More On NP Class

Frank

March 17, 2022

1 NP Class Continued

1.1 NP Problems

Thm 8.1: IS and TSP are in NP (Proof of TSP left as exercise)

Proof: Need a NTM M that decides IS

Need $b \log n$ bits to represent a set of b nodes (assuming $\log n$ bits per node). Thus, need computation tree of height $b \log n$ to generate all possible sets of b nodes.

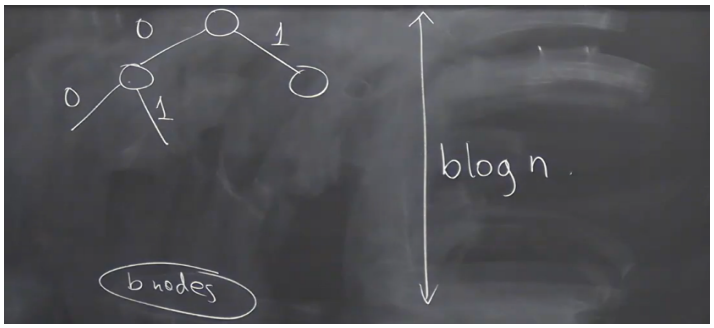

Figure 1: s21

Now, for each possible set of b nodes, we can verify in polynomial time, whether the set is an independent set

Note that we may have generated some "junk" that do not represent valid sets of b nodes. In this case just reject.

The above shows how to "guess" with NTMs

Note that the non-deterministic part takes $b \log n$ steps for M, then verifying takes roughly b^2 steps. Thus according to our definition, the running time of M is $O(n^2)$, polytime.

1.2 Alternative Definition of NP (Verifier)

All NP problems have the property of being easily verifiable. This means that given a yes-instance to the decision problem and a proof, we can easily verify that this is indeed a yes-instance by checking if the proof is valid.

However, no-instances are much more difficult to verify. For example, in the clique problem, given a set of nodes that does not form a clique (proof) does not mean the graph doesn't have cliques. But given a set of nodes that does form a clique, we can easily conclude that the graph does have cliques.

For example, "IS has a short and efficiently checkable certificate"

certificate: A string that "proves" the correctness of the yes-instance of the decision problem.

short: polynomial in $|x|$ (input to decision problem)

efficiently checkable: polytime in $|x|$

Certificates are:

comprehensive: each yes-instance has one

sound: no no-instance has one

Def: A Polytime verifier for $L \subseteq \Sigma^\star$ is a polytime algorithm V s.t.

$x \in L \iff \exists c_x \in \Sigma^\star$ s.t. $|c_x| = O(|x|^k)$ and V accepts $\langle x, c_x \rangle$

1.3 Alternative Definition on TSP

Given $\langle G = (N, E), wt, b \rangle \in TSP \iff \exists u_1, u_2, \ldots, u_n$ (permutation) of N (nodes) s.t.

1. $u_1, u_2, \ldots, u_n$ is a tour
2. $wt(\text{tour}) \leq b$

Checkable in polytime

1.4 Verifier

Thm 8.2: $L \in NP \iff L$ has a polytime verifier

Proof:

$(\implies)$

$L \in NP$ means that there's a polytime NTM M that decides L

Then given $x \in L$, the certificate would be an accepting computation path in M on x, then $c_x = \langle C_1, C_2, \ldots, C_l \rangle$, where C_1 is init config of M on x and C_l is accepting config s.t. each $C_i \vdash_M C_{i+1}$

- This certificate is short because $l \leq P(|x|)$ (poly running time of M, or number of steps M takes on the longest computation path)
- The size of each configuration is also bounded by $P(|x|)$ (M cannot write more cells than the number of steps it takes), so it is short
- Finally, polynomials are closed under composition, thus the certificate is "short"

Now we can "verify" that C_1 is indeed initial config (checking that $C_1 = q_0 x$), $C_l = x q_A y$, and since checking whether $C_i \vdash C_{i+1}$ is basically verifying that the local movement of the tapehead is "legal", it is polytime, and since we have a poly amount of steps, we have polytime as a result.

$\implies$ We can verify certificate in polytime

$\implies$ We have polytime verifier

($\Leftarrow$)

L has a polytime verifier means $\exists$ polytime verifier V for L

Here is a polytime NTM for L:

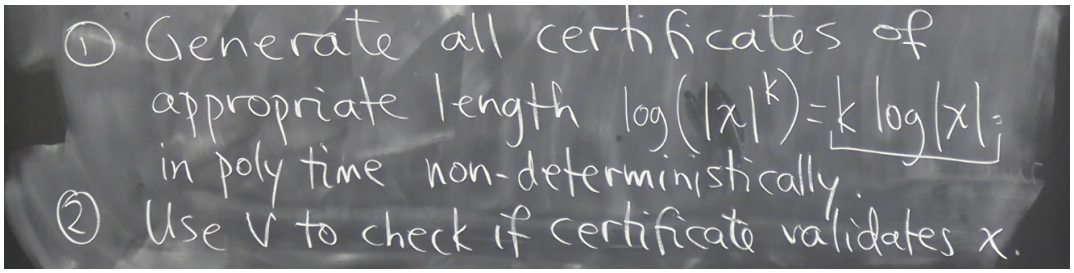

Figure 2: s22

General trend of arguing that a problem is in NP

- generate certificate $x \to c_x$
- argue c_x is polysize in $|x|$ (original input size to problem)
- describe verifier
- argue that the verifier is polytime

Note the connection between verifier and NTM.

certificate has to be polynomial in length of input because the NTM has to compute all possible certificates in polytime (to ensure the height of the computation tree is polynomial, hence number of steps is polynomial)

verifying each possible certificate has to be polynomial for the same reason (to guarantee the height of the tree is polynomial, hence the number of steps is polynomial)

Then, the overall height of the tree is polynomial, so NTM runs in polytime

2 NP Complete Problems

The IS and TSP are all NP complete problems.

NP complete problems are the hardest problems in NP, which means that all NP problems reduces to any of these problems.

2.1 Polytime Reduction

$A \leq_T B$ means that given a "black box" for B, we can create an algorithm that uses B to solve A.

Define $A \leq_T^P B$ [Cook Reduction] to be a polytime-turing reduction. This means that given a "black box" for B, we can create a polytime algorithm that uses B to solve A.

Finally, define $A \leq_m^P B$ to be a polytime-mapping reduction [Karp reduction].

Def [Cook reduction]: $A \leq_T^P B$ if there is a polytime algorithm A^B that uses an oracle for B to solve A.

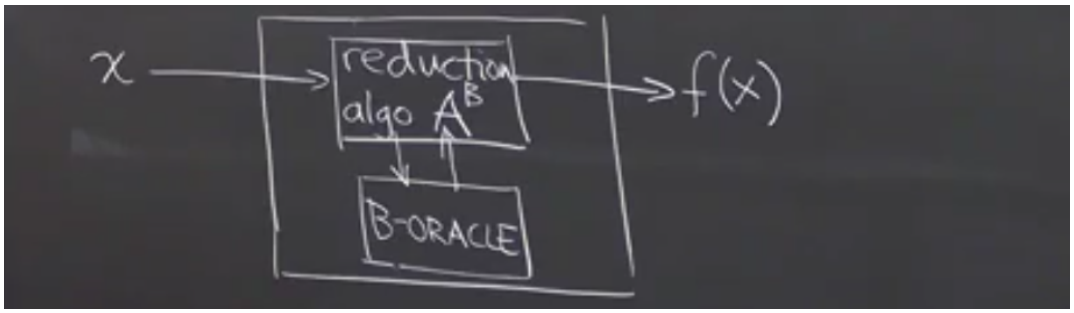

Figure 3: s23

convention: each usage of B-oracle counts as 1 step.

This means that if B is solvable in polytime and A uses B in a polytime algorithm, then we can conclude A is also polytime

Note: A or B need not be decision problems

Def [Karp reduction]: $A \leq_m^P B$ if there is a polytime computable $f : \Sigma^\star \to \Sigma'^\star$ s.t. $x \in A \iff f(x) \in B$

Note: A or B are decision problems

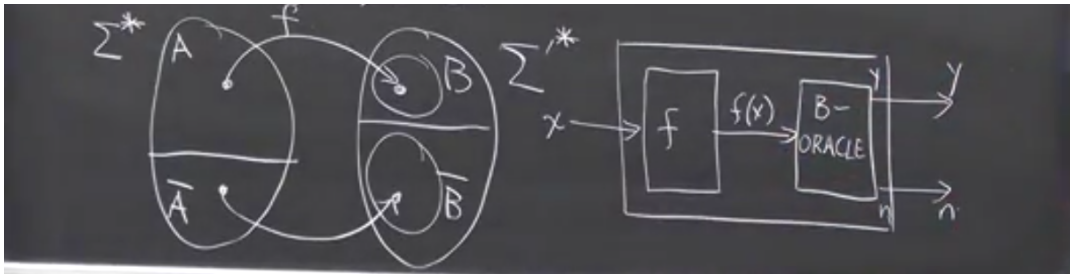

Figure 4: s24

Thm 8.3: If $A \leq_T^P B$ and B is polytime solvable, then A is polytime solvable.

Proof: Let A^B-SOLVER be a polytime algo solving A using B as a black box, and let B-SOLVER be a polytime algo for B.

Now replace each use of B-oracle with B-SOLVER. Result is polytime algo for A.

Note that A and B are not necessarily decision problems

Cor 8.4: If $A \leq_m^P B$ and $B \in P \implies A \in P$

Note that since Karp reduction applies only to decision problems, we can make arguments about the P-class.

Thm 8.5: $\leq_T^P$ is transitive.

Proof: By the fact that polynomials are closed under composition

Thm 8.6: $\leq_m^P$ is transitive

Proof: By the fact that mapping reduction is transitive and polynomials are closed under composition

2.2 Example of Polytime Reduction

$IS \leq_m^P ILP$

Where ILP is computation of m linear inequalities on n variables where variables are of integer values and all inequalities are satisfied.

example is $3x_1 + 17x_2 - 9x_3 \geq 217$

Note: linear programming problem (variables can be reals) is polytime solvable

Reduction idea:

Convert $\langle G, b\rangle \to ILP_{G,b}$ s.t. $\langle G, b\rangle \in IS \iff ILP_{G,b}$ has integer soln

Use variables for each node in the graph G

Write linear program so that

$$x_i = \begin{cases} 1 & \text{if node } i \text{ is in the IS} \\ 0 & \text{otherwise} \end{cases}$$

Write inequalities as such:

* Want at least b nodes in the IS, so we need the constraint $\sum_{i=1}^{n} x_i \geq b$
* We also want that if i is in the IS and i has an edge to j, then j must not be in the IS (o/w it is not IS). We express this constraint with: $\forall i, j$ s.t. $\{i, j\} \in E$, $x_i + x_j \leq 1$
* Also, we need the constraints: $\forall i$, $x_i \geq 0$ and $x_i \leq 1$ to satisfy the definition of x_i

Note that the number of inequalities is $\leq 1 + \frac{n(n-1)}{2} + 2n = O(n^2)$

Then f is the function that given $\langle G, b\rangle$ computes the integer linear program specified above (which takes polytime)

2.3 Definition

Def: L is an NP-complete problem (decision problem) or $L \in NPC$ if

1. $L \in NP$, and
2. $\forall L' \in NP$, $L' \leq_m^P L$

If 2. holds, L is NP-hard (at least as hard as NP-complete)

Thm 8.8: If $L \in NPC$, $L' \in NP$ and $L \leq_m^P L' \implies L' \in NPC$

Proof: If $L \in NPC$, then all problems in NP reduces to L, then by transitivity, all problems in NP reduces to L'. Also, by assumption, $L' \in NP$ (so it's not NP-hard), thus $L' \in NPC$

Note that to use this theorem, we need to "bootstrap" ourselves like we did with $\bar{D}$, so that we can begin reducing this "bootstrap" to other problems in order to prove they're NP-complete.

We will do so with a problem called SAT

2.4 SAT (satisfiability of propositional formulas)

An example of a propositional formula (formula) is $\Phi = (x \vee y) \wedge (\neg z \wedge \neg(x \vee \neg y))$

Truth assignment $\tau : vars \to \{0, 1\}$

Truth value of Φ under τ : $\tau(\Phi)$ is the truth value of the formula after using the variable assignment under τ.

Def: Φ is satisfiable if $\exists\tau$ s.t. $\tau(\Phi) = 1$

SAT

Instance: $\langle\Phi\rangle$
Q: is Φ satisfiable?

Thm: $SAT \in NPC$ (Cook-Levin)

3 Relationship Between Search and Decision Problems

Recall that a decision problem is in NP if there is a reasonably sized certificate that can be used to verify a yes-instance of the decision problem in polytime.

Now, in practice, we are mainly interested in what this certificate (or proof) is, instead of merely its existence.

Can show that decision $\leq_m^P$ search. This is very simple, just search and return true if search was successful and return false otherwise.

For example, for SAT

- Instance: $\langle\Phi\rangle$
- Decision: $\exists\tau : \tau(\Phi) = 1$?
- Search: Find τ s.t. $\tau(\Phi) = 1$, if one exists, otherwise output "none".

However, search $\leq_T^P$ decision for all NP-complete problems. (Proof uses oracle many times, hence not mapping-reduction). This is known as "self-reduction", where we reduce one form of a problem to another (in this case search form of the problem to its decision form).

3.1 Example

SAT-SEARCH $\leq_T^P$ SAT

First feed Φ into SAT oracle (SAT)

$$SAT(\Phi) = \begin{cases} \text{no} & \implies \text{"none"} \\ \text{yes} & \implies ? \end{cases}$$

Assume variables of Φ is $x_1, x_2, \ldots, x_n$, first set $x_1 = 0$, then we have $\Phi[x_1 = 0]$, which has $n - 1$ variables.

Now feed $\Phi[x_1 = 0]$ into SAT

$$\begin{aligned} \text{yes} &\implies \text{continue with } \Phi[x_1 = 0] \\ \text{no} &\implies \text{continue with } \Phi[x_1 = 1] \end{aligned}$$

Each time we ask the oracle whether $x_1 = 0$, we can infer whether $x_1 = 0$ makes Φ satisfiable; we continue with asking whether $x_2 = 0$ (given that $x_1 = 0$ or 1) and so on. Doing this n times gives us all the variables and their values.

SAT-SEARCH-SOLVER(Φ)

if SAT-ORACLE(Φ) = 0 then output "none"

else

Let $x_1, x_2, \ldots, x_n$ be vars of Φ

$\Phi_0 \leftarrow \Phi$

for $i = 1..n$ do

if SAT-ORACLE($\Phi_{i-1}[x_i = 0]$) = 1 then

$v_i \leftarrow 0$

$\Phi_i \leftarrow \Phi_{i-1}[x_i = 0]$

else

$v_i \leftarrow 1$

$\Phi_i \leftarrow \Phi_{i-1}[x_i = 1]$

output truth assignment $\tau : \tau(x_i) = v_i$

Claim: The above algorithm is a reduction of SAT-SEARCH $\leq_T^P$ SAT

Invariant: $\Phi_i = \Phi[x_1 = v_1, x_2 = v_2, \ldots, x_i = v_i]$ and Φ_i is satisfiable

Now, since we are traversing each variable in the formula with our for loop, the for loop is about linear in size of input (the formula has n variables plus repeats of the variables and $\wedge, \vee, \ldots$), in addition, copying the formula from the previous formula is also about linear in size of input, so we have $O(n^2)$

Exercise: IS-SEARCH $\leq_T^P$ IS

4 Optimization Versions of NPC Problems

Another form problem version aside from decision and search, is optimization.

For example, IS-OPT is the problem where we are given $\langle G \rangle$, and we are to find the IS with maximum cardinality

Another example is TSP-OPT, where we are given $\langle G, wt \rangle$ and we are to find the tour with minimum weight

We can prove with similar techniques that the optimization form of the problem turing-polynomial reduces to the decision form of the problem

4.1 Example

To find IS-OPT $\leq_T^P$ IS

1. Use IS-ORACLE to find maximum size of IS

   We can either linearly find the maximum size or use binary search
2. Search for IS of that size

4.2 Exceptions

We don't know if all problems in NP can have their search form reduced to their decision form in polynomial time.

For example, FACTOR-DEC, the problem of whether a number has factors greater than 1 is in P, while FACTOR-SEARCH is not known if it is in P