CS4820
Prelim 2 Review: NP-Completeness
Dongqing Wang

NP-Completeness is one of the major topics in CS4820. To review this topic, we will first look at the important concepts and definitions in this class of problems, then a summary on general paths to follow when approaching to prove a problem is NP-Complete, and finally talk about the characteristics of each NP-Complete problem that we introduced in class and hopefully develop a mind map for choosing which known problem to reduce from when proving a new problem is NP-complete.

❖ Read corresponding section of textbook to familiarize yourself with the following concepts.

- **Y $\le_p X$; Y is polynomial-time reducible to X; X is as least as hard as Y.** (Chapter 8.1 p.452- p.454)

- Given the assumption that Y $\le_p X$, If X **can** be solved in polynomial time, is Y guaranteed to be solved in polynomial time? Under the same assumption, if Y **cannot** be solved in polynomial time, is it possible for X to be solved in polynomial time? (p. 453)

- What does it mean for a problem to be **in** $\mathbb{NP}$/**in** $\mathbb{P}$**?** (Chapter 8.3); how to show a problem is **in** $\mathbb{NP}$?(p.465); what is the **verifier** and **witness** for each NP-Complete problem covered in lecture; a problem is **NP-Complete** (p. 466; what are the two properties of a NP-Complete problem?); a problem is **NP-Hard** (We use the term NP-hard to mean "at least as hard as an NP-complete problem." We avoid referring to optimization problems as NP-complete, since technically this term applies only to decision problems)**.**

- What is the difference between a decision problem and an optimization problem? Which one do we care about for NP-Completeness?

❖ General Guideline to prove a problem X is NP-Complete (typically involving 3 steps):

1. Show that X is in NP. (Spoiler Alert!)
   Show that there is a polynomial time verifier for X, which takes in an instance of X and a possible answer (witness, which has to be in polynomial length), and outputs 'yes' if the answer is indeed a correct answer for X, and outputs 'no' otherwise. The verifier must output either yes or no in polynomial time.
2. Choose a problem Y that is known to be NP-Complete, and design a reduction (which must run in polynomial time) **from Y to X**, such that for any given instance of Y, the reduction will translate this instance of Y into an instance of X. In other words, you want

to show that Y $\leq_p$X. If there is no straightforward reduction, think of the concept of "gadget" which was talked about in Chapter 8.2. Be careful in reducing the wrong way round, as in that way you will be proving a known NP-Complete problem is at least as hard as a problem in NP, which is essentially repeating oneself. For instance, if Foo is the new problem, and 3-SAT is the problem that you want to reduce from, you want to reduce **FROM** 3-SAT **TO** FOO.

3. Prove that for any arbitrary instance $s_y$ of Y and corresponding instance $s_x$ of X constructed via your reduction, there exists a solution to Y **if and only if** there exists a solution to X. Note that this part of the proof goes both ways: if there is a solution to $s_x$ then there is a solution to $s_y$, and if there is a solution to $s_y$ then there is a solution to $s_x$.

Note that for each underlined part from above, you have to prove it **explicitly** to make your proof a complete one. You don't have to write a long paragraph justifying the polynomial running time bound for their verifier, another justifying the polynomial size bound for the witness string, and another justifying the polynomial running time bound for the reduction. Typically a single sentence for each these (i.e., three sentences in total) suffices. A typical sentence containing a sufficient amount of detail is: "The reduction creates a graph with 3m+2n vertices and 6m+n edges and constructing each vertex or edge of the graph only takes constant time, so the running time of the reduction is O(m+n).

Below is a quote from previous year's review sheet that I personally found very helpful in understanding how the entire reduction works: "When a problem says, 'Prove that Foo is NP-complete,' the problem is not asking you to give an algorithm to solve Foo. It is not even asking you to show how Foo can be solved, assuming that you already have a subroutine to solve 3SAT (or Hamiltonian Path, or whatever). **When you are asked to prove that Foo is NP-complete, it means you should present an algorithm for solving some other NP-complete problem, such as SAT, assuming that you already have an algorithm that solves Foo."**

*This part is partially inspired by Chapter 8.4: General Strategy for Proving New Problems NP-Complete.

- Summary for known NP-Complete problems covered in CS4820. This part is credit to a Piazza note from by Susan Li from Spring 2017.
  - Independent Set:
    Input: Undirected graph G = (V, E), integer k.
    Question: Does there exist a set of at least k nodes such that no two nodes in the set share an edge?
    Theme: Out of n items, choose AT LEAST k that don't "interfere with each other".

- Vertex Cover:
  Input: Undirected graph G = (V, E), integer k.
  Question: Does there exist a set of at most k nodes such that any edge e∈E has an endpoint in the set?
  Theme: Out of n items, choose AT MOST k that "cover things you care about"
- Hamiltonian Cycle:
  Input: Directed graph G = (V, E)
  Question: Does there exist a directed cycle in G where each node is visited exactly once?
  Theme: Sequencing a series of "items" and ending where you started
- Subset Sum:
  Input: A set of integers S = $\{w_1, \dots, w_n\}$ with target integer W.
  Question: Is there a subset of S such that the sum of these elements is W?
  Theme: Summing integers
- SAT:
  Input: Clauses $C_1, \dots, C_m$ each with $k_m$ terms/literals, a set of n variables $x_1, \dots, x_n$
  Question: Is there a truth assignment such that the conjunction of all the clauses is satisfied?
  Theme: sometimes when there is no obvious problem to reduce from, try SAT, as it is very broad in terms of satisfying constraint.

Practice Problems:

1. For each answer give a short explanation (maximum one sentence).
For the following two problems, identify if they are solvable in polynomial time or NP-complete.
(i) Given a directed graph $G = (V, E)$ with two nodes $s, t \in V$ as source and sink, is there a simple path (i.e., one with no repeat nodes) from $s$ to $t$ in this graph using at most $k$ edges?
(ii) Given a directed graph $G = (V, E)$ with two nodes $s, t \in V$ as source and sink, is there a simple path (i.e., one with no repeat nodes) from $s$ to $t$ in this graph using at least $k$ edges?

2. A monotone SAT formula is a SAT formula with no negated variables. So, for example,

$$\phi = (x_1 \lor x_2 \lor x_3) \land (x_2 \lor x_3 \lor x_4) \land (x_1 \lor x_4)$$

is a monotone formula: each of $x_1, x_2, x_3, x_4$ only appears as a positive literal, never with a negation. A monotone formula is easy to satisfy: we can simply set all variables to be true, and this way all variables in each clause become true. Since we only need one variable to be true in each clause to satisfy the formula, this valuation easily satisfies the formula.
In this problem we have a stronger goal than typical satisfiability: we want to ensure that in each monotone SAT clause, at most one variable evaluates to false (we assume each clause has at least 2 variables). This is still easy to do by setting all variables to true but becomes nontrivial if we also require our assignment to set some number of variables to false.

In the Monotone Almost-All-True SAT problem, we are given a monotone formula $\phi$ and an integer $k$. We ask if there is a way to set k variables to false (and the rest to true) such that no more than 1 variable in each clause is false. For example, for the formula above, if k = 1, then we can find an assignment satisfying these constraints by setting any one variable to false. It is impossible, however, to find an assignment that sets k = 2 variables to false while ensuring that at most one variable in each clause is false.
Show that the Monotone Almost-All-True SAT problem is NP-complete.

3. An intergalactic team of travelers has mapped out various galaxies and are currently planning their trip to visit a few of them before returning to the base station. Some galaxies seem much more interesting than others. For each galaxy $v$, they have decided on a value $w_v$ for visiting this galaxy (while return visit to a previously visited galaxy has no additional value). Unfortunately, they have limited fuel supplies, so the amount of time they can spend traveling is at most $T$ days before they return to the base station. The intergalactic team starts and ends at their base station in galaxy $s$, while visiting some other galaxies along the way.
We can represent the galaxies as nodes $V$ of a graph $G$, with $w_v \geq 0$ representing the value of visiting node $v \in V$ , node $s \in V$ is their current location, the base station. For each pair of nodes u, $v \in V$, we know the time $t_{uv}$ it takes to travel from galaxy $u$ to galaxy $v$. Travel times are luckily relatively short: for each u, v 2 V, $t_{uv} \in \{1, 2, 3\}$, measured in number of days. Travel times are not always symmetric due to high intergalactic winds: it is possible $t_{uv} \neq t_{vu}$ for some pairs of galaxies $u$ and $v$.
Starting from the base station $s \in V$ , the team sets the goal to visiting enough galaxies $A \subset V$ to get a total value $\sum_{v \in A} w_v \geq W$ before they return to base station $s$.

Show that the Galaxy Exploration problem, which decides if such a trip plan is possible, is NP-complete.