Computer Science 604

Advanced Algorithms

Lecture 2: Computational Hardness —

Review of NP-completeness

David Juedes

School of EECS

`juedes@cs.ohiou.edu`

Computational Hardness

Showing that a computational problem likely does not have an efficient algorithm usually involves *modeling*, i.e., showing that problem has an "efficient" algorithm under some measure (which may not be practical).

In the case of NP-completeness, we first show that the problem in question can be solved via a *nondeterministic polynomial-time* algorithm, and hence is in NP. *Nondeterministic polynomial-time* is a *model* of "efficient" computation that should not be confused with traditional (deterministic) algorithms.

The existence of a *nondeterministic polynomial-time* algorithms does not mean that there is not a better way to solve the problem.

NP == Fast Witness Verification

We can model all problems in NP via deterministic algorithms that take **witnesses**.

Let A be an algorithm that takes two inputs (x,y), where the length of y is bounded by a polynomial in the length of x. If the algorithm A runs in polynomial time, then the language L defined by

$$x \in L \Leftrightarrow \exists \text{ a } y \text{ s.t. } (x, y) \text{ is accepted by } A$$

is in NP. This is the *witness* characterization of NP.

This is equivalent to other

Some Examples

Problem: Independent set (ISP)

Instance: A graph $G = (V, E)$ and integer k.

Question: Is there an independent set in G of size k?

Claim: ISP is in NP.

Witness Characterization

What witness allows to conclude that G has an independent set of size k?

How long does it take to verify this statement?

Another Example

Problem: Traveling Salesman Problem (TSP)

Instance: A complete directed graph $G = (V, E)$ with n vertices, $V = \{1, 2, \ldots, n\}$, a function *l* that assigns a length, $l_{i,j}$, to each edge (i, j) and a real number L.

Question: Does G have a Hamiltonian tour of size $\leq L$?

Try showing that this problem is in NP.

Witness Characterization

What witness allows to conclude that G has a TSP tour of length $\leq L$?

How long does it take to verify this statement?

Formal Languages

- All of the concepts developed so far (P, NP, etc.) can be put into the more rigorous framework of formal languages and computational complexity through the notion of encoding.

- Briefly, given a decision problem π, let Σ_π be a finite alphabet (usually, $\Sigma_\pi = \{0, 1\}$).

- A mapping $e : D_\pi \longrightarrow (\Sigma_\pi)^*$ is called an encoding if it is *injective*, i.e., a 1-to-1 function.

Formal Lang.

- Thus, e and π define a language:

$$L(\pi, e) = \{e(I) \in (\Sigma_\pi)^* | I \in Y_\pi\}$$

- In a strong, sense, the underlying theory identifies $L(\pi, e)$ with π. To make the underlying theory work, it is required that the encoding be *"reasonable"*, i.e., no useless padding, numbers are in unary, computable in polynomial-time, and its inverse should be computed in polynomial-time (decoding).

- Then, we can define $\pi \in P$ or $\pi \in NP$ if there is a reasonble encoding e such that $L(\pi, e) \in$ P or $L(\pi, e) \in$ NP.

Structural Complexity

The great promise of structural complexity is that we can learn something *quantitative* about the complexity of *individual problems* by studying the relationships among various classes of problems. This promise relies on notions of **reducibility** and **completeness**.

Reducibility

Let I_X and I_Y be sets of instances of the problems X and Y, respectively. A *reduction* from X to Y is a function

$$f : I_X \longrightarrow I_Y$$

such that

$$x \in X \Leftrightarrow f(x) \in Y.$$

Reductions

A problem X is reducible to Y in polynomial-time (and we write $X \leq_m^P Y$) if there is a reduction f from X to Y that is computable in polynomial-time.

Examples

$$E = \{x \in N | x \text{ is even } \}$$

$$O = \{x \in N | x \text{ is odd } \}.$$

$$E \leq_m^P O \text{ via } f(x) = x + 1$$

$$O \leq_m^P E \text{ via } f(x) = x + 1$$

Completeness

- A problem $C \in NP$ is *NP-complete* if every problem L $\in$ NP is reducible in polynomial-time to C, i.e.,

$$\forall L \in \text{NP}, L \leq_m^P C.$$

- Notice that
  1) If C is NP-complete and C $\in$ P, then
  P = NP
  2) If P $\neq$ NP, then C is not computable in polynomial-time, i.e., every NP-complete language is not computable in polynomial-time.

Important Properties

Definition: A class of languages $\mathcal{C}$ is *closed under* $\leq_m^p$ if $L \leq_m^p K$ and $K \in \mathcal{C}$ implies that $L \in \mathcal{C}$. In other words, a class $\mathcal{C}$ is closed under $\leq_m^p$ if any language that is reducible to a language in $\mathcal{C}$ is also in $\mathcal{C}$.

Property 1: P and NP are closed under $\leq_m^p$

Proof: Try this at home.

Properties

Property 2: ($\leq_m^p$ is transitive) If $L_1 \leq_m^p L_2$ and $L_2 \leq_m^p L_3$ then $L_1 \leq_m^p L_3$.

Proof: Try this at home.

Prop. # 3

Property 3: If (i.) $L, K \in$ NP, (ii.) L is NP-complete, and (iii.) $L \leq_m^p K$, then K is NP-Complete

Proof: (Easy) If L is NP-Complete then for every $A \in$ NP, $A \leq_m^p L$. Since $L \leq_m^p K$ and $\leq_m^p$ is transitive, it follows that $A \leq_m^p K$. Thus K is NP-complete. $\square$