Advanced Computational Complexity

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 600

1. Diagonalization
   - Time Hierarchy Theorem
   - Nondeterministic Time Hierarchy Theorem
   - Ladner's Theorem: Existence of NP-Intermediate Problems
   - Oracle Machines and the Limits of Diagonalization

Introducing Diagonalization

- We would like to be able to prove that certain complexity classes (e.g., P and NP) are not the same.
- This requires exhibiting a machine in one class that differs from every machine in the other class.
- Being different means that their answers are different on at least one input.
- The only general technique known for constructing such a machine is **diagonalization**.

Introducing Diagonalization (Cont'd)

- The one common tool used in all diagonalization proofs is the representation of TMs by strings.
- It is effective, in the sense that there is a universal TM that, given any string x, can simulate the machine represented by x with a small - at most logarithmic - overhead.
- Every string $x \in \{0,1\}^*$ represents some TM, denoted M_x.
- Every TM is represented by infinitely many strings.
- We use the notation M_i, where $i \in \mathbb{N}$, for the machine represented by the string that is the binary expansion of the number i, without the leading 1.

Subsection 1

Time Hierarchy Theorem

The Time Hierarchy Theorem

- The Time Hierarchy Theorem shows that allowing Turing machines more computation time strictly increases the set of languages that they can decide.
- Recall that for a function $f : \mathbb{N} \to \mathbb{N}$, $\mathrm{DTIME}(f(n))$ is the set of languages decided by a TM running in time $O(f(n))$.
- We only deal with time-constructible functions f, which means that the mapping $x \mapsto f(|x|)$ can be computed in $O(f(n))$ time.

Theorem (Time Hierarchy Theorem)

If f, g are time-constructible functions, satisfying $f(n) \log f(n) = o(g(n))$, then

$$\mathrm{DTIME}(f(n)) \subsetneq \mathrm{DTIME}(g(n)).$$

Proof of the Time Hierarchy Theorem

- We prove the simpler statement $\mathrm{DTIME}(n) \subsetneq \mathrm{DTIME}(n^{1.5})$.

  Consider the following Turing machine D.

  - Suppose it receives input x.
  - It uses the Universal TM $\mathcal{U}$ to simulate the execution of M_x on x for $|x|^{1.4}$ steps.
  - Suppose $\mathcal{U}$ outputs some bit $b \in \{0, 1\}$ in this time.
  - Then D outputs the opposite answer, i.e., $1 - b$.
  - Otherwise, it outputs 0.

  D halts within $n^{1.4}$ steps.

  So, the language L decided by D is in $\mathrm{DTIME}(n^{1.5})$.

Proof of the Time Hierarchy Theorem (Cont'd)

- Claim: $L \notin \text{DTIME}(n)$.

  Assume there is some TM M and constant c, such that TM M, given any input $x \in \{0,1\}^*$, halts within $c|x|$ steps and outputs $D(x)$.

  The time to simulate M by the universal Turing machine $\mathcal{U}$ on every input x is at most

  $$c'c|x|\log|x|,$$

  where c' is a constant that:

  - Depends on the alphabet size and number of tapes and states of M;
  - Is independent of $|x|$.

Proof of the Time Hierarchy Theorem (Cont'd)

- There is some number n_0, such that

$$n^{1.4} > c'cn\log n, \quad \text{for every } n \geq n_0.$$

  Let x be a string representing the machine M, with $|x| \geq n_0$.

  Such a string exists since M is represented by infinitely many strings.

  Then, $D(x)$ will obtain the output $b = M(x)$ within $|x|^{1.4}$ steps.

  However, by definition of D, we have

$$D(x) = 1 - b \neq M(x).$$

  This yields a contradiction.

- The proof for general f, g is similar and uses the observation that the slowdown in simulating a machine using $\mathcal{U}$ is at most logarithmic.

Subsection 2

Nondeterministic Time Hierarchy Theorem

The Nondeterministic Time Hierarchy Theorem

Theorem (Nondeterministic Time Hierarchy Theorem)

If f, g are time constructible functions, satisfying $f(n+1) = o(g(n))$, then

$$\mathrm{NTIME}(f(n)) \subsetneq \mathrm{NTIME}(g(n)).$$

- We only prove $\mathrm{NTIME}(n) \subsetneq \mathrm{NTIME}(n^{1.5})$.

  The first instinct is to mimic the deterministic case, since there is a universal TM for nondeterministic computation as well.

  This is not sufficient because the definition of the new machine D requires the ability to "flip the answer".

  That is, to efficiently compute, given the description of a NDTM M and an input x, the value $1 - M(x)$.

  However, as we have seen, the complement of an $\mathrm{NTIME}(n)$ language is not expected to be in $\mathrm{NTIME}(n^{1.5})$.

Exploiting Exponential Time

- On the other hand, the complement of every NTIME(n) language is trivially decidable in exponential time.

  This is achieved by examining all the possibilities for the machine's nondeterministic choices.

  This trivial exponential simulation of a nondeterministic machine does suffice to establish a hierarchy theorem.

Lazy Diagonalization

- The key idea is the so-called **lazy diagonalization**.

  The machine D flips the answer of each linear time NDTM M_i in only one string out of a sufficiently (exponentially) large set of strings.

  Define the function $f : \mathbb{N} \to \mathbb{N}$ by

$$\begin{aligned} f(1) &= 2; \\ f(i+1) &= 2^{f(i)^{1.2}}. \end{aligned}$$

  Given n, we can find in $O\left(n^{1.5}\right)$ time the number i, such that n is sandwiched between $f(i)$ and $f(i+1)$.

  The machine D will try to flip the answer of M_i on some input in the set $\{1^n : f(i) < n \leq f(i+1)\}$.

Lazy Diagonalization (Cont'd)

- The machine D operates as follows.
  - Suppose it receives input x.
  - If $x \notin 1^*$, reject.
  - If $x = 1^n$, then compute i, such that $f(i) < n \leq f(i+1)$.
  1. If $f(i) < n < f(i+1)$, then simulate M_i on input 1^{n+1} non-deterministically in $n^{1.1}$ time and output its answer. If M_i has not halted in this time, then halt and accept.
  2. If $n = f(i+1)$, accept 1^n iff M_i rejects $1^{f(i)+1}$ in $(f(i)+1)^{1.1}$ time.

  Part 2 requires going through all possible $2^{(f(i)+1)^{1.1}}$ branches of M_i on input $1^{f(i)+1}$.

  That is fine, since the input size $f(i+1)$ is $2^{f(i)^{1.2}}$.

  Hence, the NDTM D runs in $O\left(n^{1.5}\right)$ time.

Correctness

- Let L be the language decided by D.

  Claim: $L \notin \mathsf{NTIME}(n)$.

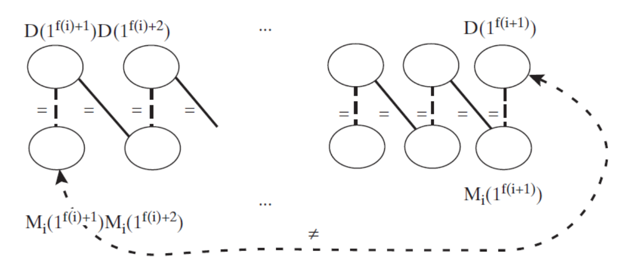
  Indeed, suppose that L is decided by an NDTM M running in cn steps, for some constant c.

  Since each NDTM is represented by infinitely many strings, we can find i large enough, such that:
  - $M = M_i$;
  - On inputs of length $n \geq f(i)$, M_i can be simulated in less than $n^{1.1}$ steps.

  This means that the two steps in the description of D ensure that:
  - If $f(i) < n < f(i+1)$, then $D(1^n) = M_i(1^{n+1})$;
  - $D(1^{f(i+1)}) \neq M_i(1^{f(i)+1})$.

Finishing Correctness

- We obtained the conditions
  - If $f(i) < n < f(i+1)$, then $D(1^n) = M_i(1^{n+1})$;
  - $D(1^{f(i+1)}) \neq M_i(1^{f(i)+1})$.



By our assumption, M_i and D agree on all inputs 1^n for n in the semi-open interval $(f(i), f(i+1)]$.

By the first statement above, $D(1^{f(i+1)}) = M_i(1^{f(i)+1})$.

This contradicts the second statement.

Subsection 3

Ladner's Theorem: Existence of NP-Intermediate Problems

Introducing Ladner's Theorem

- Is every problem in NP either in P or NP-complete?
  - If $\mathrm{P} = \mathrm{NP}$, then the conjecture is trivially true.
  - We show that if $\mathrm{P} \neq \mathrm{NP}$, then this conjecture is false
    I.e., there is a language $L \in \mathrm{NP} \backslash \mathrm{P}$ that is not NP-complete.
- A feature of the proof is an interesting definition à la Gödel of a language SAT_H which "encodes" the difficulty of solving itself.

Ladner's Theorem

Ladner's Theorem ("NP-Intermediate" Languages)

Suppose that $\mathsf{P} \neq \mathsf{NP}$. Then there exists a language

$$L \in \mathsf{NP} \backslash \mathsf{P}$$

that is not NP-complete.

- Consider a function $H : \mathbb{N} \to \mathbb{N}$.

  Define the language $\textsc{Sat}_H$ to contain all length-n satisfiable formulae that are padded with $n^{H(n)}$ 1's:

$$\textsc{Sat}_H = \{\psi 0 1^{n^{H(n)}} : \psi \in \textsc{Sat} \text{ and } n = |\psi|\}.$$

The Function H

- Based on $\textsc{Sat}_H$, define a function $H : \mathbb{N} \to \mathbb{N}$.

  $H(n)$ is the smallest number $i < \log\log n$, such that, for every $x \in \{0,1\}^*$, with $|x| \le \log n$,

  $$M_i \text{ outputs } \textsc{Sat}_H(x) \text{ within } i|x|^i \text{ steps.}$$

  If no such number i exists, then $H(n) = \log\log n$.

  $H(n)$ determines membership in $\textsc{Sat}_H$ of strings whose length is greater than n.

  Moreover, the definition of $H(n)$ only relies upon checking the status of strings of length at most $\log n$.

  So H is well defined.

Complexity of H

Claim

The function H is computable in time $O\left(n^3\right)$.

- To compute $H(n)$ we must:
  (1) Compute $H(i)$ on every $i \leq \log n$;
  (2) Simulate at most $\log\log n$ machines on inputs of lengths at most $\log n$ for less than $\log\log n(\log n)^{\log\log n} = o\left(n\right)$ steps;
  (3) Compute $\textsc{Sat}$ on inputs of size at most $\log n$.

  $H(n)$ can be computed in time $T(n) \leq \log n T(\log n) + O\left(n^2\right)$.

  Therefore, $T(n) = O\left(n^3\right)$.

Relations Between SAT_H and H

Claim

$\text{SAT}_H \in \mathsf{P}$ if and only if $H(n) = O(1)$. If $\text{SAT}_H \notin \mathsf{P}$, then

$$H(n) \stackrel{n\to\infty}{\longrightarrow} \infty.$$

- We first show that, if $\text{SAT}_H \in \mathsf{P}$, then $H(n) = O(1)$.

  Suppose there is a machine M solving SAT_H in at most cn^c steps.

  Recall that M is represented by infinitely many strings.

  So there is a number $i > c$, such that $M = M_i$.

  The definition of $H(n)$ implies that, for $n > 2^{2^i}$, $H(n) \leq i$.

  Thus, $H(n) = O(1)$.

Relations Between SAT_H and H (Cont'd)

- We now show that, if $H(n) = O(1)$, then $\mathrm{SAT}_H \in \mathrm{P}$.

  If $H(n) = O(1)$, then H can take only one of finitely many values.

  Hence, there exists an i, such that $H(n) = i$, for infinitely many n's.

  This implies that the TM M_i solves SAT_H in in^i-time.

  Otherwise, by definition, there is an input x on which M_i fails to output the right answer within this bound.

  Then, for all $n > 2^{|x|}$, we have $H(n) \neq i$.

  Note that the last statement holds even if we only assume that there is some constant C, such that $H(n) \leq C$, for infinitely many n's.

  This proves the second statement in the claim.

NP-Completeness of $\textsc{Sat}_H$ Implies Polynomiality of $\textsc{Sat}$

Claim

Suppose $H : \mathbb{N} \to \mathbb{N}$ is a polynomially computable function, such that

$$\lim_{n\to\infty} H(n) = \infty.$$

If $\textsc{Sat}_H$ is NP-complete, then SAT is in P.

- Let f be a reduction from $\textsc{Sat}$ to $\textsc{Sat}_H$ that runs in time $O\left(n^i\right)$.

  Let N be such that $H(n) > i$, for $n > N$.

  The following recursive algorithm A solves SAT in polynomial time.
  - Suppose A receives as input formula φ.
  - If $|\varphi| \leq N$, then compute the output using brute force.
  - Otherwise compute $x = f(\varphi)$.
    - If x is not of the form $\psi 0 1^{n^{H(|\psi|)}}$, then output FALSE.
    - Otherwise, output $A(\psi)$.

Proof of Ladner's Theorem

Ladner's Theorem ("NP-Intermediate" Languages)

If $\mathsf{P} \neq \mathsf{NP}$, there exists a language

$$L \in \mathsf{NP} \backslash \mathsf{P}$$

that is not NP-complete.

- Suppose that $\textsc{Sat}_H \in \mathsf{P}$.

  By the claim, $H(n) \leq C$, for a constant C.

  Thus, $\textsc{Sat}_H$ is $\textsc{Sat}$ padded with at most n^C 1's.

  But then a polynomial-time algorithm for $\textsc{Sat}_H$ can be used to solve $\textsc{Sat}$ in polynomial time.

  Since $\textsc{Sat}$ is NP-complete, $\mathsf{P} = \mathsf{NP}$.

Proof of Ladner's Theorem (Cont'd)

- Suppose that SAT_H is NP-complete.

  Then, there is a reduction f from SAT to SAT_H that runs in time $\mathrm{O}\left(n^i\right)$ for some constant i.

  Since $\mathrm{SAT}_H \notin \mathrm{P}$, by the claim, $H(n) \to \infty$.

  The reduction works in time $\mathrm{O}\left(n^i\right)$, for large n.

  So it must map SAT instances of size n to SAT_H instances of size smaller than $n^{H(n)}$.

  Thus, for large enough φ, f must map φ to a string $\psi 01^{H(|\psi|)}$, where ψ is smaller by some fixed polynomial factor, say, smaller than $\sqrt[3]{n}$.

  The last claim yields a polynomial-time recursive algorithm A for SAT.

  This contradicts $\mathrm{P} \neq \mathrm{NP}$.

Remarks on Ladner's Theorem

- The theorem shows the existence of some non-NP-complete language in NP\P if NP $\neq$ P.
- The language L seems somewhat artificial.
- Moreover, the proof has not been strengthened to yield a more natural language.
- In fact, the status of most natural languages has been resolved thanks to clever algorithms or reductions.
- Two interesting exceptions are Factoring and Graph Isomorphism.
- For these two languages:
  - No polynomial-time algorithm is currently known;
  - There is strong evidence that they are not NP-complete.

Subsection 4

Oracle Machines and the Limits of Diagonalization

The Essence of Diagonalization

- We would like to qualify the limits of diagonalization.
- Towards this goal, we call "diagonalization" any technique that relies solely upon the following properties of Turing machines:
  - I The existence of an effective representation of Turing machines by strings;
  - II The ability of one TM to simulate any other without much overhead in running time or space.
- Any argument that only uses these facts is treating machines as black boxes, in the sense that the machine's internal workings do not matter.

Oracle Turing Machine and the P vs. NP

- We use a technique involving variants of Turing machines that still satisfy Properties I and II.
- A general way to define such variants of Turing machines leads to oracle Turing machines.
- We do so in two different ways.
  - One way of defining the variant results in TMs for which $\mathrm{P} = \mathrm{NP}$;
  - Another way results in TMs for which $\mathrm{P} \neq \mathrm{NP}$.
- This discrepancy shows that assuming only Properties I and II is not sufficient in order to resolve P versus NP.
- Some additional property must be used.

Oracle Turing Machines Informally

- Oracle machines are TMs that are given access to a black box or "oracle" that can magically solve the decision problem for some language $O \in \{0,1\}^*$.
  - The machine has a special oracle tape on which it can write a string $q \in \{0,1\}^*$.
  - In one step it gets an answer to a query of the form

    "Is q in O?"

  - This can be repeated arbitrarily often with different queries.
- If O is a difficult language, e.g., one that cannot be decided in polynomial time or that is not even decidable, then the oracle gives an added power to the TM.

Oracle Turing Machines

Definition (Oracle Turing Machines)

An **oracle Turing machine** is a TM M that has a special read-write tape, called M's **oracle tape**, and three special states q_{query}, q_{yes}, q_{no}.
To execute M, we specify, in addition to the input, a language $O \in \{0,1\}^*$ that is used as the **oracle** for M.

- Whenever, during the execution, M enters the state q_{query}, with q being the contents of its oracle tape, the machine, then, moves into the state q_{yes}, if $q \in O$, and q_{no}, if $q \notin O$.
- Regardless of the choice of O, a membership query to O counts only as a single computational step.

If M is an oracle machine, $O \in \{0,1\}^*$ a language and $x \in \{0,1\}^*$, then we denote the output of M on input x and with oracle O by $M^O(x)$.
Nondeterministic oracle TMs are defined similarly.

The Classes P^O and NP^O

Definition (The Classes P^O and NP^O)

Let $O \in \{0,1\}^*$.

- P^O is the set containing every language that can be decided by a polynomial-time deterministic TMs with oracle access to O.
- NP^O is the set of all languages that can be decided by a polynomial-time nondeterministic TM with oracle access to O.

- Example: Let $\overline{\textsc{Sat}}$ denote the language of unsatisfiable formulae.

  Then $\overline{\textsc{Sat}} \in \mathsf{P}^{\textsc{Sat}}$.

  Suppose, we are given oracle access to $\textsc{Sat}$.

  To decide whether a formula φ is in $\overline{\textsc{Sat}}$, a polynomial-time oracle TM:
  - Asks its oracle if $\varphi \in \textsc{Sat}$;
  - Then outputs the opposite answer.

Example

- Let $O \in \mathsf{P}$. Then
$$\mathsf{P}^O = \mathsf{P}.$$
Allowing an oracle can only help compute more languages.
So $\mathsf{P} \subseteq \mathsf{P}^O$.
If $O \in \mathsf{P}$, then it is redundant as an oracle.
Let M^O be any polynomial-time oracle TM using O.
We can transform M^O into a standard TM (no oracle) M by simply replacing each oracle call with the computation of O.
Thus, $\mathsf{P}^O \subseteq \mathsf{P}$.

Example

- Let

$$\textsc{ExpCom} = \{\langle M, x, 1^n\rangle : M \text{ outputs 1 on } x \text{ within } 2^n \text{ steps}\}.$$

Then $\mathsf{P}^{\textsc{ExpCom}} = \mathsf{NP}^{\textsc{ExpCom}} = \mathsf{EXP}$.

Clearly, an oracle to $\textsc{ExpCom}$ allows one to perform an exponential-time computation at the cost of one call.

So $\mathsf{EXP} \subseteq \mathsf{P}^{\textsc{ExpCom}}$.

Conversely, let M^O be a nondeterministic polynomial-time oracle TM.

We can simulate its execution with an $\textsc{ExpCom}$ oracle in exp time.

Such time suffices both to:

- Enumerate all of M's nondeterministic choices;
- Answer the $\textsc{ExpCom}$ oracle queries.

Thus, $\mathsf{EXP} \subseteq \mathsf{P}^{\textsc{ExpCom}} \subseteq \mathsf{NP}^{\textsc{ExpCom}} \subseteq \mathsf{EXP}$.

Relativizing Results

- Regardless of what the oracle O is, the set of all TM's with access to O satisfy Properties I and II:
  - We can represent TMs with oracle O as strings;
  - We can use this representation to simulate such TMs using a universal TM, which, itself also, has access to O.
- It follows that any result about TMs or complexity classes that uses only I and II also holds for the set of all TMs with oracle O.
- Such results are called **relativizing results**.
- The Baker, Gill, Solovay Theorem, which follows, implies that neither $\mathsf{P} = \mathsf{NP}$ nor $\mathsf{P} \neq \mathsf{NP}$ can be a relativizing result.

The Baker, Gill, Solovay Theorem

Theorem (Baker, Gill, Solovay)

There exist oracles A, B, such that $\mathsf{P}^A = \mathsf{NP}^A$ and $\mathsf{P}^B \neq \mathsf{NP}^B$.

- First, we take A to be the language ExpCom.

  Then, by the preceding example,

  $$\mathsf{P}^A = \mathsf{NP}^A = \mathsf{EXP}.$$

  We turn to the second construction.

  For any language B, let U_B be the unary language

  $$U_B = \{1^n : \text{some string of length } n \text{ is in } B\}.$$

  For every oracle B, the language U_B is in NP^B.

  A nondeterministic TM can make a nondeterministic guess for the string $x \in \{0,1\}^n$, such that $x \in B$.

The Baker, Gill, Solovay Theorem (Cont'd)

- We showed that, for every oracle B, $U_B \in \mathrm{NP}^B$.

  We now construct an oracle B, such that $U_B \notin \mathrm{P}^B$.

  Combining, we get that $\mathrm{P}^B \neq \mathrm{NP}^B$.

  For every i, we let M_i be the oracle TM represented by the binary expansion of i.

  We construct B in stages.

  Stage i ensures that M_i^B does not decide U_B in $\frac{2^n}{10}$ time.

  Initially, we let B be empty and gradually add strings to it.

  Each subsequent stage determines the status of a finite number of strings (i.e., whether or not these strings will ultimately be in B).

Construction of the Oracle B

- Stage i: So far, we have declared whether or not a finite number of strings are in B.

  Choose n large enough so that it exceeds the length of any such string.

  Run M_i on input 1^n for $\frac{2^n}{10}$ steps.

  - Whenever M_i queries the oracle about strings whose status has been determined, we answer consistently.
  - When M_i queries strings whose status is undetermined, we declare that the string is not in B.

  After letting M_i finish computing on 1^n, we now wish to ensure that the answer of M_i on 1^n (whatever it was) is incorrect.

Construction of the Oracle B (Cont'd)

- Note that, during this computation, we have only decided the fate of at most $\frac{2^n}{10}$ strings in $\{0,1\}^n$.
  All of them were decided to be not in B.
  - Suppose M_i accepts 1^n.
    Then we declare that all remaining strings of length n are also not in B.
    This ensures $1^n \notin U_B$.
  - Suppose M_i rejects 1^n.
    We pick any string x of length n that M_i has not queried.
    Such string exists because M_i made at most $\frac{2^n}{10}$ queries.
    We declare that x is in B.
    This ensures that $1^n \in U_B$.

  In either case, the answer of M_i is incorrect.

The Baker, Gill, Solovay Theorem (Conclusion)

- Every polynomial $p(n)$ is smaller than $\frac{2^n}{10}$, for large enough n.
- Every TM M is represented by infinitely many strings.
- So the construction ensures that M does not decide U_B.
- Thus, $U_B \notin \mathrm{P}^B$.

Remarks on Diagonalization

- Is it possible that diagonalization or some other simulation method might help resolve the P vs. NP?
- If so, it has to use some fact about TMs that does not hold in the presence of oracles (i.e., a nonrelativizing fact).
- Even though many results in complexity relativize, there are some notable exceptions, but we still do not know how to use these nonrelativizing techniques to answer this fundamental question.

Remarks on Oracles

- In some settings in complexity an oracle provided to a TM is not merely a language (i.e., Boolean function) but a general function $f : \{0,1\}^* \to \{0,1\}^*$.
- An oracle TM is a useful abstraction of an algorithm.
- It uses another function as a black-box subroutine, without caring about the specifics of its implementation.