CSE547T Class 18

Jeremy Buhler

March 29, 2017

1 Polynomial Time

- There are many fine distinctions that can be made among running times.
- Language classes DTIME($f(n)$) and DTIME($g(n)$) are not the same unless $f(n) = \Theta(g(n))$.
- Moreover, "small" changes to TMs, such as allowing a second tape, can change the class of a language (as we saw for 0^n1^n).
- We'd like a notion of "efficient" vs "inefficient" that is robust to small changes in our computational model.
- *Idea*: running times that are $O(n^k)$ for constant k are preferable to running times that are $\Omega(c^n)$.
- Indeed, n^k is $o(c^n)$, no matter how big we make k or how small we make $c > 1$. Polynomials always grow more slowly than exponentials.
- Let's say that polynomial running times are "efficient."
- **Defn**: the class P consists of all languages that are in DTIME(n^k) for any $k \geq 0$. That is,

$$P = \bigcup_{k \geq 0} \text{DTIME}(n^k).$$

  Hence, P is the set of all languages that can be decided efficiently.

Is P a useful classification for languages, independent of TMs?

- Say that two models of computation Ξ_1 and Ξ_2 are *polynomially equivalent* if a language can be decided in polynomial time in Ξ_1 iff it it can be decided in polynomial time in Ξ_2.
- (The two polynomials may not be the same.)
- For example, deterministic, k-tape TMs are polynomially equivalent to deterministic, single-tape TMs.
- Let DTIME$_\Xi(f(n))$ be the class of languages decidable in time $f(n)$ in model Ξ, and let P_Ξ be the equivalent definition of class P for model Ξ.

- Then for any two polynomially equivalent models Ξ_1 and Ξ_2, we have that

$$P_{\Xi_1} = P_{\Xi_2}.$$

- Hence, the class P is invariant to a choice among polynomially equivalent models of computation.

What other models of computation are polynomially equivalent to deterministic TMs?

- Every "reasonable" model of computation we know of is polynomially equivalent to deterministic TMs!
- "Reasonable" means, approximately, physically realizable. It therefore excludes infinite-precision analog or digital computation.
- This observation is called the *extended Church-Turing thesis*.
- Hence, reasoning about what can be done efficiently on a deterministic TM is informative for real-world computers too.
- (But please don't ask me about quantum computers!)

2 Decision vs Optimization

While we are discussing polynomial equivalence, let's take a brief detour and think about a common task in CS – *optimization*.

- A generic optimization problem Q takes a problem instance and finds the *best feasible solution*.
- "Best" means (say) maximal according to some *objective function* that maps feasible solutions to *natural numbers*.
- The result $Q(x)$ is the objective value of an optimal feasible solution for an input x.
- Define $\text{DEC}_Q(x, y)$ to be the decision problem "does x have a feasible solution with objective value at least y?"
- It's easy to see that, if we can compute $Q(x)$ in time polynomial in $|x|$, then we can answer $\text{DEC}_Q(x, y)$ in time polynomial in $|x|$.
- What about the other direction?
- **Lemma**: Let $Q(x) = y^*$, and suppose we have a TM M to decide $\text{DEC}_Q(x, y)$ for any x and y. Then we can compute $Q(x)$ using only $O(\log y^*)$ calls to M.
- **Pf**: computing $Q(x)$ is equivalent to finding the largest y^* for which $\text{DEC}_Q(x, y^*)$ is true.
- Algorithm is as follows:
  - Feed M inputs $(x, 2^k)$ for $k = 0, 1, 2, \ldots$ until we find the least j such that $\text{DEC}_Q(x, 2^j) = \text{false}$.

- If $j = 0$, then either $y^* = 0$, or no feasible soln exists. Call M on $(x, 0)$ to find out.
- Otherwise, $2^{j-1} \leq y^* < 2^j$. Use binary search on this interval, calling M each time, to find y^*.

- Total number of calls to M is $O(\log 2^j) = O(\log y^*)$, since 2^j is an upper bound on y^*. QED

So what?

- **Corollary**: if
  1. $\text{DEC}_Q(x, y)$ can be decided in time polynomial in $|x|$ and $|y|$, and
  2. for all x, $Q(x) = O(2^{\text{poly}(|x|)})$,

  then $Q(x)$ can be computed in time polynomial in $|x|$.
- Condition 2 requires that the optimum objective value $Q(x)$ is at most exponential in size of x.
- Equivalently, it takes space at most polynomial in $|x|$ to write down $Q(x)$.
- **Pf**: Suppose M runs on (x, y) in time $O(|x|^a |y|^b)$.
- For any input x, we know that optimal objective value y^* is $O(2^{\text{poly}(|x|)})$. Largest y passed to M is within constant factor 2 of y^*.
- Hence, above algo to compute $Q(x)$ using M takes time

$$\begin{aligned} O(|x|^a |y^*|^b \log y^*) &= O(|x|^a \left|2^{\text{poly}(|x|)}\right|^b \log 2^{\text{poly}(|x|)}) \\ &= O(|x|^a \, \text{poly}(|x|)^b \text{poly}(|x|)) \\ &= O(\text{poly}(|x|)), \end{aligned}$$

  where the second step follows because $|w| = \Theta(\log w)$.
- Conclude that $Q(x)$ can be computed from x in time $O(\text{poly}(|x|))$. QED

Hence, an optimization problem is polytime solvable iff its "natural" decision problem is in P.

3 What About Nondeterminism?

- Last time, we saw that the obvious simulation of a nondeterministic TM by a deterministic TM required an exponential blowup in running time.
- Let's think about nondeterministic machines for a while.
- Let $\text{NTIME}(t(n))$ be the class of languages that can be decided in time $O(t(n))$ by a *nondeterministic* TM.

- Analogously to P, we can define the class

$$NP = \bigcup_{k \geq 0} \text{NTIME}(n^k).$$

Nondeterministic TMs are kind of wacky. Can we characterize class NP more intuitively?

- **Defn**: Let L be a language. A *certificate* c_w for some $w \in L$ is a string that "proves" that w is in L.
- Exactly what a certificate looks like depends on L.
- For many combinatorial problems like SSS, think of c as being "an answer", e.g. a list of which elements of X to use to make the target t.
- **Defn**: Let L be a language. A *verifier* for L is a TM that accepts the language $\langle w, c_w \rangle$, where $w \in L$ and c_w is a certificate for w.
- A verifier essentially checks the proof provided by c_w to ensure that $w \in L$.
- **Defn**: a language L is said to be *polynomially verifiable* if there exists a certificate c_w for each $w \in L$ and a verifier V that accepts $\langle w, c_w \rangle$ in time polynomial in $|w|$.
- (Note that c_w must have size polynomial in w, or V can't read it all in its allotted running time.)
- For example, a verifier for SSS using the certificates described above would take the instance X, t *and* a list of which elements of X form a solution, then add up the chosen elements and accept iff they add to t.

What does verifiability have to do with NP ?

- **Thm**: for any language L, $L \in NP$ iff L is polytime verifiable.
- **Pf**: ($\rightarrow$) Suppose $L \in NP$. Then L is decided by some nondeterministic TM N in polynomial time $O(t(n))$.
- For any $w \in L$, let its certificate c_w be a list of nondeterminitic choices made by N that yields an accepting computation for w.
- If N can choose among $O(1)$ moves at each step, then this list is a string of size $O(t(n))$ bits.
- We now build a verifier V for L as follows.
- On input $\langle w, c_w \rangle$, V simulates execution of N on w. At N's jth step, V refers to the jth element of c_w to determine which move to make.
- If $w \in L$, there exists a cert c_w that will make V accept. Otherwise, no such cert exists.
- Moreover, V runs in time $O(t(n))$, since N must accept or reject w within this time.
- ($\leftarrow$) Suppose conversely that L is polytime verifiable by some V.

- V uses certificates c_w of size polynomial in $|w|$.
- We can build a nondeterministic TM N to accept L as follows.
- On input w, N nondeterministically guesses a certificate c_w, then simulates V on input $\langle w, c_w \rangle$.
- If $w \in L$, a certificate c_w exists, and one of N's guesses will lead to acceptance. Otherwise, all guesses will reject.
- N needs time proportional to the max cert length for w to guess a certificate c_w, plus time equal to V's running time on $\langle w, c_w \rangle$ to simulate V. Both these times are polynomial in $|w|$.

4 The Big Question

- We now have two complexity classes: P and NP . Are they the same or not?
- To show what it is at stake here, let's think not purely in terms of TMs and languages but about combinatorial problems.
- A language L corresponds to a membership problem: is $w \in L$ or not?
- If a problem is in P, an instance w of this problem can be solved in time polynomial in the size of w.
- If a problem is in NP , a proof of the answer to an instance of this problem can be *checked* in time polynomial in the size of w.
- Which do you think is harder – solving a problem or checking that the answer is correct, given a proof?

With this in mind, let's consider what we actually know.

- **Thm**: $P \subseteq NP$.
- **Pf**: suppose a language L is in P. Then there exists a deterministic, polytime TM that decides L.
- But every such TM is also (trivially) a nondeterministic TM, so $L \in NP$.
- What about the opposite inclusion?
- Our intuition says "probably not!"
- First, solving a problem in general seems harder than checking a given solution.
- Second, it's hard to imagine a deterministic simulation of a nondeterministic TM that runs in time polynomial in its input size.
- Third, we can think of many problems that are in NP that seem really hard to solve.
- **Examples**: CLIQUE, HAMCYCLE, ...

- (Exercise: what are certificates and verifiers for such problems?)
- However, nobody has actually proved whether or not $NP \subseteq P$.
- In other words, **the question "Is $P = NP$?" is open**.
- It's been open for well over 40 years.