be the same as the truth value of x_i under α, and the color of node $\overline{x_i}$ to be its complement. Since α satisfies each clause C_j of ϕ, there is some literal of C_j that has value T. This means that we can assign the outer node of the gadget for C_j the color F. We color the other two outer nodes color O and complete the coloring on the inner triangle by assigning the color of O to the inner node connected to the outer node labelled F, and assigning the other two inner nodes the colors T and F. Clearly χ is a 3-coloring of G.

Now suppose that G has a 3-coloring χ. Then, as described above, χ yields a truth assigment α that can be read off by comparing the colors assigned to the nodes labelled x_i to the color of base node T. It remains to show that α satisfies ϕ. Since χ has to assign the color used at base node O to one of the three vertices in the inner triangle for each gadget, one of the outer vertices for each gadget must not use color O and since the outer vertices are connected to base node T, that outer vertex must have the color of base node F. The literal of the clause associated with that outer vertex must then have the same color as T since the literal nodes are all connected to base node O. Therefore the assignment satisfies each clause of ϕ. □

Definition 1.6. *A* Hamiltonian path *(or* Hamiltonian cycle*) is a simple path (respectively cycle) that visits all vertices of G. (That is, it visits each vertex of G exactly once.)*

Let $DHAMPATH = \{[G] \mid G$ is a directed graph that has a Hamiltonian path$\}$
and $DHAMCYCLE = \{[G] \mid G$ is a directed graph that has a Hamiltonian cycle$\}$.

Theorem 1.7. *$DHAMPATH$ and $DHAMCYCLE$ are* NP*-complete*

Before proving the hardness we note the subtlety of what can make a problem NP-complete. The corresponding notions of Eulerian paths and cycles, in which each edge must be visited exactly once, have easy polynomial-time algorithms since connectedness and simple degree constraints suffice to determine the outcome.

Proof. Again, membership in NP is immediate since a witnessing path or cycle is easy to check. For NP-hardness we again show reductions from $3SAT$: On input $[\phi]$, the basic graph produced consists of directed diamonds chained one above another, one for each variable x_i, with a path involving a series of nodes connected in both directions between the two middle level vertices of each diamond. In addition to the diamonds we include an extra top vertex connected to the top of the diamond for x_1 and an extra bottom vertex connected to the bottom of the diamond for x_n. The only difference between the reductions will be an optional edge from the bottom vertex to the top vertex.

The directed path/cycle must traverse the graph from top to bottom and in order to traverse both of the middle vertices can traverse in each diamond by the path joining them in either a left-to-right or right-to-left manner. Left-to-right will correspond to the assignment T for variable x_i whereas right-to-left traversal will correspond to the assignment F.

This gives the "Boolean" part of the reduction. It remains to decribe the clause gadgets. To do so we will add a single vertex for each clause of ϕ make sure that the bi-directional path in the diamond for each x_i has precisely $3k + 1$ internal vertices if there are k occurrences of either x_i or $\overline{x_i}$ in ϕ and allocate 2 consecutive vertices of the path to each occurrence with separator vertices between the pairs and at the ends.

If x_i occurs in clause C_j there is a directed edge from the left node of the pair associated with that occurrence to the node for C_j and a directed edge from C_j to the right node associated with that occurrence. If $\overline{x_i}$ occurs in clause C_j then the directions are reversed, i.e., there is a directed edge from the right node of the pair associated with that occurrence to the node for C_j and a directed edge from C_j to the left node associated with that occurrence. (Thus for each C_j, the corresponding node has in-degree and out-degree 3.)

This completes the reduction f. It is clear that the output directed graph G is of linear size in the size of the input formula ϕ and f can be computed in polynomial time. It remains to argue correctness.

If ϕ has a satisfying assignment α then we use a traversal from the top node to the bottom node that is left-to-right on diamond x_i iff x_i is assigned value T by α and right-to-left iff x_i is assigned value F, but taking a one-node diversion to cover the node for C_j on the occurrence of the first literal in C_j that is assigned true. Observe that the diversion can take place precisely because the assignment matches the polarity of the literal in the clause. Therefore the graph has a Hamiltonian path/cycle.

Now suppose that the graph has a Hamiltonian path (or cycle). If the path (or cycle) involves left-to-right or right-to-left traversals with one-node diversions to cover clause nodes, we can see that one can immediately read off an assignment and that assignment must satisfy ϕ by the direction of the diversions. However, we must rule out the possibility that when a clause node C_j is reached that the path continues along some out-edge that is not paired to that in-edge.

Suppose that it did so, let u be the other end of the in-edge and v be the neighbor of u on the diamond path containing u that has not yet been visited at the point that the edge (u, C_j) is taken.

By construction, v is either paired with u or is a separator vertex. In the either case, v can only be from the next vertex along the bi-directional path (given that it is not reached from u or from C_j) and so it must be an endpoint of the traversal. In the case of $DHAMCYCLE$, this is impossible because there is no endpoint. In the case of $DHAMPATH$, this is impossible because the bottom vertex must be the endpoint of the traversal. Either way we reach a contradiction. Hence the traversal corresponds to a satisfying truth assignment and the reduction is correct. □

Let $01PROG = \{[A, b] \mid A$ is an $m \times n$ integer matrix for some m, n and b is an integer vector such that $Ax \geq b$ has a solution $x \in \{0, 1\}^n\}$. It is easy to see $01PROG \in \mathsf{NP}$ since the computation of Ax given x and comparison to b can be checked in polynomial time since algorithms for integer arithmetic are efficient. NP-completeness follows easily using a reduction from $3SAT$ (or $CNFSAT$) that converts each clause such as $x_1 \vee \neg x_2 \vee x_3$ to an equivalent 01 integer inequality, $x_1 + (1 - x_2) + x_3 \geq 1$ (and rewriting in standard form as $x_1 - x_2 + x_3 \geq 2$).

Now consider $IP = \{[A, b] \mid A$ is an $m \times n$ integer matrix for some m, n and b is an integer vector such that $Ax \geq b$ has an integer solution $x\}$. It is clear that by adding inequalities $x_i \geq 0$ and $-x_i \geq -1$, we can show that $01PROG \leq_p IP$ and hence IP is NP-hard. It is in fact also NP-complete, though in this case it is not obvious and proof of membership in NP is the hard part of the argument.

The problem LP is the same as the IP problem except that solutions can be arbitrary real numbers. This has been known to be in P since the late 1970's due to the work of Khachian. On the other hand the problem, $QUAD\text{-}PROG$ which is the language consisting of the set of systems of quadratic equations with integer coefficients that have solutions over the real numbers is NP-hard. In this case it is not too hard to show that $01PROG \leq_p QUAD\text{-}PROG$.

The above examples give some idea of the range of NP-complete problems. Since the original papers, tens of thousands of natural problems have been shown to be NP-complete and an efficient solution to any one of them would solve all of them.

NP search The definition of NP only concerns decision problems of whether or not a witness exists. We are often also interested in the *search* problem of finding some witness if one exists. (Though this answer may not be unique we can make it into a function by fixing on, say, the lexicographically smallest witness if one exists.) The two problems are closely connected as shown by the following result.

Theorem 1.8. *If* $\mathsf{P} = \mathsf{NP}$ *then for every* $L \in \mathsf{NP}$ *and verifier* V *for* L *there is a polynomial-time (witness-finding) algorithm* W *such that*

$$x \in L \Leftrightarrow V(x, y) = 1.$$

Proof. This is a classic example of a "search-to-decision" reduction. We first show the result for SAT. If $P = NP$ then there is a polynomial-time algorithm M for SAT. The algorithm W will take formula ϕ with input variables $y_1, \ldots, y_n$ and operate as follows: First check if ϕ is satisfiable; if not, fail. Otherwise, determine a satisfying assignment α for y one bit at a time as follows: First run M on formula $\phi \wedge y_1$; if it outputs 1 then run it on $\phi \wedge y_1 \wedge y_2$, etc. If M outputs 0 on input $\phi \wedge y_1$ then run M on formula $\phi \wedge \overline{y_1} \wedge y_2$ and continue. At each stage M will be run on a formula equivalent to $\phi \wedge (y_1 = \alpha_1) \wedge (y_{i-1} = \alpha_{i-1}) \wedge y_i$. The last of these calls will determine the value of α_n.

It remains to argue this for arbitrary NP languages L and verifiers V. To do this we observe that the proof of NP-completeness of $CIRCUIT\text{-}SAT$ and its reduction to $3SAT$ (and hence SAT) are such that the witnesses for satisfiability allow one to simply read off the witness y such that $V(x, y) = 1$. $\square$

coNP Membership in NP is one-sided notion. There are witnesses for membership in L but not for non-membership. It is therefore natural to consider the set

$$\mathsf{coNP} = \{\overline{\mathsf{L}} \mid \mathsf{L} \in \mathsf{NP}\}.$$

This definition is equivalent to the following alternative definition which has a nicer dual formulation involving $\forall$ rather than $\exists$:

Definition 1.9. coNP *is the set of languages L such that there is a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time U such that*

$$x \in L \Leftrightarrow \forall y \in \{0,1\}^{p(|x|)}\ U(x, y) = 1.$$

Observe that for a language $L \in \mathsf{NP}$, with NP verifier V, the corresponding U for the $\overline{L}$ outputs $1 - V(x, y)$ on input (x, y).

Some natural languages in coNP are
$UNSAT = \{[\phi] \mid \phi$ is an unsatisfiable Boolean formula$\}$ and
$TAUT = \{[\phi] \mid \phi$ is a propositional logic tautology$\}$.
In both cases, one must check all possible assignments to determine that the the formula is in the language (though a single assignment is enough to show that it is not).

$UNSAT$ is not quite $\overline{SAT}$ since it excludes inputs that are not Boolean formulas; however it is easy to see that $\overline{SAT} \leq_p UNSAT$ since it is easy to map inputs that are not Boolean formulas to an obviously unsatisfiable formula like $x \wedge \neg x$. Moreover, $UNSAT \leq_p TAUT$ since ϕ is unsatisfiable iff $\neg\phi$ is a tautology.

Observe also that our notion of reduction is preserved under complement: $A \leq_p B \Leftrightarrow \overline{A} \leq_p \overline{B}$. Hence, by the Cook-Levin theorem, every language $A \in \mathsf{coNP}$ is polynomial-time reducible to $UNSAT$ and $TAUT$; i.e. using the obvious analogous definition to that for NP we have that $UNSAT$ and $TAUT$ are coNP-complete.

It is an open question whether or not $\mathsf{NP} = \mathsf{coNP}$. Since P is closed under complement, $\mathsf{P} = \mathsf{NP}$ implies that $\mathsf{NP} = \mathsf{coNP}$ but we could have the latter without the former. Either way, the conclusion would be somewhat surprising.

Theorem 1.10. $\mathsf{NP} = \mathsf{coNP}$ *if and only if every propostional logic tautology has a short (polynomial-size) proof that is easy to check.*

Proof. This follows immediately from the fact that $TAUT$ is a complete problem for coNP and the fact that NP is the set of languages that have polynomial-size proofs of membership that are efficiently checkable. $\square$

Clearly P is contained in both NP and coNP and both are contained in $\mathsf{EXP} \subseteq \mathsf{NEXP}$. Next time we will prove that $\mathsf{P} \neq \mathsf{EXP}$. For now we observe that the power of nondeterminism at higher complexity levels is related to that at lower levels.

Theorem 1.11. *If* $\mathsf{EXP} \neq \mathsf{NEXP}$ *then* $\mathsf{P} \neq \mathsf{NP}$.

Proof. We prove the contrapositive. Suppose that $\mathsf{P} = \mathsf{NP}$ and let $L \in \mathsf{NEXP}$. Then $L \in NTIME(2^{n^k})$ for some k. Let N be the nondeterministic TM running in time $O(2^{n^k})$ that determines membership in L. Define a new language

$$L_{pad} = \{(x, 1^{2^{|x|^k}}) \mid x \in L\}.$$

Consider the following nondeterministic TM N' for L_{pad}: On input $z = (x, 1^{2^{|x|^k}})$ remove the string $1^{2^{|x|^k}}$ and run M on input x and use its answer. Since the running time of N on input x is $O(2^{|x|^k})$ the total running time is only linear in the input size $|z|$ for L_{pad} and hence N' runs in polynomial time. This proves that $L_{pad} \in \mathsf{NP}$. Now since $\mathsf{P} = \mathsf{NP}$ by assumption, there must be some polynomial time (deterministic) TM M' deciding L_{pad}.

Now define a TM M for L as follows: On input x, produce the string $z = (x, 1^{2^{|x|^k}})$ and run M' on input z. The running time of M' is polynomial in its inputs size and hence TM M runs in time $2^{O(n^k)}$ which is $O(2^{n^{k+1}})$. Therefore $L \in \mathsf{EXP}$. We have therefore shown that our assumption implies that $\mathsf{NEXP} = \mathsf{EXP}$ which is what we needed to prove. $\square$