INFR11102: Computational Complexity 15/10/2019

Lecture 9: More on NP-completeness; Ladner's theorem

Lecturer: Heng Guo

1 More on NP-completeness

We will show that from 3-Sat, it is relatively easy to derive NP-hardness for other combinatorial problems.

For an optimization problem, we often consider their "threshold" version as the decision problem. Recall that for a graph $G = (V, E)$, a *vertex cover* $C \subseteq V$ is a subset of vertices so that every edge is adjacent to at least one vertex in C.

Name: VC

Input: A graph G and a number k.

Output: Does G contain a vertex cover of size at most k?

Theorem 1. VC *is* NP*-hard.*

Once again, we will build a reduction, this time from 3-Sat to VC. The key to this reduction, is to transform the "local" constraint in 3-Sat to the local constraint in VC.

Proof. Given a 3-CNF formula φ with n variables and m clauses, we construct a graph $G_\varphi = (V, E)$ as follows. We introduce two vertices v_{x_i} and $v_{\overline{x_i}}$ for each variable x_i, and connect the two. To simplify the notation, we will just call the two vertices x_i and $\overline{x_i}$. For each clause $c_j = \ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3}$, we introduce a vertex for each of its literals, and connect all of them pairwise. (Rewrite $\ell_1 = \ell_1 \vee \ell_1 \vee \ell_1$ and $\ell_1 \vee \ell_2 = \ell_1 \vee \ell_1 \vee \ell_2$ so that every clause has three literals.) In addition, connect all literals to the corresponding variable vertex. For example, if $\ell_{4,1}$ is $\overline{x_5}$, then $(\ell_{4,1}, \overline{x_5}) \in E$. So $|V| = 2n + 3m$ and $|E| = n + 6m$.

Now, we should step back and think about the construction. To get a small vertex cover, we want to occupy as few vertices as possible. We have to cover at least one of $(x_i, \overline{x_i})$. We also have to cover at least two of $\ell_{j,1}$, $\ell_{j,2}$, and $\ell_{j,3}$ for a clause c_j. Hence, the vertex cover has size at least $n + 2m$.

If φ has a satisfying assignment σ, then we interpret σ as a vertex cover as follows. We cover x_i or $\overline{x_i}$ according to $\sigma(x_i)$'s value. We also cover false literals in a clause. Since σ is a satisfying assignment, there are at most two literals covered this way. If a clause triangle still has one uncovered edge, then we cover one of its endpoints. Repeat this until no triangle contains uncovered edges. There is one vertex per each triangle that is uncovered, and it corresponds to a true literal. So the size of the cover is $n + 2m$.

We have covered all variable pairs and all clause triangles this way. The only edges in question are those connecting the uncovered vertex in each triangle, and the corresponding variable vertex. We know that the uncovered literal must be true, and hence the corresponding variable vertex is covered. So all edges are fine and we have a vertex cover of size $n + 2m$.

On the other hand, suppose we do have a vertex cover of size $n+2m$. As we have argued earlier, it must contain exactly one for each pair of variable vertices, and two of every clause triangle. We interpret the covered variable vertex as true. If a clause is not satisfied, then all literals must be false. Hence, the vertex uncovered in the triangle is connected to a uncovered variable vertex, contradicting to the assumption of a vertex cover.

In summary, our reduction takes φ, and outputs $(G_\varphi, n + 2m)$, which clearly can be computed in polynomial time. The clause φ is satisfiable if and only if G_φ has a vertex cover of size at most $n + 2m$. □

From VC, we can further show other problems to be NP-hard. For a graph $G = (V, E)$, an *independent set* $I \subseteq V$ is a subset of vertices so that no two vertices in I are adjacent.

Name: IndSet

Input: A graph G and a number k.

Output: Does G contain an independent set of size at least k?

Theorem 2. IndSet *is* NP*-hard.*

Proof. We show a reduction VC $\leq_p$ IndSet. The graph $G = (V, E)$ is kept as the same, but for the input k, we replace it with $n - k$. It is easy to see that, for a vertex cover C of size t, its complement $\overline{C} := V \setminus C$ is an independent set of size $n - k$. Hence the reduction is valid. □

For a graph $G = (V, E)$, a *clique* $C \subseteq V$ is a subset of vertices so that any two vertices in C are adjacent.

Name: Clique

Input: A graph G and a number k.

Output: Does G contain a clique of size at least k?

Theorem 3. Clique *is* NP*-hard.*

Proof. We reduce IndSet $\leq_p$ Clique. Given $G = (V, E)$, we construct $G' = (V, \overline{E})$, where $\overline{E}$ is the complement of E. Namely, $(i, j) \in \overline{E}$ if and only if $(i, j) \notin E$. It is easy to verify that an independent size in G is a clique in G'. Hence the reduction is valid. □

1.1 Hamiltonian cycle

"Non-local" problems can also be NP-hard. A *Hamiltonian cycle* is a cycle where all vertices is visited exactly once.

Name: HC

Input: A graph G.

Output: Does G contain a Hamiltonian cycle?

In other words, HC is asking whether the longest cycle in G has length n. This problem is "non-local" in the sense that the constraint involves all vertices (a.k.a. global).

Theorem 4. HC *is* `NP`*-hard.*

Proof. We will reduce from VC.

Given $G = (V, E)$, an instance of VC, we construct a graph G'. For each $e \in E$, we replace it by a "gadget" shown in Figure 1. The gadget in Figure 1 contains three possible

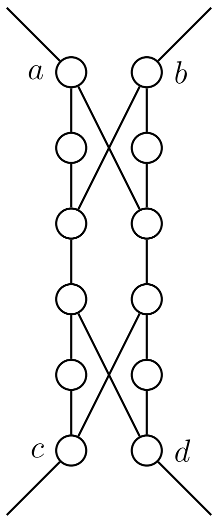

Figure 1: A gadget from VC to HC

traversals. Either in order a-b-d-c or in order b-a-c-d, or in two completely separate thread from a to c and b to d. For an edge $(u, v) \in E$, after the replacement, we use the a-c thread to represent the vertex u and the b-d thread to represent v. After replacing all edges, we connect all threads corresponding to the same vertex in some arbitrary order.

Finally, for the input k, we introduce k new vertices $x_1, \cdots, x_k$, and connect them to the start and finish of every vertex thread.

Now we claim that if there is a vertex cover $C = \{v_1, \cdots, v_k\}$ of size k in G, there must be a Hamiltonian cycle in G'. The point is that for an edge (u, v), if $u \in C$ but $v \notin C$, then we go through a-b-d-c thread so that a and c are the in and out, and similarly for the case where $v \in C$ but $u \notin C$. If both u and v are in C, then we use the two separate traversals. In summary, $u \in C$ if and only if its corresponding two vertices are the in/out in the gadget in Figure 1. The Hamiltonian cycle starts at x_1, goes through all threads associated with v_1 according to the rule above, and arrives in x_2. Then we go through all threads associated with v_2 to x_3, so on and so forth, eventually returning to x_1.

On the other hand, if there is a Hamiltonian cycle in G', it has to include all vertices in $\{x_1, \cdots, x_k\}$. However, by construction, for each visit of x_i, we can only traverse one thread that corresponds to some vertex. Hence, we will traverse exactly k threads in the Hamiltonian cycle, and we claim that these vertices are a vertex cover in G. This is because, if an edge, say (u, v) is not covered, then the Hamiltonian cycle in G' must have missed some vertices, which is impossible. □

2 Ladner's theorem (non-examinable)

In view of NP-complete problems, it is natural to wonder whether we have a dichotomy between P and NP-complete, namely whether NP $\setminus$ P consists of only NP-complete problems, if P $\neq$ NP. The answer, given by Ladner [Lad75], shows that problems of intermediate complexity exist.

Theorem 5 (Ladner)**.** *If* P $\neq$ NP*, then there exists* $L \in$ NP $\setminus$ P *and* L *is not* NP*-complete.*

In fact, Ladner showed a stronger result, that there is an infinite hierarchy between P and NP-complete, assuming P $\neq$ NP. The construction of L, as we will see, is rather artificial. There are a few natural candidate intermediate problems, such as FACTORING and GRAPHISOMORPHISM, which we will see later.

The basic idea is to take a CNF formula φ, encoded as a binary string, and pad it artificially so that the input length is $f(n)$ instead of the original n. The longer we pad, the easier the problem becomes. If $f(n)$ is exponentially large, then of course the new problem can be solved in time polynomial in $f(n)$. So to get intermediate complexity, we would want $f(n)$ to be larger than polynomial, but smaller than exponential. The difficulty is to formalize this intuition, and it will be a diagonalisation argument.

To formalize this intuition, define

$$L := \{\varphi 0 1^{f(n)-|n|-1} \mid \varphi \in \textsc{Sat} \text{ and } |\varphi| = n\},$$

where $f(n)$ will be defined next.

Suppose we enumerate all TMs determining all languages in P, $M_1, M_2, \cdots$. In particular, we can assume that M_i runs in time n^i. Define $f(n) = n^{g(n)}$ where $g(n)$ is defined recursively as follows:

- $g(1) = 1$;
- Suppose $g(n-1) = i$. For $g(n)$, we enumerate all input x of size at most $\log n$. Let $g(n) = i$ if M_i agrees with L for all such x, and $g(n) = i + 1$ otherwise.

Notice that since we only check logarithmic size x, $g(n)$ and $f(n)$ can be computed in polynomial time in n. (Since $g(n)$ is recursively defined, one actually needs to solve a recurrence to verify this claim.) It implies that $L \in$ NP.

We claim that $L \notin$ P. If so, $L = L(M_i)$ for some i and thus $g(n) \leq i$. However, $g(n)$ is non-decreasing, so $g(n) = c$ and $f(n) = n^c$ for some constant $c \leq i$ and all large enough n.

Thus $\textsc{Sat}$ can be determined in time $(f(n))^i = n^{ci}$ which is still a polynomial. It contradicts to the assumption that $\mathtt{P} \neq \mathtt{NP}$.

As a consequence, $g(n)$ must be unbounded.

The more complicated part is to verify that L is not $\mathtt{NP}$-complete. Suppose otherwise it is, and therefore there is a reduction, say R, from $\textsc{Sat}$ to L. Namely, $\varphi \in \textsc{Sat} \Leftrightarrow R(\varphi) \in L$.

Suppose $|R(\varphi)| \leq n^j$ for some constant j. We then give a polynomial-time to solve $\textsc{Sat}$, again contradicting to our assumption. Since $g(n)$ is unbounded, there exists n_0 such that for all $n \geq n_0$, $g(n) > j$. We can solve all formulas up to size n_0 by brute force.

For a formula φ of size $n \geq n_0$, we run the reduction $R(\varphi)$, which yields a string $\psi 0 1^{f(m)-m-1}$ where $m = |\psi|$ such that $\varphi \in \textsc{Sat}$ if and only if $\psi \in \textsc{Sat}$. (If $R(\varphi)$ does not have this form then we immediately know that $\varphi \notin \textsc{Sat}$.) In particular, this implies that $|R(\varphi)| = f(m) \leq n^j$. If $m \geq n \geq n_0$, then $f(m) = m^{g(m)} > m^j \geq n^j$, a contradiction. Hence $m < n$, then we run our algorithm recursively on ψ, which has a smaller size. This yields a polynomial time algorithm, contradicting to the assumption that $\mathtt{P} \neq \mathtt{NP}$.

The "padding" proof above is an unpublished result by Russell Impagliazzo. An alternative presentation can be found in [AB09, Theorem 3.3]. Another (original) proof via "punching holes" can be found in [Pap94, Theorem 14.1]. Both proofs (including the one we talked about here) can also be found in [DF03, Appendix A].

Remark (Bibliographic). Relevant chapters are [AB09, Chapter 3.3, 2.5] and [Pap94, Chapter 10, 14.1].

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[DF03] Rodney G. Downey and Lance Fortnow. Uniformly hard languages. *Theor. Comput. Sci.*, 298(2):303–315, 2003.

[Lad75] Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.