Harvard CS 121 and CSCI E-207

Lecture 19: Polynomial Time

Harry Lewis

November 16, 2010

Linear Speedup Theorem

Let $t : \mathcal{N} \to \mathcal{R}^+$ 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:

  $(\forall$ TM $M)(\forall \varepsilon > 0)(\exists$ TM $M')$ M' is equivalent to M but runs in fraction ε 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	$\sqcup$	

$\Rightarrow$

abc	baa	bcb	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	

- (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.

 · $\lceil x \rceil$ = 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 $\rightarrow$ 16 bit $\rightarrow$ 32 bit $\rightarrow$ 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 <u>growth rate</u>

Time-bounded Simulations

Q: How quickly can a 1-tape TM M_2 simulate a multitape TM M_1?

- If M_1 uses $f(n)$ time, then it uses $\leq f(n)$ tape cells
- M_2 simulates one step of M_1 by a complete sweep of its tape. This takes $\mathcal{O}(f(n))$ steps.

$\therefore$ M_2 uses $\leq f(n) \cdot \mathcal{O}(f(n)) = \mathcal{O}(f^2(n))$ steps in all.

So $L \in \text{TIME}_{\text{multitape TM}}(f) \Rightarrow L \in \text{TIME}_{\text{1-tape TM}}(\mathcal{O}(f^2))$

Similarly for

- 2-D Tapes
- Random Access TMs . . .

Basic thesis of complexity theory

Extended Church-Turing Thesis: Every “reasonable” model of computation can be simulated on a Turing machine with only a polynomial slowdown.

Counterexamples?

- Randomized computation.
- Parallel computation.
- Analog computers.
- DNA computers.
- Quantum computers.

Polynomial Time

- **Def:** Let P = $\bigcup_{p}$ TIME(p), where p is a polynomial

  = $\bigcup_{k \geq 0}$ TIME(n^k)

- P is also known as PTIME or $\mathcal{P}$

- Coarse approximation to "efficient algorithm"

Model-Independence of P

Although P is defined in terms of TM time, **P is a stable class, independent of the computational model.**
(Provided the model is reasonable.)

Justification:

- If A and B are different models of computation, $L \in \text{TIME}_A(p_1(n))$, and B can simulate a time t computation of A in time $p_2(t)$, then $L \in \text{TIME}_B(p_2(p_1(n)))$.

- Polynomials are closed under composition, e.g. $f(n) = n^2, g(n) = n^3 + 1 \Rightarrow f(g(n)) = (n^3 + 1)^2 = n^6 + 2n^3 + 1$.

How much does representation matter?

- How big is the representation of an n-node directed graph?
  - . . . as a list of edges?
  - . . . as an adjacency matrix?
- How big is the representation of a natural number n?
  - . . . in binary?
  - . . . in decimal?
  - . . . in unary?

For which of the following do we know polynomial-time algorithms?

- Given a DFA M and a string w, decide whether M accepts w.
  - What is the "size" of a DFA?

- Given an NFA N, construct an equivalent DFA M.

More computational problems: are they in P?

- Given an NFA N and a string w, decide whether N accepts w.

- Given a regular expression R, construct an equivalent NFA N.

- Given a CFG G and a string w, decide whether G generates w.

And more computational problems: are they in P?

- Given two numbers n, m, compute their product.
  - What is the "size" of the numbers?

- Given a number n, decide if n is prime.

- Given a number n, compute n's prime factorization.

Another way of looking at P

- Multiplicative increases in time or computing power yield multiplicative increases in the size of problems that can be solved

- If L is in P, then there is a constant factor k such that
  - If you can solve problems of size s within a given amount of time
  - and you are given a computer that runs twice as fast, then
  - you can solve problems of size $k \cdot s$ on the new machine in the same amount of time.

- E.g. if L is decidable in $O(n^d)$ time, then with twice as much time you can solve problems $2^{\frac{1}{d}}$ as large

Exponential time

- E $= \cup_{c>0}$TIME(c^n)

- For problems in E, a multiplicative increase in computing power yields only an *additive* increase in the size of problems that can be solved.

- If L is in E, then there is a constant k such that
  - If you can solve problems of size s within a given amount of time
  - and you are given a computer that runs twice as fast, then
  - you can solve problems only of size $k + s$ on the new machine using the same amount of time.