CS 341 — Lecture 18

Bartosz Antczak *Instructor: Bin Ma* *November 9, 2017*

18.1 Polynomial Time Reduction

The main idea of PTR is to answer the question: *how do we prove that a problem is hard?* Given some problem P, we reduce a known hard problem (many of them exist) to P.

Recall Reduction

If we reduce problem A to problem B, we are reducing the problem instance of A such that the algorithm which solves B can also solve the reduced instance of A.

Example 18.1.1. *Reducing the Longest Increasing Subsequence problem*

LIS: find the longest increasing subsequence in a list A. This can be done using the following algorithm:

---

sort A into a new list B
return LongestCommonSubsequence(A, B)

From this, we can conclude that LIS is just as difficult to solve as LCS.

Example 18.1.2. *Maximum Matching*

Given a graph $G = (V, E)$, we define a **matching** as a subset $M \subset E$ such that the edges in M do not share any vertices.
Now, if we have an algorithm A that can decide if the size of the maximum matching M of G is greater than or equal to k (this is a decision problem), then we can use the following technique to determine what the matching actually is:

1. Perform binary search on the list $\{0, 1, \ldots, \frac{n}{2}\}$ to determine what the actual optimal size is.
2. Remove all edges from the graph where $A(E/\{e\}, k)$ returns yes. The resulting graph will have k edges that represent the max matching.

18.1.1 Definition

A decision problem A is **polynomial time reducible** to B if for every instance I_A of A, there exists a polynomial time mapping $f : I_A \to I_B$ such that I_A is a *yes* instance $\iff$ $f(I_A)$ is a *yes* instance.
So how do we use this to prove that a particular problem P is hard? We simply show that a hard problem H is polynomial time reducible to P. Mathematically, the reduction would look like this:

- Define g as an algorithm that solves H. Also, define the run time of $g = O(m^\alpha)$ (consider this a hard run time). We take a problem instance I_H of our hard problem and map it to our problem P: $f(I_H)$. The time it takes to perform this mapping is polynomial by definition; $|f(I_H)| = O(n^c)$. Now we run the algorithm on our modified problem instance to retrieve the run time $g(f(I_H)) = O(n^c)^\alpha = O(n^{c \cdot \alpha})$

This means that $H \leq_p P$, which means that P is **not** easier than A. ($\leq_p$ means *is poly. time reducible*)

Example 18.1.3. *Clique and Independent Set*

Let's do another example to show how PTR works.
A **clique** in a graph G is a subgraph where every vertex pair (u, v) has an edge. An **independent set** is a subset of vertices such that no two vertices in the set have an edge connecting them.
We want to prove that finding a clique $\leq_p$ finding an independent set.

Proof

Let $I = (G, k)$ be an instance for clique. Take $G = (V, E)$ and construct a new graph $\overline{G} = (V, \overline{E})$ where $\overline{E} = \{(u, v) : (u, v) \notin E\}$. Now I get a new instance $I = (\overline{G}, k)$ for IS. *(Prof. Ma said we'll go over this proof in more detail next class I hope)*