Lecture 1

Review of Basic Computational Complexity

March 30, 2004
Lecturer: Paul Beame
Notes: Daniel Lowd

1.1 Preliminaries

1.1.1 Texts

There is no one textbook that covers everything in this course. Some of the discoveries are simply too recent. For the first portion of the course, however, two supplementary texts may be useful.

- Michael Sipser, *Introduction to the Theory of Computation*.
- Christos Papadimitrion, *Computational Complexity*

1.1.2 Coursework

Required work for this course consists of lecture notes and group assignments. Each student is assigned to take thorough lecture notes no more than twice during the quarter. The student must then type up the notes in expanded form in LaTeX. Lecture notes may also contain additional references or details omitted in class. A LaTeXstylesheet for this will soon be made available on the course webpage. Notes should be typed, submitted, revised, and posted to the course webpage within one week.

Group assignments will be homework sets that may be done cooperatively. In fact, cooperation is encouraged. Credit to collaborators should be given. There will be no tests.

1.1.3 Overview

The first part of the course will predominantly discuss complexity classes above NPrelate randomness, circuits, and counting, to P, NP, and the polynomial time hierarchy. One of the motivating goals is to consider how one might separate P from classes above P. Results covered will range from the 1970's to recent work, including tradeoffs between time and space complexity. In the middle section of the course we will cover the powerful role that interaction has played in our understanding of computational complexity and, in particular, the powerful results on probabilistically checkable proofs (PCP). These ideas were developed in the late 1980's continuing through the 1990's. Finally, we will look at techniques for separating non-uniform complexity classes from P. These have been applied particularly to separate classes inside P from P. The results here are from the 1980's and early 1990's.

1.2 Basic Complexity Classes

For this course we will use the following standard basis for defining complexity classes using multitape (offline) Turing machines.

Definition 1.1. An *(offline) Multitape Turing Machine* is a Turing machine that consists of

- Finite state control,
- A read-only input tape, and
- *Multiple* read-write storage tapes

(The read-only input tape will only be required for the definitions of space complexity.) Whenever we use the terminology Turing machine, or TM, we assume a multitape offline Turing machine unless otherwise specified; similarly for NTM in the nondeterministic case.

The two main parameters for measuring resources used by a computation are
Time = # of steps of the Turing machine, and
Space = largest numbered cell accessed on any storage tape.

Note that this is a small change from the definitions in the Sipser text which defines complexity classes in (a nonstandard way) using single tape Turing machines. Use of multiple tapes is important to distinguish between different subclasses in P. A single tape requires a Turing machine to move back and forth excessively, where a multitape Turing machine might be able to solve the problem much more efficiently.

Lemma 1.1. *For $T(n) \geq n$, if language A can be decided in time $T(n)$ by a multitape TM then a 1-tape TM can decide A in time $O(T^2(n))$ time.*

The following example shows that this is tight. Consider the language PALINDROME = $\{x \in \{0,1\}^* \mid x = x^R\}$. It is fairly straightforward to see how a multitape Turing machine could decide the language in linear time $O(|x|)$ by copying its input string in reverse onto a second tape and then comparing the two tapes character by character. It can also be shown that a single tape Turing machine must take longer:

Theorem 1.2 (Cobham). PALINDROME *requires time $\Omega(n^2)$ on 1-tape Turing machines.*

Proof. Exercise. □

One can simulate any multitape TM with only 2 tapes with a much smaller slowdown.

Lemma 1.3 (Hennie-Stearns). *For $T(n) \geq n$, if language A can be decided in time $T(n)$ by a multitape TM then a 1-tape TM can decide A in time $O(T(n) \log T(n))$ time.*

The proof of this fact is much more involved and we will look at this proof in a more general context.

1.2.1 Time Complexity

Definition 1.2. For $T : \mathbb{N} \to \mathbb{R}^+$,
$\mathsf{TIME}(T(n)) = \{A \subseteq \{0,1\}^* \mid A$ can be decided by a multitape TM M in $O(T(n))$ time$\}$, and
$\mathsf{NTIME}(T(n)) = \{A \subseteq \{0,1\}^* \mid A$ can be decided by a multitape NTM M in $O(T(n))$ time$\}$.

Definition 1.3. Recall the following derived time complexity classes,

$$\begin{aligned}
\mathsf{P} &= \bigcup_k \mathsf{TIME}(n^k) \\
\mathsf{NP} &= \bigcup_k \mathsf{NTIME}(n^k) \\
\mathsf{EXP} &= \bigcup_k \mathsf{TIME}(2^{n^k}) \\
\mathsf{NEXP} &= \bigcup_k \mathsf{NTIME}(2^{n^k}) \\
\mathsf{E} &= \bigcup_k \mathsf{TIME}(k^n) = \bigcup_k \mathsf{TIME}(2^{kn}) \\
\mathsf{NE} &= \bigcup_k \mathsf{NTIME}(k^n) = \bigcup_k \mathsf{NTIME}(2^{kn}).
\end{aligned}$$

Note that E and NE which we may encounter will be seen to be somewhat different from the others because they are not closed under polynomial-time reductions.

1.2.2 Polynomial-time Reductions

Many-one/Karp/Mapping reductions

Definition 1.4. $A \leq_m^p B$ iff there is a polynomial-time computable f such that $x \in A \Leftrightarrow f(x) \in B$

Turing/Cook/Oracle reductions

Definition 1.5. An *oracle TM* $M^?$ is an ordinary Turing machine augmented with a separate read/write *oracle query tape*, *oracle query state*, and 2 oracle answer states, Y and N. When furnished with an oracle B, whenever $M^?$ enters the oracle query state, and enters state Y or N depending on whether or not the contents of the oracle query tape is in language B. Cost for an oracle query is a single time step. If answers to oracle queries are given by membership in B, then we refer to the oracle TM as M^B.

Definition 1.6. $A \leq_T^p B$ iff there is a polynomial-time oracle TM $M^?$ such that $A = L(M^B)$

In other words, M can decide A in polynomial time given a subroutine for B that costs one time step per call.

1.2.3 NP-completeness

Definition 1.7. L is NP-*complete* iff:

1. $L \in \mathsf{NP}$

2. $\forall A \in \mathsf{NP}, A \leq_m^p L$

Theorem 1.4 (Cook). SAT = $\{\langle\varphi\rangle \mid \varphi$ *is a satisfiable propositional logic formula*$\}$ *is* NP*-complete.*

Definition 1.8. For any complexity class C, define co$\mathsf{C} = \{L \mid \overline{L} \in \mathsf{C}\}$.

For example, the class coNP is the set of all languages whose complements are in NP. The following languages are both coNP-complete:

- UNSAT = $\{\langle\varphi\rangle \mid \phi$ is an unsatisfiable propositional logic formula $\}$
- TAUT = $\{\langle\varphi\rangle \mid \varphi$ is a propositional logic tautology $\}$

Note: UNSAT $\leq_T^P$ SAT since Turing reductions don't distinguish between languages and their complements.

Definition 1.9. Define $\forall^k$ and $\exists^k$ as quantifiers over $\{0,1\}^{\leq k}$ the set of binary strings of length at most k.

Using this notation we have an alternative characterization of NP in terms of polynomial-time verifiers.

$A \in \mathsf{NP} \Leftrightarrow$ there is some $R \in \mathsf{P}$ and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $A = \{x \mid \exists^{p(|x|)} y \cdot (x,y) \in R\}$; or in functional rather than set form there is some polynomial-time computable R such that $A = \{x \mid \exists^{p(|x|)} y \cdot R(x,y)\}$.

$A \in$ coNP $\Leftrightarrow$ there is some $R \in \mathsf{P}$ and polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $A = \{x \mid \forall^{p(|x|)} y \cdot (x,y) \in R\}$.

1.2.4 Space Complexity

Definition 1.10. For $S : \mathbb{N} \to \mathbb{R}^+$, define
SPACE$(S(n)) = \{A \subseteq \{0,1\}^* \mid A$ is decided by a multitape (offline) TM using storage $O(S(n))\}$
NSPACE$(S(n)) = \{A \subseteq \{0,1\}^* \mid A$ is decided by a multitape (offline) NTM using storage $O(S(n))\}$

Definition 1.11.

$$\begin{aligned} \mathsf{PSPACE} &= \bigcup_k \mathsf{SPACE}(n^k) \\ \mathsf{L} &= \mathsf{SPACE}(\log n) \\ \mathsf{NL} &= \mathsf{NSPACE}(\log n) \end{aligned}$$

Theorem 1.5. *For* $S(n) \geq \log n$*,*
(a) [Savitch] NSPACE$(S(n)) \subseteq$ SPACE$(S^2(n))$*,*
(b) [Immerman-Szelepcsenyi] NSPACE$(S(n)) =$ co $-$ NSPACE$(S(n))$.

Savitch's theorem implies that PSPACE could equally well have been defined using nondeterministic space complexity.

Theorem 1.6. *For* $S(n) \geq \log n$*,* NSPACE$(S(n)) \subseteq$ TIME$(2^{O(S(n))})$.

Proof idea. An NSPACE$(S(n))$ computation accepts an input x iff there is a computation path from the starting configuration s to a unique accepting configuration t in the graph of all possible configurations with input x, which there are $2^{O(S(n))}$ nodes. This can be solved in time linear in the size of the graph using BFS or DFS. □

Theorem 1.7. $\mathsf{NTIME}(T(n)) \subseteq \mathsf{SPACE}(T(n))$.

Proof idea. Each computation of the NTM of length $cT(n)$ visits at most $cT(n)+1$ cells on each storage tape. The algorithm successively generates all possible sequences of nondeterministic choices of length $cT(n)$ and exhaustively tries each sequence. □

Corollary 1.8. $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \begin{matrix}\mathsf{NP}\\ \mathsf{coNP}\end{matrix} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$.

Definition 1.12. A language L is PSPACE-*complete* iff

1. $L \in \mathsf{PSPACE}$, and
2. $\forall A \in \mathsf{PSPACE}$, $A \leq_m^p L$.

Definition 1.13. Define
TQBF = $\{\langle\Psi\rangle \mid \exists k, Q_1, Q_2, \ldots, Q_k \in \{\exists, \forall\}$ such that $\Psi = Q_1 x_1 \ldots Q_k x_k \varphi$, where φ is a propositional logic formula in $x_1, x_2, \ldots, x_n$, and Ψ evaluates to true $\}$.

Theorem 1.9. TQBF *is* PSPACE-*complete.*

1.2.5 Complexity Class Hierarchy Theorems

Definition 1.14. $T(n)$ is *time-constructable* iff there is a TM running in time $\leq T(n)$ that computes $1^n \to T(n)$, where $T(n)$ is expressed in binary.
$S(n)$ is *space-constructable* iff there is a TM running in space $\leq S(n)$ that computes the function $1^n \to S(n)$, where $S(n)$ is expressed in binary.

Theorem 1.10. *1. If $g(n) \geq \log(n)$ is space-constructable and $f(n)$ is $o(g(n))$ then* $\mathsf{SPACE}(f(n)) \subsetneq \mathsf{SPACE}(g(n))$, *and* $\mathsf{NSPACE}(f(n)) \subsetneq \mathsf{NSPACE}(g(n))$.

2. If $g(n) \geq n$ is time constructable and $f(n)\log f(n)$ is $o(g(n))$ then $\mathsf{TIME}(f(n)) \subsetneq \mathsf{TIME}(g(n))$.

Proof Sketch. The general idea of all of these theorems is to diagonalize over all TM's with time (space) bounded by $f(n)$. In order to do this one diagonalizes over all Turing machines but with an extra clock (in the case of time) or space limiter (in the case of space) to make sure that the machine involved won't take too many resources.

In the case of the space hierarchy, this diagonalization must be done by a single Turing machine so in order to simulate other Turing machines with larger alphabets there must be a constant-factor slowdown in the simulation. The Immerman-Szelepcsenyi theorem allows the complementation to take place in the case of nondeterministic space.

For the time hierarchy the proof in the multitape case is very different from the single tape case given in Sipser's text despite the fact that the claim is the same. In the single tape case, the $O(\log f(n))$ factor is due to the requirement to update the clock (and shift it along the tape) at each step. In the multitape case, the $O(\log f(n))$ factor slowdown is due the requirement of having a machine with a fixed number of tapes simulate machines with arbitrary numbers of tapes. □

Corollary 1.11. $\mathsf{NL} \subsetneq \mathsf{PSPACE}$; $\mathsf{P} \subsetneq \mathsf{EXP}$.

1.2.6 Relativized Complexity Classes

Definition 1.15. Given a language A, we can define $\mathsf{P}^A = \{B \mid B \leq_T^p A\}$.

That is, to obtain the languages in P^A we can take any polynomial-time oracle TM $M^?$, plug in A as the oracle and look at what language the machine accepts. This yields an alternate version of the definition that can also be extended to nondeterministic Turing machines.

Definition 1.16. For any language A define $\mathsf{P}^A = \{L(M^A) \mid M^?$ is a polynomial-time oracle TM $\}$ and $\mathsf{NP}^A = \{L(M^A) \mid M^?$ is a polynomial-time oracle NTM $\}$.
For any complexity class C define $\mathsf{P}^{\mathsf{C}} = \bigcup_{A \in \mathsf{C}} \mathsf{P}^A$, $\mathsf{NP}^{\mathsf{C}} = \bigcup_{A \in \mathsf{C}} \mathsf{NP}^A$,

Remark. Note that $\mathsf{P}^{\mathrm{SAT}} = \mathsf{P}^{\mathsf{NP}}$ by the fact that SAT is NP-complete under polynomial-time Turing reductions.