Computer Science 511

What If...? More about Nondeterministic Computation

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #13

Goals for Today

Even though researchers have now been trying to answer this question for many years, now, we still do not know whether $\mathcal{P} = \mathcal{NP}$. In this lecture...

- ***$\mathcal{NP}$-Intermediate Languages*** are defined... and proved to exist, if $\mathcal{P} \neq \mathcal{NP}$.
- The complexity class *NEXPTIME* is introduced, and it is explained why this complexity class *cannot* be different from *EXPTIME*, if $\mathcal{P} = \mathcal{NP}$.
- ***Relativized complexity classes*** are considered and significant result concerning these is introduced. This, effectively, *eliminates* some approaches to proving either that $\mathcal{P} = \mathcal{NP}$, or that $\mathcal{P} \neq \mathcal{NP}$, that might otherwise be considered.

Goals for Today

- Unfortunately, ***proofs of some of these results are extremely long and complicated***.

  Students will be expected to understand the meaning of significant technical results, and their implications — and *not* expected to understand (or even to have looked at) proofs of these results.

What if $\mathcal{P} \neq \mathcal{NP}$? $\mathcal{NP}$-Intermediate Languages

Definition: A language $L \subseteq \Sigma^\star$ is ***$\mathcal{NP}$-Intermediate*** if it satisfies the following properties.

(a) $L \in \mathcal{NP}$.

(b) $L \notin \mathcal{P}$.

(c) L is not $\mathcal{NP}$-complete.

Note that if $\mathcal{P} = \mathcal{NP}$ then no such languages exist, because properties (a) and (b) are contradictory.

What if $\mathcal{P} \neq \mathcal{NP}$? $\mathcal{NP}$-Intermediate Languages

Claim: If $\mathcal{P} \neq \mathcal{NP}$ then an $\mathcal{NP}$-intermediate language exists.

- This is proved by describing an artificial (and confusing) language, with the following properties:
  - This is a language, $L \subseteq \Sigma_{F,\#}^{\star}$ for $\Sigma_{F,\#} = \Sigma_F \cup \{\#\}$, where Σ_F is the alphabet used to define encodings of Boolean formulas, used in recent lectures.
  - L includes strings of the form $\omega \#^{g(|\omega|)}$ where $\omega \in L_{\text{FSAT}}$ and $g : \mathbb{N} \to \mathbb{N}$ is a total function (that is *not* easy to describe, and only of interest because it is needed for this proof).
  - $L \in \mathcal{NP}$.
  - If $L \in \mathcal{P}$ then $L_{\text{FSAT}} \in \mathcal{P}$ as well, so that $\mathcal{P} = \mathcal{NP}$.
  - On the other hand, of L is $\mathcal{NP}$-hard then $L_{\text{FSAT}} \in \mathcal{P}$, and $\mathcal{P} = \mathcal{NP}$, once again.

It follows from the above that if $\mathcal{P} \neq \mathcal{NP}$ then L must be an $\mathcal{NP}$-intermediate language.

What if $\mathcal{P} \neq \mathcal{NP}$? $\mathcal{NP}$-Intermediate Languages

- No “naturally arising” languages that would, provably, be $\mathcal{NP}$-intermediate if $\mathcal{P} \neq \mathcal{NP}$, are currently known.
- A reasonably “natural” language, that is ***believed*** to be $\mathcal{NP}$-intermediate, is described next.

What if $\mathcal{P} \neq \mathcal{NP}$? $\mathcal{NP}$-Intermediate Languages

Definition: If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are undirected graphs then G_1 is ***isomorphic*** to G_2 if there exists a bijection $\varphi : V_1 \to V_2$ such that, for all vertices $u, v \in V_1$, $(u, v) \in E_1$ if and only if $(\varphi(u), \varphi(v)) \in E_2$.

Consider the following decision problem:

> ***Graph Isomorphism***
>
> *Instance:* A pair (G_1, G_2) of undirected graphs
>
> *Question:* Is G_1 isomorphic to G_2?

What if $\mathcal{P} \neq \mathcal{NP}$? $\mathcal{NP}$-Intermediate Languages

- An instance (G_1, G_2) of this problem can be encoded, as a string in $\Sigma_G^\star$, by encodings of G_1 and G_2 (as defined in Lecture #12), separated by a comma
- The language $L_{\text{Iso}} \subseteq L_G^\star$ is one of small number of "natural" languages that are *suspected* to be $\mathcal{NP}$-intermediate — and a "quasi-polynomial time" algorithm (using $O(2^{(\log n)^c})$ steps in the worst case, for inputs with size n, in the worst case) was announced in 2015.
- At present, no "natural" languages have been *proved* to be $\mathcal{NP}$-intermediate (assuming $\mathcal{P} \neq \mathcal{NP}$).

What if $\mathcal{P} = \mathcal{NP}$? Complexity Classes Collapse

- Recall that

$$EXPTIME = \bigcup_{k \geq 1} \mathrm{TIME}\left(2^{(n^k)}\right).$$

- We can also define

$$NEXPTIME = \bigcup_{k \geq 1} \mathrm{NTIME}\left(2^{(n^k)}\right).$$

- Results given in Lecture #8 can be used to establish that $EXPTIME \subseteq NEXPTIME$. It is not known whether $EXPTIME \subsetneq NEXPTIME$ — but it is suspected that this is true.

What if $\mathcal{P} = \mathcal{NP}$? Complexity Classes Collapse

Claim: If $\mathcal{P} = \mathcal{NP}$ then $\mathit{EXPTIME} = \mathit{NEXPTIME}$.

Sketch of Proof: This involves another — simpler — use of "padding".

- Suppose that $\mathcal{P} = \mathcal{NP}$, and let $L \subseteq \Sigma^\star$ such that $L \in \mathit{NEXPTIME}$. It is necessary and sufficient to prove that $L \in \mathit{EXPTIME}$.
- Since $L \in \mathit{NEXPTIME}$ there exists a nondeterministic Turing machine M, with input alphabet Σ, and positive integer constants c and k, such that
  - M decides L, and
  - for every string $\omega \in \Sigma^\star$, the depth of the computation tree of M and ω is at most $c \cdot 2^{(n^k)}$.

What if $\mathcal{P} = \mathcal{NP}$? Complexity Classes Collapse

- Let $\widehat{\Sigma} = \Sigma \cup \{\#\}$ and let

$$L_{\mathrm{pad}} = \left\{ \omega \#^{c \cdot 2^{\left(|\omega|^k\right)}} \;\middle|\; \omega \in L \right\} \subseteq \widehat{\Sigma}^\star$$

— so that if $\mu \in \widehat{\Sigma}^\star$ then $\mu \in L_{\mathrm{pad}}$ if and only if μ begins with a string $\omega \in L$ and continues (and ends) with $c \cdot 2^{\left(|\omega|^k\right)}$ copies of #.

What if $\mathcal{P} = \mathcal{NP}$? Complexity Classes Collapse

Subclaim #1: $L_{\mathrm{pad}} \in \mathcal{NP}$.

How To Prove This: Consider a nondeterministic Turing machine that does the following on input $\mu \in \widehat{\Sigma}^\star$:

1. ***Reject*** unless $\mu = \omega\#^{c \cdot 2^{\left(|\omega|^k\right)}}$ for some string $\omega \in \Sigma^\star$.
2. If μ was not rejected, run M using ω as input — ***accepting*** μ if M accepts ω and ***rejecting*** μ otherwise.

It can be argued that — since M decides L, and the computation tree for $\mathcal{M}$ and ω has depth at most $c \cdot 2^{|\omega|^k}$ — a nondeterministic Turing machine, implementing this algorithm, decides L_{pad} using polynomial time, because $|\mu| = |\omega| + c \cdot 2^{|\omega|^k}$ if the step at line 2 is reached and executed.

Thus $L_{\mathrm{pad}} \in \mathcal{NP}$. $\square$

What if $\mathcal{P} = \mathcal{NP}$? Complexity Classes Collapse

- Since $\mathcal{P} = \mathcal{NP}$ (by assumption) it follows that $L_{\text{pad}} \in \mathcal{P}$.
- Thus there exists a ***deterministic*** Turing machine $\widehat{M}$ with input alphabet $\widehat{\Sigma}$, and positive integer constants $\widehat{c}_1$, d and $\widehat{c}_0$, such that
  - $\widehat{M}$ decides L_{pad}, and
  - for any string $\mu \in \widehat{\Sigma}^\star$, $\widehat{M}$ halts, ,when executed on input μ, after making at most $\widehat{c}_1 \cdot |\mu|^d + \widehat{c}_0$ moves.

What if $\mathcal{P} = \mathcal{NP}$? Complexity Classes Collapse

Subclaim #2: $L \in$ *EXPTIME*.

How To Prove This: Consider a deterministic Turing machine that does the following on input $\omega \in \Sigma^\star$:

1. Pad ω with copies of # to produce the string $\mu = \omega\#^{c \cdot 2^{\left(|\omega|^k\right)}}$.
2. Run $\widehat{M}$ on input μ, ***accepting*** ω if $\widehat{M}$ accepts μ, and ***rejecting*** ω otherwise.

- Since $\widehat{M}$ decides L_{pad}, this Turing machine decides L.

What if $\mathcal{P} = \mathcal{NP}$? Complexity Classes Collapse

- The number of moves, used by this Turing machine, is dominated by the number of moves used in step 2. Since $\widehat{M}$ is a fixed Turing machine that can be "embedded" (used as subroutine) in this one, this is at most

$$\widehat{c}_1 \cdot \left(|\omega| + c \cdot 2^{\left(|\omega|^k\right)} \right)^d + \widehat{c}_0 \in O\left(2^{\left(|\omega|^{kd}\right)} \right).$$

- Since k and d are positive integer constants, so is kd — and it follows that $L \in EXPTIME$, as claimed. $\square$

What if $\mathcal{P} = \mathcal{NP}$? Complexity Classes Collapse

Conclusion of Proof of the Claim:

- Since L was arbitrarily chosen from *NEXPTIME*, it follows that $\mathit{NEXPTIME} \subseteq \mathit{EXPTIME}$.
- Since $\mathit{EXPTIME} \subseteq \mathit{NEXPTIME}$ as well,

$$\mathit{EXPTIME} = \mathit{NEXPTIME},$$

as claimed. $\square$

What if $\mathcal{P} = \mathcal{NP}$? Complexity Classes Collapse

- Virtually the same proof can be applied (with different amounts of "padding") to prove that if $\mathcal{P} = \mathcal{NP}$ then deterministic- and nondeterministic- complexity classes "collapse" together at higher levels too.
- For example, one could modify this argument so establish that if $\mathcal{P} = \mathcal{NP}$ then the set of languages that are deterministically decidable in "doubly exponential time" is the same as the set of languages that are nondeterministically decidable in "doubly exponential time" too.

Relativized Complexity Classes — and Limits of Strategies for Proofs

- Recall, from Lecture #6, that an ***oracle for a language*** $\boldsymbol{L \subseteq \Sigma_L^\star}$ is a device that is capable of reporting whether any string $\omega \in \Sigma_L^\star$ is a member of L .
- Recall, as well, that an ***oracle Turing machine*** $\boldsymbol{M_L}$ ***with an oracle for a language*** $\boldsymbol{L \subseteq \Sigma_L^\star}$ is a modified deterministic multi-tape Turing machine that is allowed to query an oracle for L in a single step.
- See Lecture #6 for additional details about oracle Turing machines.

Relativized Complexity Classes — and Limits of Strategies for Proofs

- Recall that we consider accesses to the oracle to have unit cost, just like applications of other transitions of a Turing machine.
- Just as for ordinary Turing machines, we can define the ***time*** used ay a (one-tape or multi-tape) Turing machine M, with an oracle for a language $L \subseteq \Sigma_L^\star$, on input ω to be the number of steps that M takes, using its oracle for L, when executed on the input string ω, before it halts.
- The ***worst-case running time*** of M can be defined as a function $T_M : \mathbb{N} \to \mathbb{N}$ in the same way as for an ordinary (one-tape or multi-tape) Turing machine, as well.

Relativized Complexity Classes — and Limits of Strategies for Proofs

- Continuing to modify definitions from previous lectures, we can define $\mathsf{TIME}_1^L(f)$, $\mathsf{TIME}_2^L(f)$ and $\mathsf{TIME}^L(f)$ by replacing references to ordinary (one-tape, two-tape and multi-tape) Turing machines in the definitions of $\mathsf{TIME}_1(f)$, $\mathsf{TIME}_2(f)$ and $\mathsf{TIME}(f)$ with references to one-tape Turing machines with an oracle for the language $L \subseteq \Sigma_L^\star$, two-tape Turing machines with an oracle for L, and multi-tape Turing machines with an oracle for L, respectively.
- We can then define

$$\mathcal{P}^L = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}^L(n^k).$$

Relativized Complexity Classes — and Limits of Strategies for Proofs

Definition: A ***nondeterministic oracle Turing machine*** $\boldsymbol{M_L}$ ***with an oracle for a language*** $\boldsymbol{L \subseteq \Sigma_L^\star}$ is a modified nondeterministic multi-tape Turing machine that is modified (by adding a query tape, and three special tapes used to access the oracle) in the same way that a multi-tape deterministic Turing machine is modified to obtain an oracle Turing machine with an oracle for L.

Relativized Complexity Classes — and Limits of Strategies for Proofs

- Once again, definitions for "standard" nondeterministic Turing machines, given in Lecture #8, can be modified, by replacing references to (multi-tape) nondeterministic Turing machines with oracles, in order to say what it means for a nondeterministic oracle Turing machine with an oracle for a language $L \subseteq \Sigma_L^\star$ to ***decide*** another language $\widehat{L}$, and to define the complexity classes $\mathrm{NTIME}^L(f)$, for a function $f : \mathbb{N} \to \mathbb{N}$, as well as the complexity class

$$\mathcal{NP}^L = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}^L(n^k).$$

Relativized Complexity Classes — and Limits of Strategies for Proofs

- We say that (correct) proofs of results about computation — and the results, themselves — ***relativize*** if they are still correct if regular (deterministic or nondeterministic) Turing machines, are used in the proofs and claims, by (deterministic or nondeterministic) Turing machines with oracles.
- While the following result *does not* imply that it is impossible to prove either that $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$, it *does* eliminate the possibility that this question will be resolved in various ways — because no proof of either of these claims can relativize.

Relativized Complexity Classes — and Limits of Strategies for Proofs

Theorem (Baker, Gill, and Solovay):

(a) There exists a language $A \subseteq \Sigma_A^\star$ (for some alphabet Σ_A) such that $\mathcal{P}^A = \mathcal{NP}^A$.

(b) There exists a language $B \subseteq \Sigma_B^\star$ (for some alphabet Σ_B) such that $\mathcal{P}^B \neq \mathcal{NP}^B$.

Unfortunately, the proof of *this* claim is also quite complicated. While details of the proof are given in a supplement for this lecture it is (once again) "for interest only" — students will not be expected to have looked at this material.