

D7: P vs NP



Sec 101: MW 3:00-4:00pm DOW 1018
IA: Eric Khiu

Midterm Survey

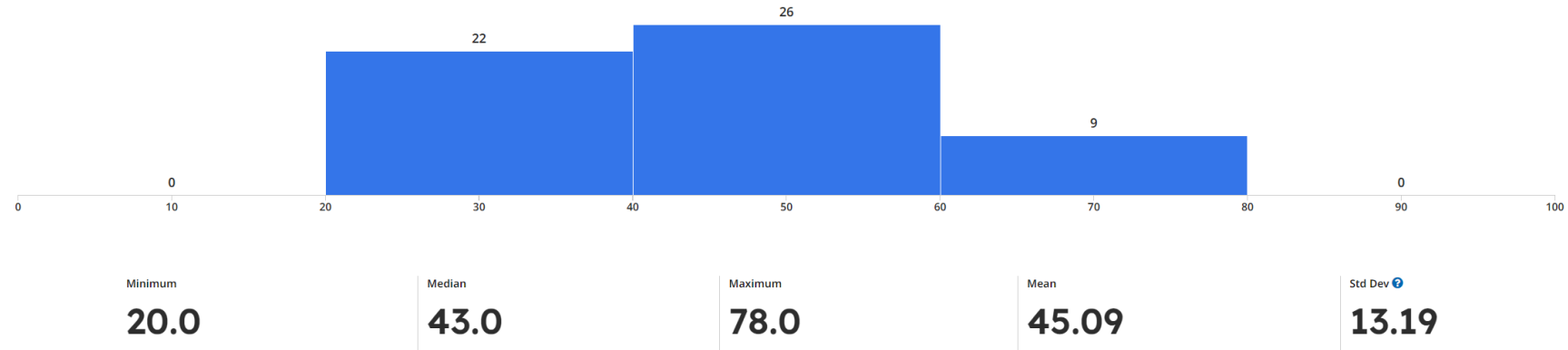


Please share your experience with EECS 376 so far!
Thank you in advance for your feedback!

Announcement

- ▶ Midterm exam results have been released on Gradescope.
- ▶ The regrade request window is open until next **Monday, June 10, at 11:59 PM**. A few important note about regrade requests:
 - ▶ Before submitting a request, please go through the [published solution](#) and compare it with your own.
 - ▶ In your regrade request, **you must explain why you believe the rubric was applied incorrectly** and specify the score you think you should receive according to the rubric.
 - ▶ We accept requests only to correct grading errors, not disagreements with the rubric.
 - ▶ Your score on the regraded problem **may increase or decrease**.
 - ▶ You may submit up to one request per (sub-)question; **no second regrade request will be accepted for the same question**.

Midterm Statistics

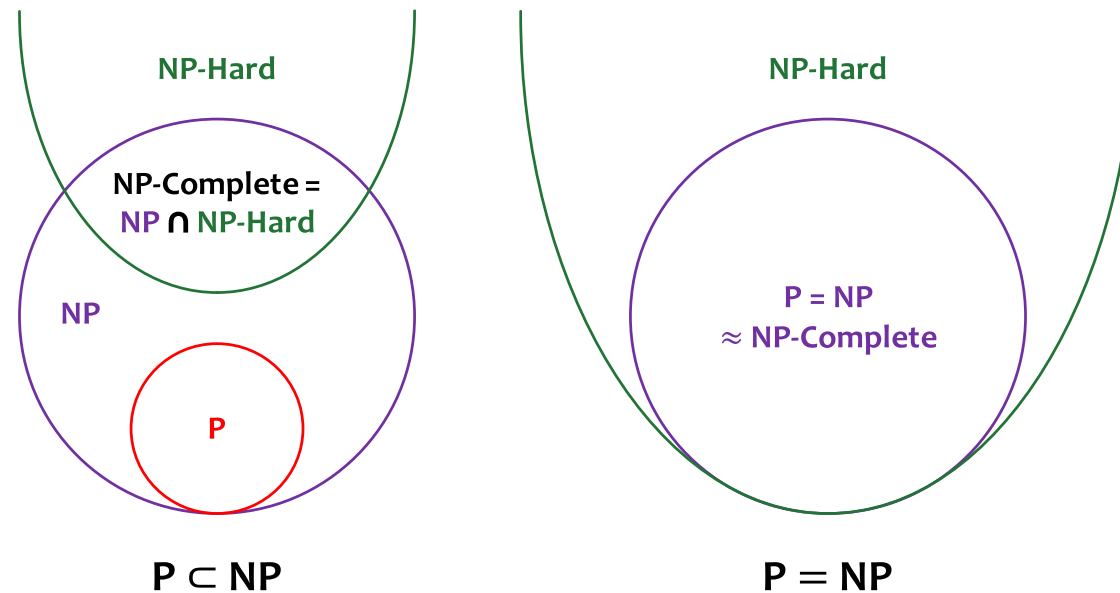


Tag	Question	Points	Mean
SAMC	4 questions	16 points	<div><div></div></div> 43%
MAMC	5 questions	30 points	<div><div></div></div> 64%
SA	6 questions	24 points	<div><div></div></div> 40%
LA	6 questions	30 points	<div><div></div></div> 30%
Tag	Question	Points	Mean
Algorithms	10 questions	53 points	<div><div></div></div> 48%
Computability	11 questions	47 points	<div><div></div></div> 40%

Keep Pushing Forward: Embrace New Challenges

- ▶ **New Topics Ahead:** Get ready for engaging subjects like **Complexity**, **Randomness**, and **Cryptography**. If earlier topics were boring/ challenging, these might just pique your interest!
- ▶ **Mathematical Shift:** We're moving towards **Probability** and **Number Theory**—a change that could align with your strengths.
- ▶ **Strength in Variety:** Everyone excels differently. A tough midterm doesn't define your final performance.
- ▶ **Stay Engaged:** Keep an open mind and stay engaged. The upcoming sections offer a fresh start and new opportunities to shine!

Unit 3: Complexity



Sec 101: MW 3:00-4:00pm DOW 1018
IA: Eric Khiu

Agenda

- ▶ Complexity Intro
- ▶ The Class P
- ▶ The Class NP
- ▶ P vs NP

Complexity: Introduction

- ▶ Last unit: What *can* and *can't* a computer solve
- ▶ This unit: What can and can't a computer solve *efficiently*
- ▶ **Complexity Class:** $DTIME(t(n))$ is the *class* of all languages decidable by a TM with time complexity $O(t(n))$

The Class P



The Class P

- ▶ **Definition:** P is the class of languages decidable by a **polynomial-time** Turing machine, where

$$P = \bigcup_{k \geq 1} DTIME(n^k)$$

- ▶ Given the significant difference between polynomial and exponential time, we informally consider P to be the class of **efficiently decidable languages**

Discuss: Given the definition of class P , what do we need to show to prove that a language is in class P ?

- ▶ Give the algorithm of the efficient decider (TM)
- ▶ Correctness proof (from last unit)
- ▶ **Runtime analysis (NEW!)**

Example: Spanning Trees

- ▶ **Definition:** A **spanning tree** of a graph $G = (V, E)$ is a subset of edges where every vertex is included in the subgraph
- ▶ Consider the following language:
$$L_{SPAN-k} = \{\langle G, k \rangle : G \text{ has a spanning tree of weight less than } k\}$$
- ▶ We can decide this language efficiently:
D = “On input $\langle G, k \rangle$:
 MST \leftarrow KRUSKAL(G)
 If **WEIGHT**(MST) < k then accept; else reject”

Example: Spanning Trees

$$L_{SPAN-k} \{ \langle G, k \rangle : G \text{ has a spanning tree of weight less than } k \}$$

D = “On input $\langle G, k \rangle$:

MST \leftarrow KRUSKAL(G) // $O(|E| \log |E|)$

If WEIGHT(MST) $< k$ then accept; else reject”

► Correctness analysis:

- $\langle G, k \rangle \in L_{SPAN-k} \Rightarrow \exists$ spanning tree T of G s.t. $\text{Weight}(T) < k$

Def of MST

$\Rightarrow \text{Weight}(\text{MST}) \leq \text{Weight}(T) < k \Rightarrow D(\langle G, k \rangle)$ accepts

- $\langle G, k \rangle \notin L_{SPAN-k} \Rightarrow \text{Weight}(T) \geq k$ for all spanning trees T of $G \Rightarrow \text{Weight}(\text{MST}) \geq k$
since MST is a spanning tree $\Rightarrow D(\langle G, k \rangle)$ rejects

► Runtime analysis:

- Kruskal Algorithm runs in $O(|E| \log |E|)$: efficient

Example: GCD

- ▶ Prove that the language $GCD = \{(x, y, g) : \gcd(x, y) = g\}$ is in class P
- ▶ Consider the following decider:

$D =$ “On input (x, y, g) :

$G \leftarrow 0$

if $x < g$ **or** $y < g$ **or** $g < 0$ **then** reject

for $d = 1, \dots, \min(x, y)$ **do**

if d divides both x and y **then** $G \leftarrow d$

if $G = g$ **then** accept; **else** reject”

- ▶ Correctness analysis:
 - ▶ $(x, y, g) \in GCD \Leftrightarrow g = \gcd(x, y) \Leftrightarrow g \text{ is the last } d \text{ getting assigned to } G \Leftrightarrow G = g \Leftrightarrow D(x, y, g) \text{ accepts}$
- ▶ Runtime analysis:
 - ▶ $O(\min(x, y))$: efficient

Discuss: What is wrong with this proof?

Exercise: $L_{<376376}$

- ▶ Show that the following language is in P

$$L_{<376376} = \{(\langle M \rangle, x) : M \text{ is a TM that halts on } x \text{ in less than } |x|^{376376} \text{ steps}\}$$

- ▶ Let D be the efficient decider for $L_{<376376}$

- ▶ **Correctness Proof Draft**

- ▶ $(\langle M \rangle, x) \in L_{<376376} \Rightarrow$ $\Rightarrow D(\langle M \rangle, x)$ accepts
 - ▶ $(\langle M \rangle, x) \notin L_{<376376} \Rightarrow$ $\Rightarrow D(\langle M \rangle, x)$ rejects

- ▶ **Exercise:** Design the decider to fill in the two blanks above + runtime analysis
- ▶ **Hint:** Consider $|x|^k$ for small x and k for intuition

Exercise: $L_{<376376}$

$$L_{<376376} = \{(\langle M \rangle, x) : M \text{ is a TM that halts on } x \text{ in less than } |x|^{376376} \text{ steps}\}$$

- Consider the following decider:

D = “On input $(\langle M \rangle, x)$:

Counter $\leftarrow 1$

While M has not halt on x **do**

If counter $\geq |x|^{376376}$ **then** reject

 Execute the (counter)th step of M on x

 counter \leftarrow counter + 1

accept”

- Or alternatively,

D = “On input $(\langle M \rangle, x)$:

 Simulate execution of $M(x)$ for up to $|x|^{376376} - 1$ steps

If M halts during this execution **then** accept; **else** reject”

Exercise: $L_{<376376}$

$$L_{<376376} = \{(\langle M \rangle, x) : M \text{ is a TM that halts on } x \text{ in less than } |x|^{376376} \text{ steps}\}$$

D = “On input $(\langle M \rangle, x)$:

Simulate execution of $M(x)$ for up to $|x|^{376376} - 1$ steps

If M halts during this execution then accept; else reject”

► Correctness Analysis

- $(\langle M \rangle, x) \in L_{<376376} \Rightarrow M$ will halt on x within $|x|^{376376} - 1$ steps $\Rightarrow D(\langle M \rangle, x)$ accepts
- $(\langle M \rangle, x) \notin L_{<376376} \Rightarrow M$ won't halt on x within $|x|^{376376} - 1$ steps $\Rightarrow D(\langle M \rangle, x)$ rejects

► Runtime analysis

- Running $|x|^{376376} - 1$ steps of a program is polynomial w.r.t. $|x|$, the only other work done is returning a bool, so this decider is efficient

TL; DPA

- ▶ P is the class of languages decidable by a polynomial-time Turing machine
- ▶ To prove that a language is in class P
 - ▶ Give the algorithm of the efficient decider
 - ▶ Correctness analysis
 - ▶ Runtime analysis (new)

Take Home Exercise 1

- ▶ Let $L_1, L_2 \in P$. Determine whether the following statements are always/ sometimes/ never true (This can be helpful to put on your cheatsheet!)
 - ▶ $\overline{L_1} \in P$
 - ▶ $L_1 \cap L_2 \in P$
 - ▶ $L_1 \cup L_2 \in P$
 - ▶ $L_1 \setminus L_2 \in P$

The Class NP



Starter: Verifiers and Certificates

- ▶ Consider the following language

$$PAIRSUM = \{(S, k): \exists a, b \in S \text{ s.t. } a + b = k\}$$

- ▶ Determine if the followings are in *PAIRSUM*
 - ▶ $S_1 = \{2, 4, 6\}$, $k_1 = 6$
 - ▶ $S_2 = \{2, 4, 6\}$, $k_2 = 7$

Share with us: What was your thought process?

Verifiers and Certificates

- ▶ A **certificate** c is an additional information used to check whether an input x is in a language L
- ▶ A **verifier** V for a language L is a TM that takes in (x, c) that satisfies:
 - ▶ $\forall x \in L$, there exists some certificate c such that $V(x, c)$ accepts
 - ▶ $\forall x \notin L$, there is **no** c such that $V(x, c)$ accepts
- ▶ For $PAIRSUM = \{(S, k): \exists a, b \in S \text{ s.t. } a + b = k\}$, $x = (S, k)$, we can have $c = (a, b)$, and we can build a verifier as follows:

$V =$ “On input $(x = (S, k), c = (a, b))$:

if $a \notin S$ or $b \notin S$ **then** reject

if $a + b = k$ **then** accept; **else** reject”

Note: You format how you want the certificate to look like!

Formatting Certificate

- Be creative! There is no *one correct way* to format the certificate. For example, we could have

V = “On input $(x = (S, k), c = (e^a, e^b))$:

if $\ln e^a \notin S$ or $\ln e^b \notin S$ then reject

if $e^a \cdot e^b = e^k$ then accept; else reject”

- You can even take in a certificate and **ignore it!**

V = “On input $(x = (S, k), c = \text{"duck"})$:

for pairs $(a, b): a, b \in S$ do

if $a + b = k$ then accept

reject”

The Class NP

- ▶ A language is *efficiently verifiable* if there exists an **efficient verifier** that verifies it
 - ▶ **Note 1:** The certificate must be polynomial in $|x|$
 - ▶ **Note 2:** Since the runtime of V is polynomial in $|x|$, the machine **halts on any input**
- ▶ **Definition:** NP is the set of **efficiently verifiable languages**
- ▶ To prove that a language is in class NP
 - ▶ Give the algorithm of the efficient **verifier**
 - ▶ Correctness analysis
 - ▶ Runtime analysis

Example: PairSum

- ▶ Prove that $PAIRSUM = \{(S, k) : \exists a, b \in S \text{ s.t. } a + b = k\}$ is in class NP .

- ▶ **Efficient Verifier:**

```
V = "On input  $(x = (S, k), c = (a, b))$ :  
    if  $a \notin S$  or  $b \notin S$  then reject  
    if  $a + b \neq k$  then reject  
    accept"
```

- ▶ **Runtime analysis:**

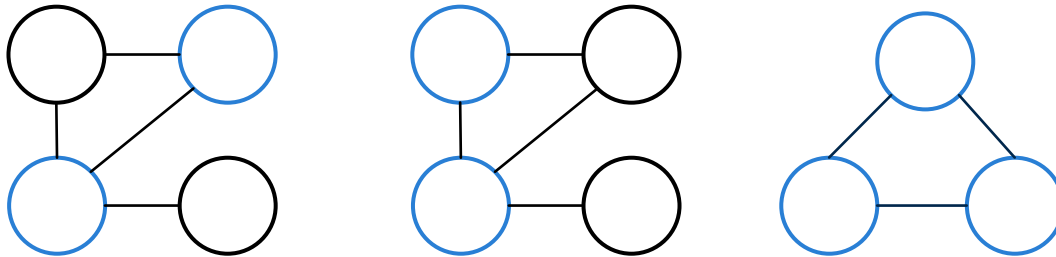
- ▶ It takes $O(|S|)$ to check if a and b are in S , and $O(1)$ to check if $a + b = k$. Therefore, verifier is efficient

- ▶ **Correctness analysis:**

- ▶ $(S, k) \in PAIRSUM \Rightarrow \exists a, b \in S : a + b = k \Rightarrow V$ will accept when given this (a, b) pair as certificate
- ▶ $(S, K) \notin PAIRSUM \Rightarrow \nexists a, b \in S : a + b = k \Rightarrow V$ will never accept any (a, b) certificate

Exercise: k -CLIQUE

- ▶ **Definition:** A **clique** of an undirected graph $G = (V, E)$ is a subset of vertices $C \subseteq V$ such that there is an edge in E between every pair of vertices in the C
- ▶ For example:



- ▶ A k -clique is a clique with k vertices
- ▶ Prove that the following language is in class NP

$$k\text{-CLIQUE} = \{(G = (V, E), k) : G \text{ is an undirected graph that has a } k\text{-clique}\}$$

Exercise: k -CLIQUE

$k\text{-CLIQUE} = \{(G = (V, E), k): G \text{ is an undirected graph that has a } k\text{-clique}\}$

- ▶ Let F be an efficient verifier for $k\text{-CLIQUE}$
- ▶ **Step 1: Format of the certificate**
 - ▶ Let $V' \subseteq V$ such that $|V'| = k$
- ▶ **Step 2: Correctness proof draft**
 - ▶ $(G, k) \in k\text{-CLIQUE} \Rightarrow \exists V' \subseteq V$ s.t. V' is a clique of size $k \Rightarrow$
 $\Rightarrow F$ will accept when given this V' subset as certificate
 - ▶ $(G, k) \in k\text{-CLIQUE} \Rightarrow \nexists V' \subseteq V$ s.t. V' is a clique of size $k \Rightarrow$
 $\Rightarrow F$ will never accept any V' certificate
- ▶ **Step 3: Construct efficient verifier**
 - ▶ **Exercise:** Give an algorithm of the efficient verifier
 - ▶ **Hint:** What does “every pair of vertices in V' have an edge in E ” mean?

Exercise: k -CLIQUE

$k\text{-CLIQUE} = \{(G = (V, E), k) : G \text{ is an undirected graph that has a } k\text{-clique}\}$

► Efficient Verifier

```
F = "On input  $(G = (V, E), V')$ :  
    if  $|V'| \neq k$  or  $V' \not\subseteq V$  then reject  
    for all pairs of vertices  $u, v \in V'$   
        if  $(u, v) \notin E$  then reject  
    accept"
```

► Runtime analysis:

- Checking whether $V' \subseteq V$ can be done naively in $O(|V|^2)$ via nested loop
- Since we limit $|V'| = k$ and $V' \subseteq V$, checking if $(u, v) \in E$ can be done in $O(|E|)$, so the for-loop is efficient

► Correctness Analysis

- $(G, k) \in k\text{-CLIQUE} \Rightarrow \exists V' \subseteq V$ s.t. V' is a clique of size $k \Rightarrow |V'| = k, V' \subseteq V$ and $(u, v) \in E$ for all pairs $u, v \in V' \Rightarrow F$ will accept when given this V' subset as certificate
- $(G, k) \in k\text{-CLIQUE} \Rightarrow \nexists V' \subseteq V$ s.t. V' is a clique of size $k \Rightarrow \forall V', |V'| \neq k$ or $V' \not\subseteq V$ or $\exists (u, v) \notin E \Rightarrow F$ will never accept any V' certificate

TL; DPA

- ▶ NP is the set of **efficiently verifiable languages**
- ▶ To prove that a language is in class NP
 - ▶ Give the algorithm of the efficient **verifier**
 - ▶ Correctness analysis
 - ▶ Runtime analysis (both certificate's size and verifier's runtime must be polynomial)
- ▶ Useful phrases for correctness analysis:
 - ▶ Verifier V will accept when given this c as certificate
 - ▶ Verifier V will never accept any certificate c

Take Home Exercise 2

- ▶ Let $L_1, L_2 \in NP$. Determine whether the following statements are always/ sometimes/ never true/ unknown.
 - ▶ $\overline{L_1} \in NP$
 - ▶ $L_1 \cap L_2 \in NP$
 - ▶ $L_1 \cup L_2 \in NP$
 - ▶ $L_1 \setminus L_2 \in NP$

P vs NP



P vs NP Starter

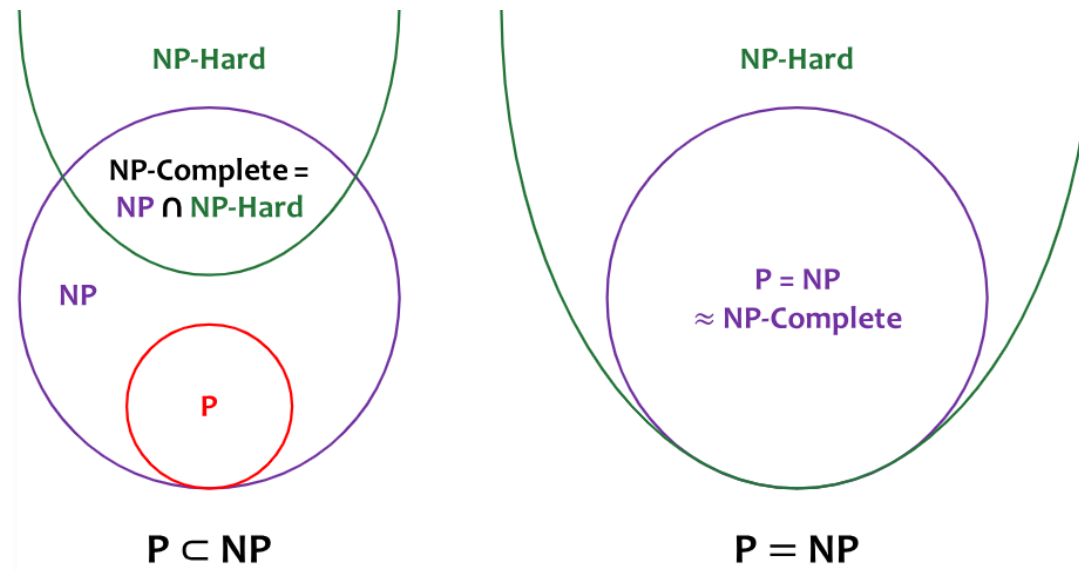
Discuss: Which of the following statements is/are true?

- A. $\forall L \in P, L \in NP$
- B. $\forall L \in NP, L \in P$
- C. $\exists L \in P: L \notin NP$
- D. $\exists L \in NP: L \notin P$

- ▶ A is true, you will prove this in HW 7 ($P \subseteq NP$)
- ▶ C is false since A is true
- ▶ B and D are unknown!
 - ▶ If B is true, then all languages in NP are in P , so $P = NP$
 - ▶ If D is true, then there exists a language in NP that is not in P , so $P \neq NP$

Relationship Between P and NP

- There is no proof as to whether $P = NP$



- If $P = NP$, then all languages in NP are NP -complete, except the two trivial languages Σ^* and \emptyset .

Our First NP-Complete Language

- ▶ We don't know if $P = NP$, but we've been able to establish **relationships between the problems within NP**
- ▶ Imagine some “hard” language in NP such that if we could decide this language efficiently, then we'd be able to decide **all** languages in NP efficiently
 - ▶ Knowledge about only this one language would give us a result for the entire class NP
 - ▶ This type of language is known as **NP-Complete**
- ▶ The language SAT is NP-Complete by the Cook Levin Theorem
 - ▶ Now if we could decide SAT efficiently, we could decide any language in NP efficiently, which would mean that $NP \subseteq P$ and thus $P=NP$!

The Language SAT

- ▶ *Literal*: Boolean variable with a value of TRUE or FALSE (such as x or $\neg x$)
- ▶ *Boolean Formula ϕ* : formula made up of literals and logical operators (such as $x \wedge \neg y$ or $x \vee y \vee z$)
- ▶ *Satisfying assignment*: a set of truth values assigned to each literal in a Boolean formula ϕ such that ϕ evaluates to true
$$SAT = \{\langle \phi \rangle : \phi \text{ is a satisfiable Boolean formula}\}$$
- ▶ Example: $x \vee \neg y \in SAT$; $x \wedge \neg x \in SAT$
- ▶ The first step in the Cook Levin proof is to show that **SAT is in NP**

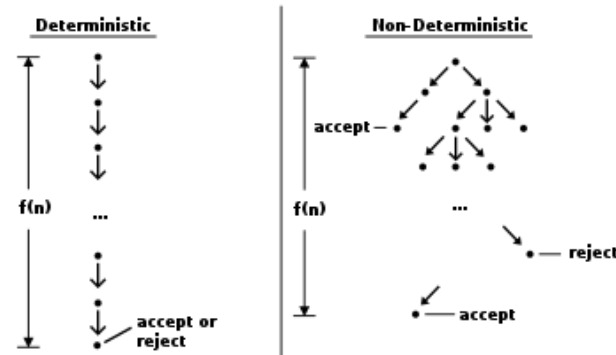
High Level Cook Levin Theorem Proof

- ▶ The meat of the Cook Levin theorem is a reduction from an arbitrary language L in NP to SAT
- ▶ This reduction takes the form of a function f that meets the following requirements:
 - $x \in L$ implies that $f(x) \in \text{SAT}$
 - $x \notin L$ implies that $f(x) \notin \text{SAT}$
 - f is computable in polynomial time
- ▶ As L is in NP, we know there is an efficient verifier for L
 - ▶ f can inspect V 's "source code" to build a corresponding (poly-sized) formula ϕ
 - ▶ ϕ is designed so that it is satisfied by some assignment of its variables *if and only if* $V(x, c)$ would accept for some c

Back Matter

Why is it called NP?

- ▶ There is an alternative but equivalent definition often used that says the class NP is the set of languages that have a **Non-deterministic Polynomial-time (efficient) decider**




- ▶ Don't worry about this definition in this class, we will only use the definition on the previous slide
- ▶ Main takeaway: All NP languages are **decidable** (Will prove $NP \subseteq EXP$ in HW7 EC)

Reddit Review

- ▶ My favorite thing to do after every EECS 376 exam
- ▶ Here, I'll share some of my favorite EECS 376 Reddit post of all time

EECS Practice Exams Aren't Real

 **r/uofm** • 3 mo. ago
legobaba

...

Proof: EECS Practice Exams Aren't Real

Meme

If (A) every EECS practice exam is a previous term's actual exam, and (B) actual EECS exams are nothing like the practice exams, then where does practice exam C, provided for a given EECS class (let's say, oh I don't know, 376) come from?

Case 1: Exam C did not come from a previous term: This trivially contradicts condition A, so Exam C must come from a previous term.

Case 2: Exam C came from a previous term: This would imply that Exam C was the actual exam for some term $n - i$. Assuming the syllabus of C's course has not changed significantly from term $n - i$ to term n , we can conclude that Exam C must be representative of the actual exam given in term n . This contradicts condition B.

Thus, we have arrived at a paradox, and the only logically sound conclusion is that practice exam C cannot exist. In light of this I preemptively request a regrade of 100% on every EECS exam I have ever or will ever take because the practice exams I was provided did not exist.

https://www.reddit.com/r/uofm/comments/1ba1hqp/proof_eecs_practice_exams_arent_real/?rdt=47458

The Devil's Advocate



LectronPusher • 1y ago • Edited 1y ago

Your post comes off as antagonistic, so I don't know if I can change your mind much by adding this, but I'd like to play devil's advocate and maybe help you view this from an educator's perspective because I know that the instructors haven't just thrown the syllabus together to "fuck your GPA". These classes have been built over many years with input from dozens of instructors who do care about students. Also, for me, understanding why instructors give homeworks and exams has helped me focus on using them correctly in future courses. I'm in an eecs class right now that only has shitty reading quizzes, and I wish we had homework like 376 that would help me learn better.

Ok, so.. devil's advocate.

I think the main thing is to get into the instructor's heads and ask why they would structure the class this way. If how the course is structured doesn't make sense for what you think their goals are, then maybe go the other way and ask what does the structure of the course imply about their goals?

Well, we have lectures, exams, and homeworks to look at. And also the fact that it's a required course to major in CS.

Lectures obviously introduce you to the concepts, but notably they're organized around specific, named, examples. Examples like SAT, the Halting problem, etc. are *named* because they want you to be able to remember them, and to refer to them when you need to apply similar concepts to problems.

Exams are different, they exist to test your knowledge of concepts, but of course they differ between types of classes, so what are the features of 376 exams? Well, they ask questions with consistent formats, but requiring creative solutions, and they also ask analytical questions that require knowing relationships between topics. Missing are questions of rote memorization of lecture content i.e. SAT is NP-hard, NP-complete, or neither. Instead, you'll more likely see a Q like: if SAT is P, what else is P?

The notable thing about these questions is that the creativity and analysis you're asked to do is truly impossible to do with just having seen and studied the lecture content. I don't think it would be possible for someone to only look at lecture and take notes and then be able to solve the exams. They require a level of doing these types of problems that infuses their patterns into your brain and helps you get that information back out when an exam comes. You've got to have done practice problems in order to solve these, and that's where homeworks, practice exams, and discussions come in.

Practice exams and discussions are obvious enough, they're just practice problems to help you learn and act as a reference, but that's an important thing to have available in a class like this because of the homeworks and exams.

Homeworks have longer problems than exams and require even more creativity and analysis to do well on. You can't solve them by memorization or looking up facts. So how can you solve them? Pattern matching and pattern extrapolation. You need to find the equivalent example from lecture or discussion and figure out how to apply it to the homework question. There won't always be something 1-to-1 with lecture, and it might be difficult to see, but most questions are solvable by extrapolating examples and looking for patterns in previous problems. They can have twists and gotchas, and I've struggled on stupid problems in 376 for hours and still got them wrong. In those cases, piazza and OH are your best bet, and where you can get feedback.

Ok, but why are they graded so harshly?

To get you to do them.

If they aren't graded and there were no deadline no one would do them. I can't defend the *amount* of work HWs are, or any harsh nitpicking by graders. But the fact is that deadlines and grades are motivations, and we need motivation in order to *do*.

I don't buy the argument that this is a GPA sponge or a "fuck your GPA" course. The university of michigan is extremely focused on its students doing well and graduating. UofM has a very small acceptance rate for a public university, but has an extremely high graduation rate (92% vs 64% national avg). Its goal is to only take in students who will succeed, and support you once you're inside. I'm not going to argue about whether this is good or bad, but that is the university's policy.

Summing up, the structure of the course requires you to do. It requires you to apply knowledge. Not just recall facts, but construct analysis and proofs. And it tries to give you motivation in the form of grades and deadlines.

So this is not a course on learning the theory side of computer science, it's a course on doing the theory side of computer science. The fact that it's required for the major says that the CS department expects their graduates to be able to do this.

That's the theory anyways. I can't defend difficulty or decisions about curving exams. But the structure of combining homeworks and exams in the way 376 does is explicit and intentional, and it's designed for learning.

https://www.reddit.com/r/uofm/comments/125wcmk/im_sorry_but_holy_shit_why_is_eecs_376_a_required/