Lecture 8: NP Completeness

1 Complexity Theory

We have seen that problems have varying hardness, but some problems have a "subproblem" that is easily checkable. For example:

- $SAT(\varphi)$: Is a given formula φ satisfiable?
- 3-colorability: Is a given graph 3-colorable?

A common characteristic shared by these problems is that it is "hard" to find a solution for them, but we can easily check whether a given solution is correct. In the case of $SAT(\varphi)$, if we are given a particular truth assignment τ, we have already developed a polynomial time algorithm $eval(\varphi, \tau)$ that deterministically decides if $\tau \models \varphi$. Thus, given a (possible) solution τ we can easily verify that φ is satisfiable (finding the appropriate τ is a different problem). Similarly for the 3-colorability problem, if we are given a particular coloring of the nodes, we can easily check every node to see whether its neighbors are all of a different color; if they are, the coloring we were given "testifies" that the graph has the desired property that makes it belong to the class of graphs with no monochromatic edges. In the literature, the example that helps determine if an instance of the problem has a solution is called a *witness* since it *witnesses* that the instance has the desired property. In the case of $SAT(\varphi)$, the witness is the truth assignment τ.

The above problems also have a common formulation: they both aim at answering a question of the form: *Given an instance of an object, does it satisfy certain property?*. There are other types of problems that cannot be stated in this way, for example:

- *Chess*: In a chess game, does white always win?

For this kind of problem, we cannot get a solution easily (in fact, it is very difficult to even state the problem formally, which would require a discussion on game theory and assumptions on the opponent's strategy). Next we formally define a *decision problem*.

Definition 1 (Decision Problem). *Given a finite alphabet* Σ*, a* Decision Problem *is some language* $L \subseteq \Sigma^*$*. It is* nontrivial *if* $L \neq \emptyset$ *and* $L \neq \Sigma^*$*.*

Thus we know *SAT* and 3-colorability are decision problems, while the *Chess* problem is not. We can think of the first two problems as decision problems where x is a string encoding an instance of the problem, and L is the language of all instance encodings that have some property P.

As part of the exhaustive algorithms that solve the above problems, we have a *subproblem*: given a candidate witness y, does y make x have property P? This subproblem can be formalized as a decision problem by itself as follows: "Does $(x, y) \in L_P$, where L_P is the language of all strings (x, y) where y makes x have property P ?" For our previous examples, this subproblem is decidable in polynomial time. Often, the subproblems need not be formalized, i.e. it will suffice to say "... and checking if a witness y makes x have property P is easy..."

To make matters more clear, going back to our 2 examples we have:

- L_{SAT} is the language of all satisfiable formulas $x \in \Sigma^*$, where Σ is the set of all valid formula symbols (propositions, connectives and parentheses). A formula $\varphi \in L_{SAT}$ iff φ has property *P: φ is satisfiable.* One way to see if $\varphi \in L_{SAT}$ is to use some truth assignment τ that witnesses that φ has property P, which is decidable in polynomial time by $eval(\varphi, \tau)$.

- L_{3C} is the language of all 3-colorable graphs x (here assume that x is some valid encoding of a graph using some alphabet Σ that makes x unambiguous). Thus, a particular graph $G \in L_{3C}$ iff G has property *P: there is a 3-coloring for G that makes all of its edges non-monochromatic.* Again, one way to check if $G \in L_{3C}$ is to provide some 3-coloring of the vertices of G. It is decidable in polynomial time if the witness is a valid 3-coloring of G.

1.1 The class NP

In the previous examples we recognized that we have at our disposal a polynomial time algorithm to solve the subproblem, i.e. to check whether a particular witness y makes x have property P. The question that arises immediately is: *how do we come up with a candidate solution?*. The search space for these candidate solutions is exponential in the size of the problem for the above examples, so it becomes increasingly time consuming to try them all.

Note, however, that if we could "magically guess" a candidate solution, we could check it efficiently. These problems are said to belong to the **NP** class of problems, which stands for **Nondeterministic Polynomial time**. "Magically guessing" here means *nondeterministically selecting* a candidate; and once we have one, it can be checked in polynomial time. Alternatively, we can say that a problem in **NP** can be solved in polynomial time by a *nondeterministic algorithm* that first *guesses* a witness and then checks it. Now we are ready to introduce a formal definition for the **NP** class of problems.

Definition 2 (NP). *A decision problem $L \in \Sigma^*$ is in NP if there exist:*

- *a polynomial P, and*
- *a polynomial time algorithm $A(x, y)$,*

such that $x \in L$ iff $\exists\ y \in \Sigma^{\leq P(|x|)}$ where $A(x, y) = 1$.

Where $\Sigma^{\leq k}$ is the set of all strings of length less than or equal to k. In other words, a decision problem is in **NP** if when we are given a "short" witness y, a polynomial time algorithm $A(x, y)$ exists to check if the witness makes x have the desired property. Formally, $A(x, y)$ is a membership indicator of (x, y) to the subproblem language, so we have $A(x, y) = 1$ iff y is a witness for x.

Example 1 ($SAT(\varphi)$)**.** *For the case of $SAT(\varphi)$, we saw that $L = L_{SAT}$, x is φ, the witness y is τ, and the algorithm $A(x, y)$ is $eval(\varphi, \tau)$ which takes polynomial time.*

The restriction that $y \in \Sigma^{\leq P(|x|)}$ ensures that the witness is "short enough" so that the checking algorithm $A(x, y)$ is guaranteed to always take polynomial time. This is important because there may be problems where the witness cannot be guaranteed to have a length bounded by $P(|x|)$, and thus it could not be shown that $A(x, y)$ takes polynomial time.

In the following examples the particular L, A, x and y will not be explicitly pointed out to encourage the reader to formalize them.

1.2 Some Examples

We present some well known problems with a short analysis:

Example 2 (Diophantine equation)**.** *Does $P(x_1, x_2) = 0$ have integer solutions?*

The solution is undecidable because we cannot bound the size of x_1 and x_2. Hilbert's 10th problem asked if a technique for solving a general Diophantine existed. The impossibility of obtaining a general solution was proved by Matyasevich in the early 1970's. However, verifying the correctness of a particular solution (x_1', x_2') takes only polynomial time. This problem is not in NP, because short witnesses may not exist.

Example 3 (Compositness Problem)**.** *Given a number $n \in N$, is n composite?*

Given two factors n_1 and n_2, it is easy to verify if $n = n_1 n_2$. However, the time taken to find suitable n_1 and n_2 is exponential on the length of the binary representation of n. This problem is in NP. However, in this particular case we can be a little more precise. In 2002 it was shown that this problem is also in the class PTIME, which is a subclass of NP.

Example 4 (Eulerian Cycle on graph $G(V, E)$)**.** *Is there a bijective mapping $P : E \to \{1, \ldots, |E|\}$ such that if $P(e_2) = (P(e_1) + 1) \pmod{|E|}$ then there exist vertices $u, v, w \in V$ such that $e_1 = (u, v)$ and $e_2 = (v, w)$?*

An Eulerian Cycle on a graph G is a cycle that visits each edge exactly once. An Eulerian Cycle exists iff the degree of each node in the graph is even. Thus, this problem is not only in NP, it is in fact in PTIME.

Example 5 (Hamiltonian Cycle on graph $G(V, E)$)**.** *Is there a bijective mapping* $P : V \to \{1, \ldots, |v|\}$ *such that if* $P(v) = (P(u) + 1) \bmod |V|$*, then* $(u, v) \in E$*?*

A Hamiltonian cycle on a graph G is a cycle that visits each node exactly once and terminates at the initial node. This problem is in NP.

Example 6 (k-Coloring Problem)**.** *Consider a graph representation of a map.* $G = (V, E)$*, whose nodes are the partitions in the map and neighboring partitions (nodes) are connected by edges. A* k-coloring *is a mapping* $C : V \mapsto \{1 \ldots k\}$*, obeying the constraint* $C(u) = C(v)$ *only if* $(u, v) \notin E$*. We say* G *is* k*-colorable if it has a* k*-coloring.* Graph k-colorability *is the problem if checking whether a given graph is* k*-colorable.*

Graph k-colorability is also in NP. The witness is the k-coloring C.

Example 7 (Perfect Matching on graph $G(V, E)$)**.** *Is there a subset* E' *of* E *such that every node is touched by one edge of* E'*?*

This problem is in NP. It is in fact also in PTIME.

Note that some problems that look very similar to each other may belong to different complexity classes, e.g. one may be in P and other in NP. A classical example is 2-coloring and 3-coloring of a graph. Finding if a graph can be colored by 2 colors is easy but doing the same with 3 colors is hard.

Note that an upper and lower bound for NP problems can easily be provided:

- NP $\subseteq$ EXPTIME since the problem can be solved by trying all possibilities.
- PTIME $\subseteq$ NP since an NP problem has at least the complexity of $A(x, y)$ which is in PTIME.

The NP class lies somewhere between polynomial and exponential time.

2 Reducibility and NP-completeness

We turn our attention to a very important tool that will allow us to relate problems to other "similar" problems.

Definition 3 (Polynomial Reduction)**.** *Let* $L_1, L_2 \subseteq \Sigma^*$*. We say that* L_1 *is* polynomially reducible *to* L_2*, denoted* $L_1 \leq_p L_2$ *if there exists a PTIME-computable function* $f : \Sigma^* \to \Sigma^*$*, such that* $x \in L_1$ *iff* $f(x) \in L_2$*.*

Example 8. *Let* L_+ *be a language such that* $(a, b, c) \in L_+$ *iff* $a + b = c$*. Similarly, let* L_- *be the language such that* $(a, b, c) \in L_-$ *only iff* $a - b = c$*. To reduce* L_+ *to* L_- *we have* $f(a, b, c) = (b, c, a)$*, where* f *is the function that reduces an instance of* L_+ *to* L_-*.*

Notice that $\leq_p$ is reflexive and transitive, but not symmetric. It defines a pre-order on the set of all problems. Also, $\leq_p$ is an invariant of PTIME, that is, if $L_2 \in$ PTIME and $L_1 \leq_p L_2$, then $L_1 \in$ PTIME. This is because, given an polynomial algorithm A for L_2 and a polynomial function f that reduces L_1 to L_2, we can obtain a polynomial algorithm A' for L_1 as follows: on input x, A' first calls f to compute $f(x)$ and then calls A with input $f(x)$. Then A' accepts input x iff A accepts $f(x)$ iff $f(x) \in L_2$ iff $x \in L_1$.

When we say that $L_1 \leq_p L_2$, it intuitively means that L_2 is at least as hard as L_1, or more clearly, that L_1 is no harder than L_2. Hardness here means that if we solve the decision problem for L_2, we can use the function f to solve the decision problem for L_1.

Definition 4 (NP-hard)**.** *A decision problem $L \subseteq \Sigma^*$ is* NP-hard *if for every $L' \in NP$ we have $L' \leq_p L$.*

Note that this definition does not restrict L to be in NP, that is, L could be some language *harder* than NP.

Definition 5 (NP-Complete)**.** *A decision problem $L \subseteq \Sigma^*$ is* NP-complete *if $L \in NP$ and L is NP-hard.* NPC *is the class of all NP-complete problems.*

Now, since NP-complete problems lie at a relatively low level in the full complexity hierarchy, it is easy to imagine that NP-hard problems exist. However, it is not at all obvious that any natural NP-complete problems should exist, that is, the class NPC is nonempty. That this is the case is one of the central results of computer science:

Theorem 1 (Cook-Levin)**.** *SAT is NP-complete.*

If it is the case that $L_1 \leq_p L_2$ and $L_2 \leq_p L_1$, we say that $L_1 \equiv_p L_2$. The relation $\equiv_p$ defines a set of equivalence classes within NP, ordered by $\leq_p$. In particular, all NP-complete problems are polynomially equivalent. So they fall in the same equivalence class under $\equiv_p$.

Intuitively, we say that NPC problems are the hardest problems in NP. Similarly, we can also say that PTIME contains the easiest problems in NP because of polynomial reducibility. But what is in between? Moreover we might ask whether P = NP or P $\neq$ NP. We don't know! It is obvious that $P \subseteq NP$, but it is still unknown whether $P \subsetneq NP$. There are a lot of problems in NP for which we cannot find polynomial algorithms to solve so far. The concept of NP-completeness is important because it is at the crux of the P $\stackrel{?}{=}$ NP question:

Theorem 2. *If $L \in NPC$, and $L \in PTIME$, then $P = NP$.*

This theorem states that if we can find a polynomial solution for *any* NPC problem, then all NPC problems become PTIME problems since they are all reducible to each other.

Another useful result is:

Theorem 3 (Ladner)**.** *If $P \neq NP$, then $(NP - P) - NPC \neq \emptyset$*

This is known as Ladner's Theorem. It states that there are problems in NP that are neither NP-complete nor in P, where $(NP - P)$ is the set of problems that are *strictly* in NP (but not in P), provided that $P \neq NP$, which is still unknown.

3 Co-NP and CoNP-Completeness

Recall that a formula φ is satisfiable iff $(\neg\varphi)$ is not valid, and that φ is valid iff $(\neg\varphi)$ is not satisfiable. Thus the two decision problems SAT and VALID are related. It is reasonable to ask whether $SAT \equiv_p VALID$?

Suppose that this is indeed the case. Then there is some function $f : Expr \rightarrow Expr$ such that φ is SAT iff $f(\varphi)$ is VALID. But we have a problem because while we have a witness for SAT (namely, a truth assignment), it is not immediately clear what we can use as a witness for validity. (In fact, it can be shown that a short witness for validity doesn't exist). However, notice that if we wish to prove that validity doesn't hold, then a short witness exists–namely, a truth assignment which makes the formula as a whole return 0.

It turns out that the class of problems that have short counterexamples is interesting in itself. The next definition formalizes this concept.

Definition 6 (Co-NP)**.** *A problem (language) $L \subseteq \Sigma^*$ is in Co-NP is there is a polynomial P and a PTIME algorithm A, such that for all $x \in \Sigma^*$, we have $x \notin L$ iff there is some $y \in \Sigma^{\leq P(|x|)}$ such that $A(x, y) = 0$.*

Observe that in NP, when given an answer we can 'easily' verify that it is indeed a solution, while for Co-NP, when given something that is *not* an answer, we can 'easily' verify that it is *not* a solution.

Lemma 1. *A problem $L \in Co\text{-}NP$ iff $L = \Sigma^* - L'$ for some $L' \in NP$*

Proof. You will prove this result in assignment 3. □

Like the NP-complete class, **NPC**, we can define the Co-NP-complete class, **Co-NPC**, which represents the hardest problems in Co-NP as follows:

Definition 7 (Co-NPC)**.** *A problem $L \in Co\text{-}NPC$ if*

1. *$L \in Co\text{-}NP$*
2. *$L' \leq_p L$ for all $L' \in Co\text{-}NP$*

Lemma 2. *A problem $L \in Co\text{-}NPC$ iff $L = \Sigma^* - L'$ for some $L' \in NPC$.*

Proof. Again, you will prove this result in assignment 3. □

How are SAT and VALID related? Recall that SAT has two requirements, namely that we have a well-formed formula and that it is satisfiable, or (WFF $\wedge$ SAT). Taking the negation we get ($\neg$wff $\vee$ $\neg$sat). Thus, we cannot exactly say that SAT is the complement of VALID because VALID contains well-formed formulae and not just expressions. Reformulated, $\Sigma^* - SAT \neq VALID$ because while SAT is all satisfiable formulae, $\Sigma^* - SAT$ includes nonsensical expressions as well. However, we can say that $SAT \in NPC$ and $VALID \in Co\text{-}NPC$.

More generally, how are NP and Co-NP related? We know that both contain as a subset the complexity class PTIME. Is $P = NP \bigcap Co\text{-}NP$? Is it possible that NP = Co-NP?

We know that all problems in P are trivially in NP (discussed earlier). P is *also* contained in Co-NP: suppose some problem $L \in P$. If we run the algorithm to solve the problem, then invert the output, we end up with a solver for $\Sigma^* - L$, which is also in P.

Lemma 3. $P \subseteq NP \bigcap Co\text{-}NP$

Do there exist problems in both NP and Co-NP that are *not* in P? In this regard consider the problem PRIMES, which is the problem of testing primality (is a given natural number prime?). It is trivial to show that PRIMES is in $Co\text{-}NP$, since any non-trivial factorization of a number gives an easily checked short counterexample for its primality. Showing that PRIMES is in NP is attributed to Pratt, 1976. Thus, PRIMES is in $NP \bigcap Co\text{-}NP$. However, it was not known until 2002 whether PRIMES is in P. In 2002, Agarwal et al showed that PRIMES is in P. Whether $P = NP \bigcap Co\text{-}NP$ remains an open problem.

Recall that P is *closed* under the *complement* operation. So, if $P = NP$, then also $P = Co\text{-}NP$. Open problem: $NP \overset{?}{=} Co\text{-}NP$

Lemma 4. *If $L \in NPC$ and $L \in Co\text{-}NP$, then $NP = Co\text{-}NP$*

Theorem 4 (Ladner). *The full glory of Ladner's theorem can be expressed as following:*

- *If $P \neq NP \Rightarrow (NP - NPC) - P \neq \emptyset$.*
- *If $P = NP \Rightarrow NP = Co\text{-}NP$.*
- *If $NP \neq Co\text{-}NP \Rightarrow P \neq NP$.*

4 Reducing 3-COLOR to SAT

The first NP-complete problem was SAT, and the proof that it is NP-complete is due to Cook and Levin. Cook and Levin proved that all NP problems are polynomially reducible to SAT. Here we show one particular example of reducing a NP problem to SAT. We will focus on 3-COLOR, and show that 3-Color $\leq_p$ SAT. This means that in theory, if we have a SAT solver, we can do 3-Coloring. But how practical is this? *Prima facie* it might seem that using a SAT solver would be inferior to a system designed especially for 3-Coloring. However, because a *lot* of work has been done on SAT-solvers, it turns out that using SAT as a generic problem solving environment is not a bad idea.

4.1 3-COLOR

Informally, the 3-COLOR problem asks if a graph can be colored in such a way that two nodes connected by an edge have different colors. Formally, given a graph $G = (V, E)$, does there exist a function $f : V \to \{1, 2, 3\}$, where 1, 2 and 3 represent "colors", such that $f(u) \neq f(v)$ if $(u, v) \in E$? If such a function exists, the graph G is said to be 3-*colorable*.

3-COLOR $\leq_p$ SAT

We first build a formula φ_G that represents the coloring constraints on the graph G. For each vertex $v \in V$, we define three propositions $p_{v,i}$, where $i \in \{1, 2, 3\}$. $p_{v,i}$ is 1 if vertex v's color is i and 0 otherwise. The condition that each vertex is colored with exactly one color $i \in \{1, 2, 3\}$ can be expressed as

$$\varphi_1 = \bigwedge_{v \in V} (p_{v,1} \vee p_{v,2} \vee p_{v,3}) \wedge \neg(p_{v,1} \wedge p_{v,2}) \wedge \neg(p_{v,2} \wedge p_{v,3}) \wedge \neg(p_{v,3} \wedge p_{v,1})$$

And the requirement that two adjacent vertices have different colors is expressed as

$$\varphi_2 = \bigwedge_{(u,v) \in E} \neg(p_{u,1} \wedge p_{v,1}) \wedge \neg(p_{u,2} \wedge p_{v,2}) \wedge \neg(p_{u,3} \wedge p_{v,3})$$

Let $\varphi_G = (\varphi_1 \wedge \varphi_2)$. Given G, φ_G can be constructed in time and space linear in the size of G. So the function that maps G to φ_G is polynomially computable. To complete the reduction we need to show that G is 3-colorable iff φ_G is satisfiable.

- *If G is 3-colorable then φ_G is satisfiable*:

  Let $c : V \to \{1, 2, 3\}$ be a 3-coloring of G. Define the truth assignment

  $$\tau_c(p_{v,i}) = \begin{cases} 1 & \text{if } c(v) = i, \\ 0 & \text{otherwise.} \end{cases}$$

  From the construction of φ_G, it is easy to see that $\tau_c \models G$.

- *If φ_G is satisfiable then G is 3-colorable*:

  Let τ be a truth assignment that satisfies φ_G. Then $\tau \models \varphi_1$ and $\tau \models \varphi_2$. Define $c_\tau : V \to \{1, 2, 3\}$ as follows:

  $$c_\tau(v) = i \text{ for } \tau(p_{v,i}) = 1$$

  Because φ_1 is satisfied by τ, every vertex $v \in V$ is assigned exactly one color by c_τ. Therefore, c_τ is a coloring of G. Because φ_2 is also satisfied by τ, vertices v and u cannot have the same color if (v, u) is an edge of G. Therefore, c_τ is a 3-coloring of G.