Harvard CS 121 and CSCI E-207
Lecture 19: Computational Complexity

Salil Vadhan

November 8, 2012

• Reading: Sipser §7.1
Objective of Complexity Theory

• To move from a focus:
  • on what it is possible in principle to compute
  • to what is feasible to compute given “reasonable” resources

• For us the principle “resource” is time, though it could also be memory (“space”) or hardware (switches)
What is the “speed” of an algorithm?

- **Def:** A TM $M$ has running time $t : N \to N$ iff for all $n$, $t(n)$ is the maximum number of steps taken by $M$ over all inputs of length $n$.

  → implies that $M$ halts on every input
  → in particular, every decision procedure has a running time
  → time used as a function of size $n$
  → worst-case analysis
Example running times

• Running times are generally increasing functions of $n$

\[ t(n) = 4n. \]

\[ t(n) = 2n \cdot \lceil \log n \rceil \]

\[ \lfloor x \rfloor = \text{least integer } \geq x \text{ (running times must be integers)} \]

\[ t(n) = 17n^2 + 33. \]

\[ t(n) = 2^n + n. \]

\[ t(n) = 2^{2^n}. \]
“Table lookup” provides speedup for finitely many inputs

**Claim:** For every decidable language $L$ and every constant $k$, there is a TM $M$ that decides $L$ with running time satisfying $t(n) = n$ for all $n \leq k$.

**Proof:**

⇒ study behavior only of Turing machines $M$ deciding infinite languages, and only by analyzing the running time $t(n)$ as $n \rightarrow \infty$. 
Why bother measuring TM time, when TMs are so miserably inefficient?

• **Answer:** Within limits, multitape TMs are a reasonable model for measuring computational speed.

• The trick is to specify the right amount of “slop” when stating that two algorithms are “roughly equivalent”.

• Even coarse distinctions can be very informative.
Complexity Classes

• Def: Let $t : \mathbb{N} \rightarrow \mathbb{R}^+$. Then $\text{TIME}(t)$ is the class of languages $L$ that can be decided by some multitape TM with running time $\leq t(n)$.

  e.g. $\text{TIME}(10^{10} \cdot n)$, $\text{TIME}(n \cdot 2^n)$

  $\mathbb{R}^+ = \text{positive real numbers}$

• Q: Is it true that with more time you can solve more problems?

  i.e., if $g(n) < f(n)$ for all $n$, is $\text{TIME}(g) \subsetneq \text{TIME}(f)$?

• A: Not exactly . . .
Linear Speedup Theorem

Let $t : \mathbb{N} \rightarrow \mathbb{R}^+$ be any function s.t. $t(n) \geq n$ and $0 < \varepsilon < 1,$
Then for every $L \in \text{TIME}(t),$ we also have
$L \in \text{TIME}(\varepsilon \cdot t(n) + n)$

• $n =$ time to read input

• Note implied quantification:

$$(\forall \text{ TM } M)(\forall \varepsilon > 0)(\exists \text{ TM } M') M'$ is equivalent to $M$ but runs in fraction $\varepsilon$ of the time.

• “Given any TM we can make it run, say, 1,000,000 times faster on all inputs.”
Proof of Linear Speedup

• Let $M$ be a TM deciding $L$ in time $T$.

• A new, faster machine $M'$:

(1) Copies its input to a second tape, in compressed form.

```
  a b c b a a b c b □
```

⇒

```
  abc baa bc □ □ □ □ □
```

• (Compression factor $= 3$ in this example—actual value TBD at end of proof)

(2) Moves head to beginning of compressed input.

(3) Simulates the operation of $M$ treating all tapes as compressed versions of $M'$’s tapes.
Analysis of linear speedup

• Let the “compression factor” be $c$ ($c = 3$ here), and let $n$ be the length of the input.

• Running time of $M'$:

  (1) $n$ steps

  (2) $\lceil n/c \rceil$ steps.

    \cdot $\lceil x \rceil = \text{smallest integer} \geq x$

  (3) takes ?? steps.
How long does the simulation (3) take?

• $M'$ remembers in its finite control which of the $c$ “subcells” $M$ is scanning.

• $M'$ keeps simulating $c$ steps of $M$ by 8 steps of $M'$:
  
  1. Look at current cell on either side.
     
     (4 steps to read $3c$ symbols)
  
  2. Figure out the next $c$ steps of $M$.
     
     (can’t depend on anything outside these $3c$ subcells)
  
  3. Update these 3 cells and reposition the head.
     
     (4 steps)
End of simulation analysis

- It must do this $\lceil t(n)/c \rceil$ times, for a total of $8 \cdot \lceil t(n)/c \rceil$ steps.

- Total of $\leq (10/c) \cdot t(n) + n$ steps of $M'$ for sufficiently large $n$.

- If $c$ is chosen so that $c \geq 10/\varepsilon$ then $M'$ runs in time $\varepsilon \cdot t(n) + n$. 
Implications/Rationalizations of Linear Speedup

• “Throwing hardware at a problem” can speed up any algorithm by any desired constant factor

• E.g. moving from 8 bit → 16 bit → 32 bit → 64 bit parallelism

• Our theory does not “charge” for huge capital expenditures to build big machines, since they can be used for infinitely many problems of unbounded size

• This complexity theory is too weak to be sensitive to multiplicative constants — so we study growth rate
Growth Rates of Functions

We need a way to compare functions according to how fast they increase not just how large their values are.

**Def:** For $f : \mathbb{N} \rightarrow \mathbb{R}^+$, $g : \mathbb{N} \rightarrow \mathbb{R}^+$, we write $g = O(f)$ if there exist $c, n_0 \in \mathbb{N}$ such that $g(n) \leq c \cdot f(n)$ for all $n \geq n_0$.

- Binary relation: we could write $g = O(f)$ as $g \preceq f$.
- “If $f$ is scaled up uniformly, it will be above $g$ at all but finitely many points.”
- “$g$ grows no faster than $f$.”
- Also write $f = \Omega(g)$. 
Examples of Big-$O$ notation

• If $f(n) = n^2$ and $g(n) = 10^{10} \cdot n$

  $g = O(f)$ since $g(n) \leq 10^{10} \cdot f(n)$ for all $n \geq 0$

  where $c = 10^{10}$ and $n_0 = 0$

• Usually we would write: “$10^{10} \cdot n = O(n^2)$”

  i.e. use an expression to name a function

• By Linear Speedup Theorem, TIME($t$) is the class of languages $L$ that can be decided by some multitape TM with running time $O(t(n))$ (provided $t(n) \geq 1.01n$).
Examples

• $10^{10} \cdot n = O(n^2)$.

• $1764 = O(1)$.

  1: The constant function $1(n) = 1$ for all $n$.

• $n^3 \neq O(n^2)$.

• Time $O(n^k)$ for fixed $k$ is considered “fast” (“polynomial time”)

• Time $\Omega(k^n)$ is considered “slow” (“exponential time”)

• Does this really make sense?
More Relations

**Def:** We say that $g = o(f)$ iff for every $\varepsilon > 0$, $\exists n_0$ such that $g(n) \leq \varepsilon \cdot f(n)$ for all $n \geq n_0$.

- Equivalently, $\lim_{n \to \infty} g(n)/f(n) = 0$.
- “$g$ grows more slowly than $f$.”
- Also write $f = \omega(g)$.

**Def:** We say that $f = \Theta(g)$ iff $f = O(g)$ and $g = O(f)$.

- “$g$ grows at the same rate as $f$”
- An equivalence relation between functions.
- The equivalence classes are called growth rates.

**Note:** If $\lim_{n \to \infty} g(n)/f(n) = c$ for some $0 < c < \infty$, then $f = \Theta(g)$, but the converse is not true. (Why?)
More Examples

Polynomials (of degree \( d \)):

\[ f(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0, \text{ where } a_d > 0. \]

- \( f(n) = O(n^c) \) for \( c \geq d \).
- \( f(n) = \Theta(n^d) \)
  
  - “If \( f \) is a polynomial, then lower order terms don’t matter to the growth rate of \( f \)”
- \( f(n) = o(n^c) \) for \( c > d \).
- \( f(n) = n^{O(1)}. \) (This means: \( f(n) = n^{g(n)} \) for some function \( g(n) \) such that \( g(n) = O(1) \).)
More Examples

Exponential Functions: \( g(n) = 2^n^{\Theta(1)} \).

- Then \( f = o(g) \) for any polynomial \( f \).
- \( 2^{n^\alpha} = o(2^{n^\beta}) \) if \( \alpha < \beta \).

What about \( n^{\lg n} = 2^{\lg^2 n} \)?

Here \( \lg x = \log_2 x \)

Logarithmic Functions:

\[ \log_a x = \Theta(\log_b x) \text{ for any } a, b > 1 \]
Asymptotic Notation within Expressions

When we use asymptotic notation within an expression, the asymptotic notation is shorthand for an unspecified function satisfying the relation.

- \( n^{O(1)} \)

- \( n^2 + \Omega(n) \) means \( n/2 + g(n) \) for some function \( g(n) \) such that \( g(n) = \Omega(n) \).

- \( 2^{(1-o(1))n} \) means \( 2^{(1-\epsilon(n))} \cdot g(n) \) for some function \( \epsilon(n) \) such that \( \epsilon(n) \to 0 \) as \( n \to \infty \).
Asymptotic Notation on Both Sides

When we use asymptotic notation on both sides of an equation, it means that for all choices of the unspecified functions in the left-hand side, we get a valid asymptotic relation.

- \( n^2/2 + O(n) = \Omega(n^2) \) because for every function \( f \) such that \( f(n) = O(n) \), we have \( n^2/2 + f(n) = \Omega(n^2) \).

- But it is not true that \( \Omega(n^2) = n^2/2 + O(n) \).