Lecture 6: Space Hierarchy and NP

Anup Rao

April 11, 2024

Last time we discussed the time-hierarchy theorem. Similarly, one can prove a Space hierarchy theorem. To do this, we need the concept of a space constructible function.

Definition 1 (Space Constructible Functions)**.** *We say that the map* $s : \mathbb{N} \to \mathbb{N}$ *is* space constructible *if* $s(n) \geq \log n$ *and on input* x *there is a Turing Machine that computes* $s(|x|)$ *in space* $O(s(|x|))$*.*

We saw in lecture 3 that:

Theorem 2. *There is a turing machine* M *such that given the code of any Turing machine* α *and an input* x *as input to* M*, if* α *takes* $S \geq \log |x|$ *space to compute an output for* x*, then* M *computes the same output in* $O(CS)$ *space, where here* C *is a number that depends only on* α *and not on* x*.*

One can prove the following space hierarchy theorem:

Theorem 3 (Space Hierarchy)**.** *If* q, s *are space-constructible functions satisfying* $q(n) = o(s(n))$*, then* $\mathsf{DSPACE}(q(n)) \subsetneq \mathsf{DSPACE}(s(n))$*.*

We leave out the details of the proof, since they are exactly the same as the previous result.

Here are some consequences of these hierarchy theorems:

Corollary 4. $\mathbf{P} \neq \text{Exp}$*.*

Proof On the one hand, $\mathbf{P} \subseteq \mathsf{DTIME}(n^{\log n})$. On the other hand, by the time hierarchy theorem, $\mathsf{DTIME}(n^{\log n}) \neq \mathsf{DTIME}(2^n)$, since $n^{\log n} = o(2^n)$. ■

Hierarchy Theorem for Circuits

We define the class $\mathsf{SIZE}(s(n))$ to be the set of functions $f : \{0,1\}^* \to \{0,1\}$ that can be computed by circuit families of size $s(n)$.

We have proved the following theorems:

Theorem 5. *Every function* $f : \{0,1\}^* \to \{0,1\}$ *is in* $\mathsf{SIZE}(O(2^n/n))$*.*

Theorem 6. *For every large enough* n*, there is a function* $f : \{0,1\}^n \to \{0,1\}$ *that cannot be computed by a circuit of size* $2^n/3n$*.*

We can use this theorem to prove a hierarchy bound for size.

Theorem 7. *There is a constant c such that for every functions $s(n), s'(n)$ satisfying $2^n/n > s'(n) > cs(n) > n$, we have that* $\mathsf{SIZE}(s(n)) \subsetneq \mathsf{SIZE}(s'(n))$.

Proof Suppose every function on n bits can be computed using a circuit of size $k2^n/n$. Set $c = 3k$. Let ℓ be such that $k2^\ell/\ell = s'(n)$. Then every function on ℓ bits can be computed by a circuit of size $s'(n)$. On the other hand, there is some function on ℓ bits that cannot be computed using a circuit of size $2^\ell/3\ell = s'(n)/c$, as required. ■

NP

In the last class, we introduced the concept of *complexity classes*. We saw the classes P, L, E, EXP and $PSPACE$. These classes were obtained by considering functions that can be computed with limited time or limited space. Today, we explore a different kind of class, the class NP.

Recall: $L \subseteq P \subseteq PSPACE \subseteq EXP$.

NP is interesting chiefly because many problems that we would like to solve efficiently with a computer, but cannot solve, belong to NP. The list of such problems includes essentially all problems solved today with machine learning, and many other practically important problems. Before giving the definition of NP, let us see some examples of problems in NP.

Independent Set Given a graph G and a number k, does the graph have an independent set of size k? Let $ISet(G, k) = 1$ if the graph has an independent set, and 0 otherwise.

Recall that an independent set is a set of nodes that does not contain any edges.

Subset sum : Given a list of numbers $a_1, \ldots, a_\ell, t$, is there some subset of the numbers $a_1, \ldots, a_\ell$ that sums to t? Let $SubSum(a_1, \ldots, a_\ell, t) = 1$ if there is such a subset, and 0 otherwise.

Composite numbers : Given a number N, decide if it is composite or not. Let $Comp(N) = 1$ if N is composite, and 0 otherwise.

Matching : Given a graph G and a number k, are there k disjoint edges in the graph? Let $Match(G, k)$ be 1 if there are k such edges, and 0 otherwise.

All of these problems have something in common: although it may be hard to efficiently compute the functions they define, it is very easy to *check* a solution if one is given to us! For example, if $ISet(G, k) = 1$, then there is a an independent set S of size k, and given G, S, k, one can check that S is an independent set of size k in polynomial time. Similarly, if $SubSum(a_1, \ldots, a_\ell, t) = 1$, then there is a subset of the numbers $S \subseteq \{a_1, \ldots, a_\ell\}$, that if given as input can be verified to have the sum t.

NP is the class of all functions f that have the above property, where if $f(x) = 1$, then this can be checked efficiently by an efficient verifier:

Definition 8. *$f : \{0,1\}^* \to \{0,1\}$ is in* **NP** *if there exists a polynomial p and a polynomial time machine V such that for every $x \in \{0,1\}^*$,*

$$f(x) = 1 \Leftrightarrow \exists w \in \{0,1\}^{p(|x|)}, V(x,w) = 1$$

V is usually called the *verifier* and w is usually called the witness or certificate or proof. For example, in the independent set problem above, the witness w would correspond to an independent set, and the verifier V would be the program that checks that w is in fact an independent set of size k in the input graph.

The witness w is restricted to being of polynomial length to ensure that the running time of V is actually polynomial in the length of x. If we allowed the witness to be arbitrarily long, then V would be allowed to run very long computations on x.

Many important combinatorial optimization problems can be cast as problems in **NP**.

P, NP and EXP

Fact 9. *$P \subseteq NP \subseteq EXP$.*

To see the first containment, observe that if $f \in P$, there is a polynomial time Turing machine M with $M(x) = f(x)$. But M itself is a verifier for f (with a witness of length 0) proving that $f \in NP$.

For the second containment, if $f \in NP$, then f has a verifier $V(x,w)$. Consider the algorithm that on input x runs over all possible w and checks if $V(x,w) = 1$. If any witness makes $V(x,w) = 1$, the algorithm outputs 1, otherwise it outputs 0. This algorithm computes f and runs in exponential time, so $f \in EXP$.

Nondeterministic Machines, and a Hierarchy Theorem

The original definition of **NP** was by considering Turing machines that are allowed to make non-deterministic choices: namely after each step, the machine is allowed to make a guess about which state to transition to in the next step. The machine computes 1 if there is a single accepting computational path, and 0 otherwise.

We can define $\mathsf{NTIME}(t(n))$ in the same way as $\mathsf{DTIME}(t(n))$, it is the set of functions computable by non-deterministic machines in time $O(t(n))$, and then you can check that $\mathbf{NP} = \bigcup_c \mathsf{NTIME}(n^c)$. Just as for deterministic time, there is a non-deterministic time hierarchy theorem:

Theorem 10. *If r, t are time-constructible functions satisfying $r(n+1) = o(t(n))$, then*

$$\mathsf{NTIME}(r(n)) \subsetneq \mathsf{NTIME}(t(n)).$$