CS281 Notes

Boran Erol

March 2024

1 NP

A directed graph has an Hamiltonian cycle if and only if the adjacency matrix of the graph has 1's at every row and column.

2 Hierarchy Theorems

2.1 Exercises

2.1.1 Problem 1

CMU Graduate Complexity Theory HW1 Problem 1

Suppose by contradiction that there's a Turing Machine M such that $M(x) \neq L(x)$ only for finitely many x. We can then create a new Turing Machine M' that remembers all these finitely many strings and then checks if the input is equal to one of these before proceeding to simulate M. Thus, $M'(x) = L(x)$ for all x, which contradicts the fact that $L \notin$ TIME(n).

3 Space Complexity

3.1 Basic Notions

Intuitions about time complexity usually break down for space complexity.

Let's first define the basic notions.

Definition 1. A Turing Machine M runs in space $S(n)$ if

1. The input tape head doesn't go past the input. (You can't cheat using the input tape.)
2. Other heads reach at most $S(|x|)$ calls on any input x.

Notice that you can reuse cells, unlike time!

Definition 2. $L \in SPACE(S(n))$ if L is decided by a deterministic Turing Machine M in space $cS(n)$ for some constant $c > 0$.

Definition 3. $L \in NSPACE(S(n))$ if L is decided by a non-deterministic Turing Machine M in space $cS(n)$ for some constant $c > 0$.

Definition 4. $L \in PSPACE(S(n))$ if L is decided by a deterministic Turing Machine M in space $cS(n)$ where $S(n)$ is a polynomial in n.

Definition 5. $L \in NPSPACE(S(n))$ if L is decided by a non-deterministic Turing Machine M in space $cS(n)$ where $S(n)$ is a polynomial in n.

We'd like to recover as much as we can from the theory relating to time complexity.

Definition 6. A function $S : \mathbb{N} \to \mathbb{N}$ is **space-constructible** if

1. $S(n) \geq log(n)$
2. There exists a Turing Machine M that maps 1^n to $S(n)$ in space $O(S(n))$.

Why do we have the $log(n)$ restriction?

Without using $log(n)$, you can't even keep track of where you are on the input tape. There's not much you can do without doing this. You can decide the parity of the input and some other basic languages, but you can't decide interesting languages.

The second property is the analogue from time constructibility. The Turing Machine needs to be self-aware. It needs to be able to produce a yardstick which it can use to monitor its own space usage. Naturally, we require the yardstick to be computable in an efficient manner, at most as much space as its measuring.

Lemma 3.1. The universal Turing Machine U simulates any Turing Machine M on x with a constant-factor overhead in space.

Proof. □

Theorem 3.2. Let $s, S : \mathbb{N} \to \mathbb{N}$ be a space constructible functions. Assume $s(n) << S(n)$. Then, $SPACE(s(n)) \subseteq SPACE(S(n))$.

Proof. We'll use a diagonalization argument. Let D be the Turing Machine defined as follows.

Input: x, a description of a Turing Machine

1. Until $S(|x|)$ is exceeded, simulate M_x on x.
2. If M_x halts, output $\neq M_x(x)$.
3. Otherwise, output 0.

Clearly, D runs in space $O(S(n))$.

There's actually an issue with this construction. What if M_x uses very little space and loops forever? Then, D doesn't decide $L(D)$ but only recognizes it.

The way to solve this is to use an alarm clock. We need to set the alarm clock such that we only halt Turing Machines that are looping. **We'll use a theorem later to fix this issue, and I should fill this.** □

3.2 Configurations

Definition 7. A **configuration** of a (deterministic or non-deterministic) Turing Machine M consists of:

- current state of M
- current contents of each tape
- head locations

The key to understanding space complexity is to think in graph-theoretic terms.

Definition 8. A **configuration graph** G_m of a (deterministic or non-deterministic) Turing Machine M is an infinite directed graph where vertices correspond to configurations and edges correspond to transitions. In other words, there's an edge (C, C') if and only if C leads to C' in one-step.

Let's now recall some definitions from graph theory.

Definition 9. The **degree** of a directed graph $G(V, E)$ is

$$\sup\{deg^+(v) : v \in V\}$$

Notice that G_M has degree 1 if M is deterministic and degree 2 if M is non-deterministic.

Let x, y be unique strings. Since the input tape is immutable, there's no edge between $C_{M,x}$ and $C_{M,y}$ for any possible configuration.

Definition 10. Let $G_{M,x}$ be the subgraph reachable from $C_{M,x}^{START}$.

When a Turing Machine M halts, it doesn't need to leave its work tapes clean. However, without loss of generality, we can assume that M cleans its work tapes before halting. Therefore, we can assume that there's a unique accepting configuration denoted $C_{M,x}^{ACCEPT}$.

In this configuration, all work tapes are blank and all tape heads are at the origin. The output tape has a single 1 on it.

Proposition 3.3. Let M be a (deterministic or non-deterministic) Turing Machine M that runs in space $S(n)$. Then,

- We can represent vertices of $G_{M,x}$ by length $O(S(n))$ Boolean strings.
- On input x, we can construct $C_{M,x}^{START}$ and $C_{M,x}^{ACCEPT}$ in time $O(S(n))$.
- On input x, C, $C\prime$, we can check if $(C, C') \in G_{M,x}$ in time $O(S(n) + n)$.

Proof. A vertex of $G_{M,x}$ can be encoded by

- Currents contents and head locations for each read and write tape. This is $(k-1)S(n)log|\Gamma| = O(S(n))$.
- The current state of M. This is $log|Q| = O(1)$.
- Input head location. This is $log(n+1) \leq S(n+1)$.

The second bullet point is an immediate consequence of the first bullet point. (In fact, it's even easier, since the work tapes are empty.)

The third bullet point is also trivial. Just use the transition function to check if there's an edge between the two configurations. □

3.3 Time versus Space

Theorem 3.4. For every space-constructible $S : \mathbb{N} \to \mathbb{N}$,

$$DTIME(S(n)) \subseteq NTIME(S(n)) \subseteq SPACE(S(n))$$

Proof. Since we don't have any time requirements, we can just try out every possible non-deterministic guess and reuse the space. □

Theorem 3.5. For every space-constructible $S : \mathbb{N} \to \mathbb{N}$,

$$SPACE(S(n)) \subseteq NSPACE(S(n)) \subseteq DTIME(2^{O(S(n))})$$

Here, an initial proof attempt might be to run all possible branches of the non-deterministic Turing Machine M. However, notice that M can recycle the space it's using and branch 2^n times, running in 2^n time for every branch. Therefore, the simulation technique will fail.

Proof. We can construct $G_{M,x}$ in time $2^{O(S(n))}$ since we can construct every vertex in time $O(S(n))$ and we can check every edge one by one in $O(S(n))$ as well. Then, run your favorite graph algorithm to decide if $C_{M,x}^{ACCEPT}$ is reachable from $C_{M,x}^{START}$. □

These bounds are not tight at all. It is still open whether $P = PSPACE$.

3.4 Deterministic versus Nondeterministic Space

Theorem 3.6 (Savitch)**.** For every space-constructible $S : \mathbb{N} \to \mathbb{N}$,

$$NSPACE(S(n)) \subseteq SPACE(S(n)^2)$$

Proof. Let M be a non-deterministic Turing Machine that runs in space $S(n)$. Recall that $G_{M,x}$ has $2^{cS(n)}$ vertices and $M(x) = 1$ if and only if $C_{M,x}^{ACCEPT}$ is reachable from $C_{M,x}^{START}$.

We define an algorithm $LEADSTO$ as follows:

Input: Configurations C, C', and an integer l

Output: Accepts if and only if C leads to C' within l steps in $G_{M,x}$

1. If $l = 1$, then output ACCEPT if and only if $(C, C') \in G_{M,x}$.
2. For each C'', if $LEADSTO(C, C'', \lfloor \frac{l}{2} \rfloor)$ and $LEADSTO(C'', C', \lfloor \frac{l}{2} \rfloor)$, output ACCEPT.

Then, let $M(x) = LEADSTO(C_{M,x}^{START}, C_{M,x}^{ACCEPT}, 2^{cS(n)})$ and we're done.

The running time of this algorithm is absolutely horrendous, but it doesn't matter.

Notice that the height of the recursion stack is at most $cS(n)$. Notice that the graph allocates $O(S(n))$ space for every iteration of the for loop. We thus conclude the proof. □

Can you use this idea to prove that there exists an oracle O such that $EXP^O = NEXP^O$?

3.5 PSPACE-Completeness

One-way to understand the class PSPACE is to find complete problems.

Definition 11. L is said to be **PSPACE-Complete** if

1. $\forall L' \in$ PSPACE : $L' \leq_p L$
2. $L \in$ PSPACE

Why are we still considering polynomial time reductions instead of polynomial space reductions? Notice that if we consider polynomial space reductions, every language in PSAPCE is trivially PSPACE-Complete

Definition 12. A **totally quantified Boolean formula (TQBF)** has the form

$$Q_1x_1Q_2x_2...Q_nx_n : \phi(x_1, x_2, ..., x_n)$$

where Q_i are logical quantifiers.

This formula doesn't have any free variables in it. All variables are bound to quantifiers. Therefore, TQBFs are constants!

We also have **partially quantified Boolean formulas**.

Definition 13. $TQBF = \{\tau : TQBF_\tau \text{ is true}\}$

Notice that $SAT \leq_p TQBF$ since SAT is $TQBF$ where all quantifiers are $\exists$.

Lemma 3.7. $TQBF \in$ PSPACE

Proof. We construct a recursive algorithm.

□

Recursive algorithms work really well with space.

Let's now prove that $TQBF$ is PSPACE-Hard.

Lemma 3.8. $TQBF$ is PSPACE-Hard.

Proof. Let's start with a failed attempt. Let $L \in$ PSPACE and M be a Turing Machine that decides L in space $S(n)$, where $S(n) \leq n^c$ for some $c > 0$ for large enough n. By Savitch's construction, $M(x) = LEADSTO(C_{M,x}^{START}, C_{M,x}^{ACCEPT}, 2^{cS(n)})$.

Failed attempt:

$$LEADSTO(C, C', 2^{cS(n)}) = \exists C'' : LEADSTO(C, C'', 2^{cS(n)}) \wedge LEADSTO(C'', C', 2^{cS(n)})$$

□

Corollary 3.8.1. $L \in$ PSPACE if and only if there exists some polynomial time Turing Machine M such that

Do we need big-O notation to make this fully formal?

No, because you can store the first finitely many strings in the language of short length in the state of your Turing Machine M.

There's a recurring theme: adding quantifiers increases computational power.

SPACE = TIME + QUANTIFIERS

P = CNF + EXISTS

NP = CNF + EXISTS (Cook-Levin's Theorem)

PSPACE = CNF + polynomially many quantifiers

Here's an astronomical analogy:

EXP is the limits of the universe. We'll never reach there. PSPACE corresponds to the observable universe. This is where light can reach the earth. With a PSPACE computation, you can start it and the computation will produce an output after thousands of generations.

What about CNF + a constant number of quantifiers? If we have a single quantifier,

$$REACH = \{(G, u, v) : \exists \text{ a path from u to v}\}$$

Guess the logarithmic sequence in Savitch's proof. Since there's finitely many, we're good.

Challenge Problem

$$UNREACH = \{(G, u, v) : \text{ there's no path from u to v}\}$$

Prove that $UNREACH \in NSPACE(\log n)$.

4 The Polynomial Hierarchy

4.1 Motivation

The right way to approach the polynomial hierarchy is to ponder the role of quantifiers in computation. $n^{O(1)}$ quantifiers is usually overkill. However, one quantifier is often not enough. Here's an example:

$$\text{MINDNF} = \{(\phi, 1^s) : \exists \text{ DNF formula } \phi' \text{ of size at most } s \text{ such that } \phi \equiv \phi'\}$$

There's a natural solution to this problem using two quantifiers: $\exists \phi' \forall x : \phi(x) = \phi'(x)$

It's not known whether there's a single quantifier solution to this problem, and it is believed not to exist.

Definition 14. Let $k \geq 1$. $L \in \Sigma_k$ if and only if for some $c > 0$ and some polynomial time Turing Machine M,

$$x \in L \iff \exists u_1 \in \{0,1\}^{n^c} : \forall u_2 \in \{0,1\}^{n^c} : ... : M(x, u_1, ..., u_k) \text{ accepts}$$

Definition 15. Let $k \geq 1$. $L \in \Pi_k$ if and only if for some $c > 0$ and some polynomial time Turing Machine M,

$$x \in L \iff \forall u_1 \in \{0,1\}^{n^c} : \exists u_2 \in \{0,1\}^{n^c} : ... : M(x, u_1, ..., u_k) \text{ accepts}$$

Notice that $\text{co}\Sigma_k = \Pi_k$.

Observe that $\forall k \geq 0 : \Sigma_k \subseteq \Sigma_{k+1} \wedge \Pi_k \subseteq \Pi_{k+1}$ by ignoring the last quantifier.

Similarly, $\forall k \geq 0 : \Sigma_k \subseteq \Pi k + 1 \wedge \Pi_k \subseteq \Sigma_{k+1}$ by ignoring the first quantifier.

This gives rise to a nice mnemonic structure, which is the topic of the next section.

4.2 Lattice Structure

It is believed that all containments in this lattice are proper.

Definition 16.

$$\text{PH} = \Sigma_1 \cup \Sigma_2 ... = \Pi_1 \cup \Pi_2 ...$$

This is like a Jenga tower, pinching it horizontally or vertically causes it to collapse.

Suppose you'd like to prove a result in computational complexity and you can't prove it directly. One (possibly) easier goal is to prove your result assuming $P \neq NP$. One (possible) even easier goal is to prove your result assuming that the polynomial hierarchy collapses at some level k.

Theorem 4.1. If $\Sigma_k = \Pi_k$ for some k, then $PH = \Sigma_k$.

This can be thought of pinching the polynomial Jenga tower horizontally.

Proof. We'd like to show that $\forall m \geq n : \Sigma_m \subseteq \Sigma_k \wedge \Pi_m \subseteq \Sigma_k$.

We'll argue by inducting on m. The base case is our assumption. Suppose the statement holds for some $m \in \mathbb{N}$. □

Logspace Reductions are also used in computational complexity that is (possibly) weaker than polynomial time reductions. Other than that, polynomial time reductions are the "finest" reductions out there.

Conversion to a depth-2 formula requires a quantifier.

The empty language and its complement are P-Complete are polynomial time reductions for trivial reasons.

Theorem 4.2. If some L is PH-Complete, then $PH = \Sigma_k$ for some k.

Proof. Assume L is PH-Complete. Then, $\exists k \in \mathbb{N} : L \in \Sigma_k$. But then every L' in PH reduces deterministically in polynomial time to L, so $PH = \Sigma_k$. $\square$

4.3 Characterization of Σ_k using oracles

Theorem 4.3. $NP^{NP} = \Sigma_2$

Proof. We first show $\Sigma_2 \subseteq NP^{SAT}$. Let $L \in \Sigma_2$. Then,

$$x \in L \iff \exists u_1 : \forall u_2 : M(x, u_1, u_2) = 1 \iff \exists u_1 :\neq (\exists u_2 : M(x, u_1, u_2)$$

We now show the following inclusion.

$\square$

5 Randomness

Lemma 5.1. BPP = coBPP

Proof. □

5.1 Randomness vs. the Polynomial Hierarchy

In this section, we'll prove the theorem by Sipser and Gacz that shows $BPP \subseteq \Pi_2 \cap \Sigma_2$.

Before proving the theorem, observe that $BPP \subseteq PSPACE$ trivially since we can run the Turing Machine M for all possible randomness r and tally up the results.

Theorem 5.2. $BPP \subseteq \Pi_2 \cap \Sigma_2$

Proof. Let M be a polynomial time Turing Machine such that for all strings x,

$$\Pr[M(x,r) = L(x)] > 1 - \frac{1}{|x|^c}$$

We're going to prove the following claim:

$$x \in L \iff \exists u_1, ..., u_{|x|^c} \in \{0,1\}^{|x|^c} : \forall r \in \{0,1\}^{|x|^c} : \bigvee_{i=1}^{|x|^c} M(x, r \oplus u_i) = 1$$

Notice that the right hand-side has two logical quantifiers. Moreover, the operation is polynomial time. Then, $BPP \subseteq \Sigma_2 \implies coBPP \subseteq \Pi_2$. Since $BPP = coBPP$, we conclude the proof.

Let's now prove the claim. We define the notion of an *accepting set*. Fix $x \in L$ and define

$$A_x = \{r : M(x,r) = 1\}$$

Notice that

$$M(x, r \oplus u_i) = 1 \iff r \oplus u_i \in A_x \iff r \in A_x \oplus u_i$$

Then,

$$\bigvee_{i=1}^{|x|^c} M(x, r \oplus u_i) = 1 \iff r \in \bigcup_i (A_x \oplus u_i)$$

Since this is true for all $r \in \{0,1\}^*$, we have that

$$\bigcup_i (A_x \oplus u_i) = \{0,1\}^{|x|^c}$$

We have two possible cases:

Case 1: $x \notin L$ Intuitively, accepting sets are really small. More formally,

$$|A_x| \leq \frac{1}{|x|^c} \cdot 2^{|x|^c}$$

Then,

$$|\bigcup_i (A_x \oplus u_i)| \leq |x|^c \cdot |A_x| < 2^{|x|^c}$$

Case 2: $x \in L$ Intuitively, accepting sets are almost the entire set.

Then,

$$|A_x| \geq (1 - \frac{1}{|x|^c}) \cdot 2^{|x|^c}$$

We're going to use the probabilistic method to produce the u_i's.

If we pick the shifts at random, the probability that we cover everything is positive. Then, by the probabilistic method, this gives us a concrete instantiation. □

5.2 Exercises

5.2.1 Exercise 1

In the following definition, r is an infinite random string.

Definition 17. L $\in$ ZPP if there's a Turing Machine M such that for all strings $x \in \{0,1\}^*$,

- $M(x,r) = L(x)$ if M halts on x, r
- $\mathbb{E}_r$[Runtime of M on (x,r)] $< |x|^c$

Challenge Problem: We'd like to trade correctness of running time. Specifically, we'd like to prove ZPP = RP $\cap$ coRP.

If you have an algorithm that has no false positives and another algorithm that has no false negatives, you can combine them into an algorithm that has zero error and expected polynomial running time.

The solution to this challenge problem is related to the ideas from the coin flip problems.

5.2.2 Solution

Let $L \in$ ZPP. Let A be a zero-error randomized algorithm for L such that for all strings x,

$$\mathop{\mathbb{E}}_r[\text{run time A(x,r)}] \leq |x|^c$$

Using Markov's inequality,

$$\Pr_r[\text{run time A(x,r) exceeds } 100|x|^c] \leq \frac{1}{100}$$

Conversely, let $L \in$ RP $\cap$ coRP. Then, we have algorithms A', A'' that run in time $|x|^c$ such that A' has no false positives and A'' has no false negatives.

Here's how we define A:

1. Sample a random string r.
2. If $A'(x,r) = 1$, return 1.
3. If $A''(x,r) = 0$, return 0.
4. Repeat the first three steps.

Our new algorithm is clearly zero error! Moreover, we have that in every round

$$\Pr_r[\text{A terminates}] \geq \frac{2}{3}$$

Then, the expected the running time is

$$\sum_{i=1}^{\infty} 2|x|^c \cdot i \frac{1}{3^{i-1}} \frac{2}{3}$$

$$\frac{4}{3}|x|^c \sum_{i=1}^{\infty} \frac{i}{3^{i-1}}$$

$$\leq \frac{1}{3}(\frac{\sqrt(3)}{3})^i$$

Notice that the following attempt doesn't work:

Let T be the running time. Then, we have that

$$T \leq \frac{2}{3}2|x|^c + \frac{1}{3}(2|x|^c + T)$$

However, this assumes that T is finite, which is what we're trying to prover anyway.

Can you just argue using the number of rounds?

6 Evolution of Mathematical Proofs

Thales of Miletus is the first recorded mathematician. Interestingly, his proofs are much more interactive.

Euclid, in 300 BC, was the first person to use axioms to develop geometry.

7 Interactive Proofs

7.1 Motivation and Definitions

Definition 18. $L \in \text{IP}$ if there's a polynomial time machine V such that

- If $x \in L$, then there's a prover P such that

$$\Pr_{r \in \{0,1\}^{|x|^c}}[V^P(x,r) = 1] \geq \frac{2}{3}$$

- If $x \notin L$, then for all provers P we have that

$$\Pr_{r \in \{0,1\}^{|x|^c}}[V^P(x,r) = 0] \geq \frac{2}{3}$$

Definition 19. The **completeness** of an interactive proof is the probability that V accepts for some $x \in L$.

Definition 20. The **soundness** of an interactive proof is the probability that V rejects for some $x \notin L$.

In isolation, these are easy to achieve. To achieve perfect soundness, reject everything. To achieve perfect completeness, accept everything. However, achieving high soundness and completeness together isn't easy.

The power of interactive proofs comes from the fact that V has access to randomness. If V is deterministic, this reduces to NP again.

Lemma 7.1. Suppose there's an interactive proof system for deciding some language L where V is deterministic. Then, $L \in NP$. The converse of this statement is also true.

Proof. If V is determinstic, the prover can just compute the entire interaction on its own and send the transcript π to a verifier that just verifies that the transcript is an actual transcript by also running V. □

Lemma 7.2. If there's an interactive proof where the prover P' is allowed randomness for deciding L, there's an interactive proof with a deterministic prover P.

Proof. □

It's crucial that we're picking the prover **based on the verifier** here. This is about the definition of IP. In IP, we pick a verifier such that all provers act a certain way.

7.2 Graph Non-Isomorphism

Graph isomorphism (GI) is suspected to be NP-Intermediate.

Graph non-isomorphism (GNI) is not known to be in BPP or NP. However, it is clearly in coNP.

Lemma 7.3. GNI is in IP.

Proof. Here's our verifier that takes as input (G_1, G_2).

1. Sample a random permutation $\pi \in S_n$ and $i \in \{1, 2\}$.
2. Send $G = \pi(G_i)$ to the prover.
3. The prover sends a bit b. If $i = b$, accept. Otherwise, reject.

□

7.3 IP vs. PSPACE

Lemma 7.4. IP $\subseteq$ PSPACE.

Let's first provide an incorrect proof attempt.

incorrect proof attempt. Let $L \in$ IP and let V be a polynomial time verifier for L. Let x be a string. Then,

$$\Pr_r[V^P(x,r) = 1] \geq \frac{2}{3} \text{ if } x \in L$$

$$\Pr_r[V^P(x,r) = 1] \leq \frac{1}{3} \text{ if } x \notin L$$

Let $a_1, ..., a_{|x|^c}$ be the max prover's answers. We can then loop over all possible values of r and tally $V^{a_1,...,a_{|x|^c}}(x,r) = 1$ to calculate the probability. $\square$

This doesn't work, since it assumes the prover speaks once at the beginning of the protocol and doesn't produce answers $a_1, ..., a_{|x|^c}$ dependent on r. $a_1, ..., a_{|x|^c}$ should come after r since the provers answers depend on r.

Here's the actual proof:

Proof. Notice that V is a well-defined algorithm, but P is not. Our goal is to maximize the following probability over all choices of P.

The height of the tree is bounded above by $|x|^c$ since the running time of V is bounded by $|x|^c$. The arity? of every node in the tree is bounded above by $2^{|x|^c}$ since that is the longest message V can read.

Notice that picking a P corresponds to picking an action of P at every layer of this tree. Then, to find the maximum probability of acceptance, we should pick the action that maximizes the probability of acceptance at each step. This corresponds to picking one branch (given by orange).

Now, focus on the bottom layer. In the bottom layer, the probability that V accepts is completely independent of P. Therefore, we can just calculate probabilities of acceptance and rejection.

Let's now formalize the algorithm:

Algorithm $M(u_1, ..., u_i)$:

1. If $i = |x|^c$, return $V(u_1, ..., u_{|x|^c})$.
2. If it's a P layer, return $\max_{u_{i+1}} M(u_1, ..., u_{i+1})$.
3. If it's a V layer, return $\mathbb{E}u_{i+1}M(u_1, ..., u_{i+1})$.

There's a technicality we have to concern ourselves with. How do we calculate the expectation in V layers?

Sherstov: You can't condition on the first n bits of r and have a uniform distribution. You can't pick fresh randomness during the process. To see this, try modifying the proof for $GNI \in IP$.

For the nominator, tally up all the r's that take that branch.

For the denominator, tally up all the r's that end up in that layer.

This sounds a lot like conditional probability. Specifically, it sounds like we're conditioning on an r that reaches that V node and then calculating the conditional probability a branch is taken.

However, you can't reduce this proof to uniform conditional probabiltiies, because the verifier might be acting as it pleases. $\square$

Let's now prove the opposite inclusion. So far, we've seen that NP and BPP are contained in IP and GNI $\in$ IP.

7.4 Warmup: $coNP \subseteq IP$

Let's now show that coNP is also contained in IP. The next proof is a beautiful demonstration of the power of algebraic reductions.

UNSAT is coNP-Complete and it's a combinatorial problem. We'll turn UNSAT into an algebraic problem. We'll then use the additional algebraic structure in order to produce efficient interactive protocols.

The algebraic object will be an arithmetic circuit. Arithmetic circuits with addition and multiplication gates can be seen as elements in $F[x_1, ..., x_n]$. Using this dictionary, we can define the degree of an arithmetic circuit C as the degree of the multivariate polynomial $f \in F[x_1, ..., x_n]$.

The next lemma shows us that arithmetic circuits have efficient interactive protocols.

Lemma 7.5. The following problem has a polynomial time IP.

Input: An arithmetic circuit C over F, $a \in F$

Goal: Check whether

$$\sum_{x \in \{0,1\}^n} C_x = a$$

Completeness = 1

Soundness = $1 - \frac{ndegC}{|F|}$

Proof. We'd like to calculate

$$\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} ... \sum_{x_n \in \{0,1\}} C(x_1, ..., x_n)$$

Notice that

$$\sum_{x_2 \in \{0,1\}} ... \sum_{x_n \in \{0,1\}} C(x_1, ..., x_n)$$

is a polynomial $p \in F[x_1]$ and $deg(p) \leq deg(C)$.

Moreover, $\sum_x C(x) = a \iff p(0) + p(1) = a$.

In the protocol, the verifier asks for $p(x_1)$ and the prover sends $\tilde{p}(x_1)$. Then, V picks a random $r \in F$ and accepts if $p(r) = \tilde{p}(r)$. In order to check $p(r) = \tilde{p}(r)$, we use recursion.

Let's now analyze this algorithm. If the prover is truthful, it can convince the verifier with probability 1. Otherwise, if $p \neq \tilde{p}$, we have that

$$\Pr_r[\tilde{p}(r) \neq p(r)] \geq 1 - \frac{deg(C)}{|F|}$$

Then, using a union bound,

$$\Pr_r[\text{Verifier concludes that } \tilde{p}(r) \neq p(r)] \geq 1 - \frac{degC}{|F|} - \frac{(n-1)degC}{|F|} = 1 - \frac{ndegC}{|F|}$$

□

Lemma 7.6. coNP $\subseteq$ IP

Proof. Recall that UNSAT is coNP-Complete. Thus, it suffices to show that UNSAT is in IP.

Let ϕ be a CNF formula and C_ϕ be the arithmetic circuit for ϕ over a field F such that $|F| \geq |\phi| 2^n$.

Notice that C_ϕ acts exactly like ϕ on Boolean inputs. However, it is richer: we can evaluate C_ϕ on non-Boolean inputs.

Since F is large enough,

$$\phi \in \text{ UNSAT } \iff \sum_{x \in \{0,1\}^n} C_\phi(x) = 0$$

Moreover, there's an IP protocol for checking whether the right hand side with soundness greater than

$$1 - \frac{n|\phi|}{p} \geq 1 - \frac{n}{2^{n+1}}$$

You can make the field F smaller if you work harder, but we don't care. □

Let's now prove the actual result.

Lemma 7.7. PSPACE $\subseteq$ IP

Here's a failed naive attempt:

incorrect proof attempt. Since TQBF is PSPACE-Complete, it suffices to show that TQBF $\in$ IP. Here's the verifier with input $\exists x_1 \forall x_2 ... \phi(x_1, ..., x_n)$.

We'll use an interactive proof to check that

$$\sum_{x_1 \in \{0,1\}} \prod_{x_2 \in \{0,1\}} ... C_\phi(x_1, ..., x_n) \neq 0$$

$\square$

There are many issues with this proof attempt. The more interesting thing is that this proof is actually pretty close to the actual solution.

One issue is that the field needs to have size at least $2^{2^{n/2}}$.

Another issue is that the degree of this polynomial is $2^{n/2}$. Therefore, the prover can't send $p(x_1)$.

Here is where most researchers would stop. However, a good researcher would have the instinct to try to patch this solution.

To researchers in TCS today, you look at this object and you realize one thing: this object is over-algebrized. You have too much algebraic structure that is irrelevant to the problem at hand, which only concerns Boolean values. Therefore, we should get rid of it. We'll trade algebraic structure with efficiency.

We'll define three operators on $F[x_1, x_2, ..., x_n]$:

- $\forall_i : q(x_1, ..., x_i, ..., x_n) \mapsto q(x_1, ..., 0, ..., x_n) q(x_1, ..., 1, ..., x_n)$
- $\exists i : q(x_1, ..., x_i, ..., x_n) \mapsto (1 - q(x_1, ..., 0, ..., x_n))(1 - q(x_1, ..., 1, ..., x_n))$
- $L_i : q(x_1, ..., x_i, ..., x_n) \mapsto (1 - x_i) q(x_1, ..., 0, ..., x_n)) + x_i q(x_1, ..., 1, ..., x_n))$

The first two operators at most double the degree. The third operator changes the degree in x_i to 1. The value of q is unaffected on $\{0,1\}^n$. The third operator gets rid of the irrelevant algebraic structure and buys us some efficiency by ensuring that the degree doesn't blow up.

Lemma 7.8. There is a polynomial time interactive protocol for the following problem.

Input: $O_1, O_2, ..., O_m \in \{\exists_1, \forall_1, L_1, ..., \exists_n, \forall_n, L_n\}$. An arithmetic C over F. $(u_1, ..., u_n) \in F^n$,$a \in F$.

Goal: Check if

$$(O_1 O_2 ..., O_m)(u_1, ..., u_m) = a$$

Completeness = 1

Soundness = $1 - \frac{mD}{|F|}$, where $D = \max_i \{\deg(O_i O_{i+1} ... O_m C)\}$.

Proof. Let F be a field and C be an arithmetic circuit over F. Let $O_1 \in \{\exists_i, \forall_i, L_i\}$. Let $p(x_i) = (O_2 O_3 ... O_m C)(u_1, ..., u_{i-1}, x_i, u_{i+1}, ..., u_n)$. Here, x_i is purely formal. x_i doesn't stand for anything.

Here's the protocol:

1. The verifier asks for $p(x_i)$.
2. The prover sends $\tilde{p}(x_i)$.
3. The verifier randomly samples $r \in F$.
4. The verifier checks $p(r) = \tilde{p}(r)$ using recursion.
5. If the verifier thinks $p(r) = \tilde{p}(r)$, the verifier returns $(O_1, \tilde{p})(u_i) = a$ and accepts accordingly.

6. If the verifier thinks $p(r) \neq \tilde{p}(r)$, the verifier rejects.

Notice that the verifier can check whether $p(r) = \tilde{p}(r)$ by checking whether $(O_2O_3...O_mC)(u_1, ..., u_{i-1}, r, u_{i+1}, ..., u_n) = \tilde{p}(r)$.

Notice that even for this protocol to be efficient, D has to be a polynomial in $m, n, |C|$.

By assumption, $\deg p \leq D$.

Assume $(O_1O_2..., O_m)(u_1, ..., u_m) = a$. Then, we have two cases since **argue why yourself**

Case 1: $(O, \tilde{p})(u_i) \neq a$

In this case, we always reject.

Case 1: $(O, \tilde{p})(u_i) \neq a$

In this case, we always reject.

Case 2: $\tilde{p} \neq p$

By Schwarz-Zippel, $\Pr_r[p(r) \neq \tilde{p}(r)] \leq \frac{D}{|F|}$.

Then, the probability that the verifier thinks $p(r) = \tilde{p}(r)$ □

You might think that this is stronger than what we need. In the proof, we're only going to evaluate the circuit at the all zeros point. However, we need to be able calculate arbitrary sums for recursive steps, so we in fact need the full power of this lemma.

Now, notice that

$$\exists_{x_1} \forall_{x_2} \exists_{x_3} ... \exists_{x_{n-1}} \forall_{x_n} \phi(x_1, ..., x_n) = 1$$

is equivalent to

Notice that all intermediate degrees are less than or equal to $2 \deg C_\phi$, since we linearize after every quantifier.

Also notice that we're "evaluating" the arithmetic circuit at $(0, 0, 0, ..., 0)$, but this is only to be formally correct. At this point, the arithmetic circuit is already a constant, so it doesn't matter where we evaluate it.

The fact that IP = AM follows from the IP Characterization Theorem.

Let $IP(n)$ and $AM(n)$ denote the class of languages with n round IP and AM proofs, respectively.

Then, $IP(n) \subseteq AM(n + 2)$.

Babai and Moran: $IP(2n) \subseteq IP(n)$.

In Public Coin versus Private Coin Interactive Protocols, Goldwasser and Sipser prove that every private coin interactive protocol has a corresponding public coin interactive protocol (an Arthur-Merlin protocol)

Goldreich has a 2019 paper called On Emulating Interactive Protocols with public coins.

Theorem 7.9. Let L be the language of an interactive protocol with private coins. Then, there's an Arthur-Merlin protocol for L.

Proof. □

Theorem 7.10. If GI is NP-Complete, the polynomial hierarchy collapses.

Proof. □

8 Probabilistically Checkable Proofs

8.1 Motivation

The motivation for PCPs came from practitioners. If SAT is hard to solve, can we try to approximate it? In the quest to prove the hardness of approximation for 3SAT, PCPs were discovered.

After PCPs were discovered, a headline in NYT read: ”New Short Cut Found for Long Math Proofs”.

Sherstov thinks this is the most important achievement of theoretical computer science since Cook-Levin.

Traditional proofs (NP) has a ”proof” (witness) and an efficient verifier that checks the validity of the proof.

Let π be a proof. Then,

$$\max_{\pi} V^{\pi}(x) = L(x)$$

In other words, the verifier can only be convinced if $x \in L$.

Interactive proofs show that if you allow interaction, this process becomes incredibly powerful, bumping NP up to PSPACE.

In PCPs, you make the verifier less powerful compared to IPs. You just allow the verifier oracles access to the proof. PCPs are a TA's dream.

Let V be a deterministic Turing Machine. Let L be a language and x be a string. Let $n = |x|$. Let π a proof of length poly(n).

The verifier will only read $O(1)$ bits of the proof. Then, we consider

$$\max_{\pi} V^{\pi}(x) = L(x)$$

However, this makes the verifier too weak! We can just enumerate all possible bits the verifier probes and decide whether the verifier accepts or rejects in polynomial time. This is not weaker than polynomial time since the verifier runs in polynomial time.

In order to make the verifier stronger, we allow the verifier some randomness r and consider

$$\max_{\pi} \Pr_{r}[V^{\pi}(x, r)] = 1 \text{ if } x \in L$$

$$\max_{\pi} \Pr_{r}[V^{\pi}(x, r)] \leq \frac{1}{3} \text{ if } x \notin L$$

Don't get weirded out by these parameters. These can be adjusted by error correction and sequential repetition.

8.2 Introduction and Definitions

Definition 21. Let P be a computationally unbounded prover and V be a probabilistic polynomial-time algorithm. We say that (P, V) is a **PCP system** for a language L with completeness error ϵ_c and soundness error ϵ_s if the following holds:

Completeness:
$$\forall x \in L : \Pr[V^{\pi}(x, r) = 1] \geq 1 - \epsilon_c$$

Soundness:
$$\forall x \notin L : \forall \tilde{\pi} : \Pr[V^{\tilde{\pi}}(x, r) = 1] \leq \epsilon_s$$

Let's now define some parameters.

Σ is the alphabet. l is the proof size. q is the query complexity. r is the randomness complexity.

Definition 22. We denote by the complexity class **PCP** languages decidable by PCP proof system with paramers $\Sigma = exp(n), l = exp(n), q = poly(n), r = poly(n)$.

Notice that the proof can't be larger than exponential size since we can't query larger proofs: queries of exponential proofs have polynomial size.

Here are some questions we can explore:

1. Which languages have PCPs? At least more than PSPACE.
2. Do PCPs have benefits for NP languages? Yes.
3. Do PCPs have benefits for tractable languages? Yes.
4. Are there ZK PCPs for NP languages? Yes.

As seen, PCPs are very powerful. However, the PCP model is weird: the PCP verifier has oracle access to a large proof.

PCPs are useful for hardness of approximation problems and interactive arguments in cryptographic primitives.

We can actually show that PCP $\subseteq NEXP$.

Lemma 8.1. For non-adaptive verifiers, $l \leq 2^r q$.

For adaptive verifiers, $l \leq 2^r |\Sigma|^q q$.

Proof. Notice that using r random bits and q queries, a verifier can query proofs up to length $2^r q$. Therefore, without loss of generality, we can use proofs of this length. For adaptive verifiers, the argument is similar, however, we also take into account the different decisions the verifier can make based on the characters it reads, which explains the Σ term. $\square$

Lemma 8.2 (Derandomization)**.** PCP$[l, r] \subseteq NTIME((2^r + l) \cdot poly(n))$

Proof. Let (P, V) be a PCP system for L where the PCP verifier uses r random bits to query a proof of length l. Consider the decider $D(x, \pi)$ that loops over all possible choices of randomness, tallies up the number of times V accepts and accepts if and only if the tally is greater than a $1 - \epsilon_c$ fraction of the runs. This verifier still needs to guess the proof, so it's a non-deterministic Turing Machine. $\square$

Corollary 8.2.1. PCP $\subseteq NEXP$

Let's now show that $PSPACE \subseteq PCP$. In order to do this, we'll show $IP \subseteq PCP$.

Before we dive into hardness of approximation and slowly develop the PCP theorem, here's the theorem:

Theorem 8.3. Let $L \in$ NP. Then, L has a PCP verifier V with $c \cdot \log n$ bits of randomness and c queries, where c is a constant.

Notice that the converse of this theorem holds trivially since the randomness is logarithmic, so we can enumerate over the randomness using a nondeterministic polynomial time Turing Machine.

Notice that $2^{c \log n} \cdot c$ is an upper bound on the length of all proofs, since otherwise the verifier can't even pick bits of the proof to look at.

Here's how we build a non-deterministic Turing Machine N for L: (*I don't understand this argument.*)

On input x, guess a proof π of length $2^{c \log n} \cdot c$.

If $\forall r \in \{0,1\}^{c \log n} : V^{\pi}(x,r) = 1$, accept. Otherwise, reject.

Since we're checking logarithmically many bits of the proof, we can even check exponential-sized proofs!

The idea will be to endow the proof with a rich structure that allows checking global properties via local constraints.

8.3 Approximate Hardness of SAT

People in industry don't think SAT is hard: we have SAT solvers!

Definition 23. Given a CNF ϕ, opt(ϕ) is the maximum number of clauses that can be satisfied by a single assignment.

Theorem 8.4. There's a deterministic polynomial time algorithm that takes in as input a CNF formula ϕ and finds x satisfying $\geq \frac{1}{2}\text{opt}(\phi)$ clauses.

Proof. Let $\phi = C_1 \wedge C_2 \wedge ... \wedge C_n$. For every x_i, set the value of x_i to satisfy at least half of the clauses where x_i appears.

How do you rigorously extend this to satisfy all clauses? □

Notice that this is in fact better than how we introduced it. This algorithm actually satisfies half of all clauses! A corollary of this is $\text{opt}(\phi) \geq m/2$.

A natural question to ask after seeing this theorem is: Can you satisfy $(1-\epsilon)\text{opt}(\phi)$ clauses in polynomial time?

The answer, unfortunately, turns out to be no.

Theorem 8.5. For some constant $\epsilon > 0$, it is NP-Hard to device for a given 3-CNF ϕ whether

- ϕ is satisfiable.
- $\text{opt}(\phi) \leq (1-\epsilon)m$.

Notice that this theorem is an extremely strong answer to the previous question we asked. The following theorem would've sufficed:

Theorem 8.6. For some constant $\epsilon > 0$, it is NP-Hard to device for a given 3-CNF ϕ whether

- $\text{opt}(\phi) \geq (1-\epsilon_1)m$
- $\text{opt}(\phi) \leq (1-\epsilon_2)m$.

8.4 Approximate Hardness of Independent Set

Theorem 8.7. For any $\delta > 0$, it is NP-Hard to find an independent set of size $\geq \delta \cdot \text{opt}$.

Basically, this theorem tells us that if you want to approximate an independent set, you're doomed.

The proof goes through using a reduction of SAT to INDSET.

Recall that the reduction worked such that $\text{opt}(\phi) = \text{opt}(G)$.

Therefore, approximation algorithms for INDSET translate to approximation algorithms for 3-SAT.

Notice that this immediately carries over the previous approximation theorem for 3-SAT to INDSET. However, we're striving for a stronger result.

In order to do this, we need a concept from graph theory. We now define powers of graphs.

Let G be an undirected graph. In G^k, two vertices are adjacent

Definition 24. Let G be an undirected graph. Here's how we define G^k,

The vertex set of G is all k-subsets of vertices in G.

$(S, T) \in G^k$ if and only if G has an edge with both endpoints in $S \cup T$.

Lemma 8.8. $\{S_1, S_2, ..., S_r\}$ is an independent set in G^k if and only if $S_1 \cup S_2 \cup ... \cup S_r$ is independent in G.

Here's some combinatorics terminology:

Definition 25. Let S be a set.

$$\binom{S}{k} = \{A \subseteq S : |A| = k\}$$

You can think of $\binom{n}{k}$ as n^k among friends.

Proof. □

8.5 Approximate Hardness of Vertex Cover

Recall that S is a vertex cover if and only if $\bar{S}$ is an independent set.

However, interestingly, it isn't hard to approximate vertex cover.

Theorem 8.9. For **some** $\delta > 0$, it is NP-Hard to find a vertex cover of size $\leq (1 + \delta)(\text{opt}(G))$.

Proof. For every clause, you get 7 vertices. □

However, we can't upgrade this to any $\delta > 0$, since we have an explicit algorithm to approximate vertex cover to a factor of 2!

Notice that this doesn't contradict the inapproximability of independent set since

We now present a polynomial time algorithm to approximate vertex cover to size $\leq 2\text{opt}$.

The algorithm works as follows:

Let $S = \varnothing$.

While $E \neq \varnothing$:

1. Take any edge $(u, v) \in E$.
2. Add u, v to S.
3. Remove any edge incident on u or v.

The chosen edges in vertex cover form a matching.

8.6 The PCP Theorem

Definition 26. QUADEQ $= \{Q : Q$ is a system of quadratic equations over F_2 that has a solution$\}$.

Recall the LINEQ can be decided using Gaussian Elimination.

Lemma 8.10. QUADEQ is NP-Complete.

Proof. Separately for each clause. Proof in Notability PCP document. $\square$

Theorem 8.11 (PCP Theorem, weak version)**.** Every language in NP has a PCP verifier with $O(1)$ queries and $poly(n)$ random bits.

Proof. It suffices to show that there's a PCP verifier for QUADEQ.

$\square$

Sherstov doesn't think reducing the amount of randomness to $\log n$ is cool.