CMPSCI 611 Advanced Algorithms **Fall 2000**

Lecture 17: November 9, 2000

Lecturer: Micah Adler *Scribes: Anoop George Ninan, Joe Daverin*

17.1 The Classes P and NP

We had a brief introduction to the classes **P** and **NP** in the previous lecture. We shall review the definitions of **P** and **NP** before we proceed.

Definition 17.1 *A decision problem* Π *is in* **P** *if and only if there is a polynomial time algorithm* A *such that* (X *is a "YES" instance of* Π) $\Leftrightarrow$ ($A(X)$ = *"YES"*).

P is said to be a class of *tractable* decision problems, since it turns out that there are usually low-degree polynomial time algorithms to solve them.

Definition 17.2 *A decision problem* Π *is in* **NP** *if and only if there is a polynomial time algorithm* A *such that* (X *is a "YES" instance of* Π) $\Leftrightarrow$ ($\exists Y$ *that is polynomial in the size of the input* $X, (|Y| = O(|X|^k))$, *and* $A(X, Y)$ = *"YES"*.

Y is a *witness or proof* that convinces us in polynomial time that $A(X)$ = "YES". **NP** is said to be a class of *intractable* decision problems since we do not know how to find Y efficiently.

Note: P $\subseteq$ NP. To show this, given a decision problem $\Pi \in$ **P** (i.e., there is a polynomial time algorithm A such that $A(X)$ = "YES" if and only if X is a "YES" instance of Π), use $A(X, Y)$ and ignore Y. ($A(X, Y)$ = "YES") $\Rightarrow$ (**P** $\subseteq$ **NP**).

17.2 Examples of Problems in NP

17.2.1 The Max-Cut Problem

The Max-Cut problem is as follows:

Input: A graph G and an integer k.

Question: Is the max-size cut in $G \geq k$?

Claim 17.3 *The Max-Cut problem is in* **NP**.

Proof: Approach: Show that there is a polynomial time algorithm $A(X, Y)$, which verifies that Y is a valid description of a cut in G and that $|Y| \geq k$.

If $A(X, Y)$ = "YES", Y is a cut of size $\geq k$.

If $A(X, Y)$ = "NO", there is no such cut. ■

17.2.2 The Longest-Path Problem

The Longest-Path problem is this:

Input: A graph G, two nodes $a, b \in G$, and an integer k.

Question: Is there a path from a to b in G of length $\geq k$?

Claim 17.4 *The Longest-Path problem is in* **NP**.

Proof: Approach: Show that there is a polynomial time algorithm $A(X, Y)$, which verifies that Y is a valid description of a path in G from a to b and that $|Y| \geq k$.

If $A(X, Y) =$ "YES", Y is a path of length $\geq k$.

If $A(X, Y) =$ "NO", there is no such path. ■

Note: If we have a problem $\Pi \in \mathbf{NP}$, nothing guarantees us that $\Pi \in \mathbf{P}$. All we know is that Π does have an exponential time solution (i.e., consider all possibilities). We next present a technique that helps us provide *evidence* that a given problem belongs to **P**.

17.3 Polynomial Time Reduction

GOAL: We want to be able to say that if $\Pi_1 \in \mathbf{P}$ then $\Pi_2 \in \mathbf{P}$.

Why do we have such a goal?

1. We know that we can solve Π_2 efficiently if we know that $\Pi_1 \in \mathbf{P}$.
2. If $\Pi_2 \notin \mathbf{P}$ then $\Rightarrow \Pi_1 \notin \mathbf{P}$ leads to a contradiction.

Definition 17.5 *Π_2 is polynomial time reducible to Π_1, ($\Pi_2 \leq_p \Pi_1$), if and only if there is a polynomial time algorithm f that transforms any instance X of Π_2 to an instance $f(X)$ of Π_1 such that (X is a "YES" instance of Π_2) $\Leftrightarrow$ ($f(X)$ is a "YES" instance of Π_1).*

Note: The function f above maps all "YES" instances of Π_2 to the "YES" instances of Π_1 and all the "NO" instances of Π_2 to the "NO" instances of Π_1.

From this definition, we get the following results:

1. If $\Pi_2 \leq_p \Pi_1$ and $\Pi_1 \in \mathbf{P}$, then $\Pi_2 \in \mathbf{P}$.
2. If $\Pi_2 \leq_p \Pi_1$ and $\Pi_2 \notin \mathbf{P}$, then $\Pi_1 \notin \mathbf{P}$.

As stated earlier, we shall use the above results to provide *evidence* that problems are not in **P**.

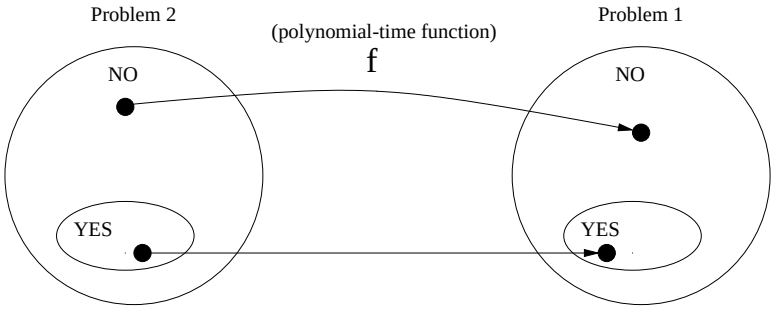

Figure 17.1: An illustration of polynomial time reduction.

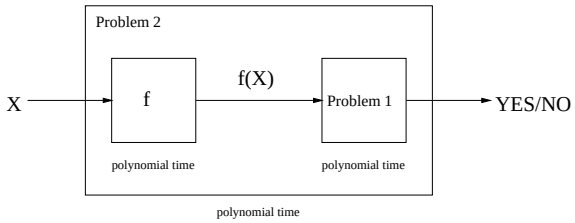

Figure 17.2: Diagrammatic proof of result 1 (above).

17.4 NP-Completeness

To define **NP–Completeness**, we first define what **NP–Hard** is, and then define **NP–Completeness**.

Definition 17.6 *A decision problem* Π *is* **NP–Hard** *if and only if* $\forall \Pi' \in \mathbf{NP}, \Pi' \leq_p \Pi$.

Definition 17.7 *A decision problem* Π *is* **NP–Complete**, $(\Pi \in \mathbf{NPC})$, *if and only if* Π *is in* **NP** *and* Π *is in* **NP–Hard**.

Note: If $\Pi \in \mathbf{NPC}$ and $\Pi \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$. See [CLR90, page 932] for proof.

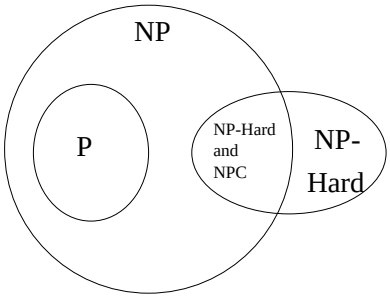

Figure 17.3: Relationships amongst $\mathbf{P}, \mathbf{NP}, \mathbf{NPC}$ and **NP–Hard**.

To show that a problem Π is in **NPC** we use the following technique:

1. Show that $\Pi \in$ **NP**.
2. Find some Π^* already known to be in **NPC**. Show that $\Pi^* \leq_p \Pi$.

Step 2 above is reasonable because we know that $\Pi' \leq_p \Pi^*, \forall \Pi' \in$ **NP**. So all we need to show is that $\Pi^* \leq_p \Pi$.

17.5 Examples of problems in NPC

17.5.1 SAT

The satisfying assignment problem, SAT, is as follows:

Input: A boolean formula ϕ (a formula of boolean variables with operations $\neg$, $\vee$, $\wedge$).

Question: Is ϕ satisfiable?

Here are two examples of boolean formulas:

- $\phi = (X_1 \wedge X_2) \vee (\neg X_2)$, which is satisfiable.
- $\phi = X_1 \wedge X_2 \wedge (\neg X_1)$, which is not.

Theorem 17.8 *SAT* $\in$ **NPC**, *by Cook ['71].*

The proof of the above theorem is beyond the scope of these notes. Please refer [CLR90, page 941] for details.

17.5.2 3-SAT

3-SAT is a more specific form of SAT.

Input: A boolean formula ϕ that is in conjunctive normal form (CNF), with 3 literals per clause.

Question: Is ϕ satisfiable?

What it means for a formula ϕ to be in 3-CNF:

- ϕ takes the form $X_1 \wedge X_2 \wedge X_3 \wedge \ldots \wedge X_n$, where
- each X_i takes the form $(l_{i1} \vee l_{i2} \vee l_{i3})$, where
- each l_{ij} is a literal, and
- each l_{ij} can be l_{ij} or $\neg l_{ij}$.

Claim 17.9 *3-SAT is in* **NPC**. *This can be proved by showing that SAT* $\leq_p$ *3-SAT.*

17.5.3 CLIQUE

Definition 17.10 *A clique of size k in a graph G is a completely connected subgraph of G with k vertices.*

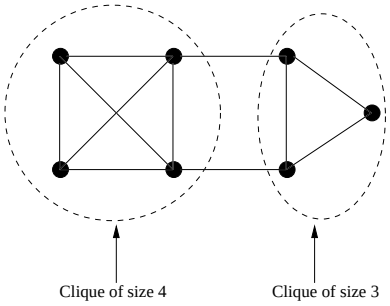

Figure 17.4: Example of cliques.

The CLIQUE problem is as follows:

Input: A graph G and an integer k.

Question: Does G contain a clique of size $\geq k$?

Theorem 17.11 *Clique is in* **NPC**.

Proof: Approach:

- Prove CLIQUE is in **NP**. We do this by creating an algorithm $A(X, Y)$ that interprets Y as a valid set of vertices of the graph G, verifies that all of the required edges are indeed in G, and verifies that $|Y| \geq k$.
- Prove that 3-SAT $\leq_p$ CLIQUE. The idea here is to take as input any 3-CNF formula ϕ, and transform it in polynomial time, into G_ϕ, k_ϕ such that G_ϕ has a clique of size k_ϕ if and only if ϕ is satisfiable. We construct $G_\phi, k_\phi = (V_phi, E_\phi), k_\phi$ as follows:
  - k_ϕ is determined by the number of clauses in ϕ.
  - $V_\phi = n_{ij}$ correspond to literals l_{ij} in ϕ.
  - $E_\phi = \{\langle n_{ij}, n_{hk}\rangle : i \neq h \wedge l_{ij} \neq \neg l_{hk}\}$.

This proof is continued in the next lecture. ■

References

CLR90 T. Cormen, C. Leiserson, and R. Rivest, Introduction to Algorithms, MIT Press, 1990.