CS 40: Computational Complexity

Sair Shaikh

October 6, 2025

Collaboration Notice: Talked to Henry Scheible '26 to discuss ideas.

Problem 1. Imagine it is 1980, so we don't yet know Immerman and Szelepcsényi's amazing technique for relating NSPACE($f(n)$) to coNSPACE($f(n)$). Therefore, we don't know how to do the direct diagonalization argument for proving the nondeterministic space hierarchy theorem.
However, we do know Savitch's Theorem. Prove that this, plus a padding argument, plus the deterministic space hierarchy theorem, together imply a somewhat weak nondeterministic space hierarchy theorem. Specifically, give a proof that if α and β are real-valued constants with $0 < \alpha < \beta$, then

$$\text{NSPACE}(n^\alpha) \neq \text{NSPACE}(n^\beta).$$

Hint: First, as an example, figure out how to show that NSPACE(n^5) $\neq$ NSPACE(n^{11}). This should not require any padding. Now ask yourself: how would padding help reduce the gap between 5 and 11 in this example?

Solution. Assume, for the sake of contradiction, NSPACE(n^α) = NSPACE(n^β) for $\alpha < \beta$.

First, assume that you have $2\alpha < \beta$ (this is the case with $\alpha = 5$ and $\beta = 11$). Using the deterministic space hierarchy theorem, noting that $2\alpha < 2\beta$, we know that:

$$\text{DSPACE}(n^{2\alpha}) \subsetneq \text{DSPACE}(n^\beta)$$

Thus, pick $L \in \text{DSPACE}(n^\beta) \setminus \text{DSPACE}(n^{2\alpha})$. Since determinism is, in particular, a vacuous case of non-determinism, we note that $L \in \text{NSPACE}(n^\beta)$. By our assumption, then, $L \in \text{NSPACE}(n^\alpha)$. By Savitch's Theorem, we know that:

$$\text{NSPACE}(n^\alpha) \subseteq \text{DSPACE}(n^{2\alpha})$$

Thus, $L \in \text{DSPACE}(n^{2\alpha})$ which contradicts our assumption.

Thus, the only case left to handle is when $2\alpha > \beta$. We will show that we can use a padding argument to produce a pair (α', β') that satisfies the first case.

Padding an input by any polynomial amount can be done in logarithmic space, since we only have to maintain counters up to a fixed maximum. Thus, padding an input x to a length of $k|x|$ reduces the space-complexity required appropriately (by simulating the turning machine for the pre-padded language) from $O(n^i)$ to $O(n^i/k)$. This implies we can substract the same thing from the exponents of our assumption to get:

$$\text{NSPACE}(n^{\alpha-\gamma}) = \text{NSPACE}(n^{\beta-\gamma})$$

until $2(\alpha - \gamma) < \beta - \gamma$, in which case we obtain a contradiction using the first case.

Problem 2. Prove that DSPACE(n) $\neq$ NP. That said, it remains open whether one of these two classes is contained in the other!
Hint: Remember that this problem set is associated with the lecture on hierarchy theorems.

Solution. Assume for the sake of contradiction that DSPACE(n) = NP.
From the deterministic Space Hierarchy Theorem, we know that:

$$\text{DSPACE}(n) \subsetneq \text{DSPACE}(n^2)$$

Thus, pick a language $L \in \text{DSPACE}(n^2) \setminus \text{DSPACE}(n)$ and let M be the deterministic turing machine that decides L in these bounds. Using padding we can construct a language L' as follows:

$$L' = \{x\perp^{|x|^2-|x|} : x \in L\}$$

where $\perp$ is not in the alphabet for L. Then, we claim that $L' \in \text{DSPACE}(n)$. Indeed, we can modify M to construct a machine M' that does the following:

1. First, checks that the input length is a square number. REJECT if not.
   We can do this in $O(\log(n))$ space. First, we maintain a counter to count up the input length ($\log(n)$ sized). Then, we iterate over integers starting at 1, square them (multiplication can be done in log-space via last problem set), and compare if they're less than, equal to, or greater than the input length. If less than, we continue this loop; if greater than, we REJECT (i.e. we have checked all integers upto $\sqrt{n}$ and none of them are equal to $\sqrt{n}$); if equal to, we proceed to the next step. Overall, this takes $O(\log(n))$ space.
2. Then, it simulates M on the first $\sqrt{n}$ characters of the input. ACCEPT or REJECT based on Ms output. This takes $O(\sqrt{n}^2) = O(n)$ space, plus an additional log-space counter to make sure we do not go over $\sqrt{n}$ characters. Thus, this can be done in $O(n)$ space.

Thus, overall M' runs in $O(n)$ space, thus, $L' \in \text{DSPACE}(n)$. Then, by our assumption, $L' \in$ NP.

Next, we note that deciding L can be reduced to deciding L' via a polynomial time transducer. Indeed, all the transducer needs to do is pad a given input x until its length is $|x|^2$. This only involves keeping counters to count up to at most $|x|^2$, which can be done in $O(\log(n^2)) = O(\log(n))$ space, thus polynomial time.
However, then, deciding L is only a polynomial factor slower than deciding L, where the polynomial factor is the time-bound for running the transducer. Thus, since $L' \in$ NP, we conclude that $L \in$ NP.

By our assumption, this implies that $L \in \text{DSPACE}(n)$ which is a contradiction. Thus,

$$\text{DSPACE}(n) \neq \text{NP}$$

Problem 3. (This problem is not about hierarchies, but it will teach you a useful lesson about 3SAT.) Prove that there is a constant $\alpha < 1$ such that 3SAT has a deterministic algorithm that runs in time $2^{\alpha r}$, where r is the number of variables in the input 3-CNF formula.
The lesson to be learned is that we can get an asymptotic improvement over the running time of the naïve algorithm that tries out every possible assignment to the r variables.

Proof. The key idea that provides this speed-up over the naive algorithm is that instead of trying all possible assignments, we can use a slightly more greedy approach to look at the clauses and not check assignments that will not work, thus getting asymptotically faster run-time.

In particular, assume you are given the clause $x \vee y \vee z$ where x, y, z are literals. Assume x, y, z correspond to distinct variables. Then, we know there is at least one assignment of the variables corresponding to x, y, z such that the clause is not true, i.e. the assignment picking each variable to be the opposite of what would make the clause true. Thus, we only need to check the remaining 7 assignments for the variables corresponding to x, y, z. Plugging each of these 7 assignments into the formula, we can then leverage recursion to obtain a speed-up.

In detail, the algorithm is as follows:

1. First, scan through the clauses. If any clause contains a repeated literal, delete the duplicated instance. If any clause is unsatisfiable (i.e. contains both a variable and its negation), REJECT. Now we may assume that every clause contains literals that correspond to distinct variables. This takes $O(n)$ time.

2. If the number of variables is at most 3, brute-force check for a valid assignment. ACCEPT if found, else REJECT.

3. Otherwise, scan to the first clause. Assume the clause is $x \vee y \vee z$ (the argument still holds if the clause has 1 or 2 literals). Then, there are at most 7 valid assignments of the variables corresponding to x, y, z. Loop over these assignments. For each assignment, scan through the input string, deleting any clauses that are already satisfied by the current assignment of the (up to) 3 variables and deleting the variables from clauses where they appear otherwise. Now, we have reduced our formula to one with 3 fewer variables. Overall, each of these substitution steps takes $O(n)$ time, since you can over the formula once, thus all of the substitution steps combined take $O(n)$ time since there's at most 7 assignments.

4. Search for a satisfying assignment to this formula recursively. ACCEPT if found, REJECT otherwise.

This algorithm satisfies the recursion:

$$T(n) \leq 7 \cdot T(n-3) + cn$$

where n is the number of variables and c is some positive constant. Unrolling this for k steps, we get:

$$T(n) \leq 7^k \, T(n-3k) \; + \; c \sum_{i=0}^{k-1} 7^i \, (n-3i)$$

Taking this to the base-case, we get (loosely):

$$T(n) \in O(n \cdot 7^{n/3})$$

Next, note that $O(n \cdot 7^{n/3}) \subsetneq O((7.5)^{n/3})$ as:

$$\lim_{n \to \infty} \frac{n \cdot 7^{n/3}}{(7.5)^{n/3}} = \lim_{n \to \infty} n \cdot \left(\frac{14}{15} \right)^{n/3} = 0$$

(not hard to show using some calculus, may have even shown this in CS30 or CS31).

Thus, we have:

$$T(n) \in O((7.5)^{n/3}) = O(2^{\log_2(7.5) n/3})$$

with $\alpha := \frac{\log_2(7.5)}{3} \leq \frac{\log_2(8)}{3} = 1$. □