Notes 11-28

Cameron Chalk

1 The classes $\mathcal{P}$ and $\mathcal{NP}$

Recall from last lecture the class of decision problems called $\mathcal{P}$.

A problem $X \in P$ if there is a polynomial-time algorithm A such that:

- *instance s is a yes-instance of $X \iff A(s) = yes$*
- *instance s is a no-instance of $X \iff A(s) = no$*

Also recall our definition of the class of decision problems called $\mathcal{NP}$. Intuitively, we call $\mathcal{NP}$ the class of decision problems that can be efficiently verified. For example, recall the 3-colorability problem. If we are given a coloring of the graph, we can efficiently check if the coloring satisfies that no two adjacent vertices share a color (simply check each edge in $\mathcal{O}(m)$ time). However, we do not know if the problem is in $\mathcal{P}$. We can give a formal definition of $\mathcal{NP}$ as follows:

A problem $X \in \mathcal{NP}$ if there is a polynomial time algorithm A taking arguments s (the input instance for the problem) and t (the certificate *or* witness*, for example, an assignment of colors to vertices for the* 3*-coloring problem) such that:*

- *instance s is a yes-instance of $X \iff \exists t$ such that $A(s,t) = yes$*
- *instance s is a no-instance of $X \iff \forall t$ we have $A(s,t) = no$*

Note first that $\mathcal{NP}$ does not mean "not $\mathcal{P}$". (It stands for "non-deterministic polynomial time"). Note that all problems in $\mathcal{P}$ are also in $\mathcal{NP}$: if a problem is in $\mathcal{P}$, then we can take the polynomial time algorithm A, and show that the problem is in $\mathcal{NP}$ by using the same algorithm A, ignoring the extra certificate argument t.

However, we do not know if all problems in $\mathcal{NP}$ are also in $\mathcal{P}$; we have identified a lot of problems which are in $\mathcal{NP}$ which are considered difficult, and are highly conjectured to have no polynomial time solutions. However, this separation has yet to be proven. This is known as the $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$ question.

So far, we do not seem to have the tools to prove $\mathcal{P} \neq \mathcal{NP}$. However, one thing we can do is identify the problems in $\mathcal{NP}$ which are "most likely" to not

be in $\mathcal{P}$. That is, we would like to identify which problems "most likely" do not have a polynomial-time algorithm to solve them. Thus, when we encounter such a problem in our work, we will not waste too much time trying to come up with a polynomial-time algorithm to solve it.

To clarify what we mean by "most likely", we identify the following class of problems called $\mathcal{NP}$-*complete* problems. The idea is that if we could solve any one $\mathcal{NP}$-complete problem in polynomial time, then we can solve *any* problem in $\mathcal{NP}$ in polynomial time (that is, $\mathcal{P} = \mathcal{NP}$). More clearly, if $X \in \mathcal{NP}$-complete and $X \in \mathcal{P}$ then $\mathcal{P} = \mathcal{NP}$.

Recall that most computer scientists conjecture $\mathcal{P} \neq \mathcal{NP}$, so we mean "most likely" not in $\mathcal{P}$ in the sense that all of the collected evidence leading to strong conjectures of $\mathcal{P} \neq \mathcal{NP}$ would be refuted. Thus, we suggest it is not wise to spend a lot of time trying to show that an $\mathcal{NP}$-complete problem has a polynomial time solution; it is better to find an approximate solution to the problem, or try to simplify the problem.

To show that a problem is $\mathcal{NP}$-complete, we must first define what it means to solve one problem using an algorithm which solves another problem.

Polynomial time reduction: *For problems X and Y we write $X \leq_p Y$ (i.e., X is poly-time reducible to Y) if there is a poly-time algorithm $R(s_x) = s_y$ which maps instances of problem X to instances of problem Y such that:*

- *instance s_x is a yes-instance of $X \iff R(s_x)$ is a yes-instance of Y*
- *instance s_x is a no-instance of $X \iff R(s_y)$ is a no-instance of Y*

Claim: if $X \leq_p Y$ and $Y \in \mathcal{P}$ then $X \in \mathcal{P}$.
Proof: if A is a polynomial time algorithm for Y, then $A(R)$ is a polynomial time algorithm for X.

Now we can formally define $\mathcal{NP}$-completeness:

A problem X is $\mathcal{NP}$-complete if

- $X \in \mathcal{NP}$
- $\forall Z \in \mathcal{NP}$, $Z \leq_p X$

The second constraint is also called showing that the problem is $\mathcal{NP}$-hard; that is, the problem is at least as hard as every other problem in $\mathcal{NP}$. By the above claim, if you could solve X in polynomial time, then $\mathcal{P} = \mathcal{NP}$ which is highly unlikely, and therefore this definition of $\mathcal{NP}$-completeness works as a functional definition of "most likely not in $\mathcal{P}$".

Showing the second constraint is very hard; you have to show that **EVERY SINGLE PROBLEM IN** $\mathcal{NP}$ is poly-time reducible to your problem X (note: there are infinitely many problems in $\mathcal{NP}$). Luckily, this hard work was done for one problem, the first $\mathcal{NP}$-complete problem (Levin & Cook 1971) (you don't need to understand this construction in detail, but it is important

to know that there is a complicated proof to show that the problem is $\mathcal{NP}$-complete):

CircuitSAT: *Given a Boolean circuit represented as a graph, is there a way to set inputs such that it outputs a one?*

The intuition is that any algorithm can be reduced to their operations on AND, NOT, and OR gates. Thus we can implement a verification algorithm A for $Z \in \mathcal{NP}$ as a circuit. The input to the circuit is a witness t. The circuit outputs one iff there exists a witness that makes A output yes.

Given that we have one $\mathcal{NP}$-complete problem CircuitSAT, we can do the following. To show that a problem X is $\mathcal{NP}$-complete, it is enough to show:

- $X \in \mathcal{NP}$
- some known $\mathcal{NP}$-complete problem $\leq_p X$

The key is that $\leq_p$ is transitive. For example, call the known $\mathcal{NP}$-complete problem Y. We know that since Y is $\mathcal{NP}$-complete, $\forall Z \in \mathcal{NP}$, $Z \leq_p Y$. Therefore if we show that $Y \leq_p X$, because $\leq_p$ is transitive, then $\forall Z \in \mathcal{NP}$, $Z \leq_p X$, which is second constraint to show the problem is $\mathcal{NP}$-complete.