BU CS 332 – Theory of Computation

https://forms.gle/muSXckPgw2Rw97rz6

Lecture 23:

- NP-completeness example
- Space complexity

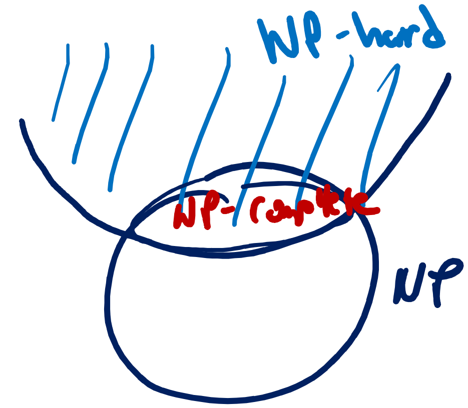
Reading:

Sipser Ch 7.4-7.5,
8.1-8.2

Alexander Poremba & Mark Bun

April 28, 2026

NP-completeness review



Definition: A language B is NP-complete if

1) $B \in \mathrm{NP}$, and
2) Every language $A \in \mathrm{NP}$ is poly-time reducible to B, i.e., $A \leq_{\mathrm{p}} B$ ("B is NP-hard")

If $A \leq_p B$ and $B \in NP$ then $A \in NP$

The usual way to prove NP-completeness:

If

1) $C \in \mathrm{NP}$, and
2) There is an NP-complete language B (e.g., 3-SAT, VERTEX-COVER, IND-SET, …) such that $B \leq_{\mathrm{p}} C$

then C is NP-complete.

Subset Sum

SUBSET-SUM = $\{\langle w_1, \ldots, w_m, t\rangle \mid$
there exists a subset of natural numbers $w_1, \ldots, w_m$ that sum to $t\}$

Ex. $\langle 1, 3, 4, 8, 15\rangle \in$ SUBSET-SUM

because $3+4+8 = 15$

Theorem: SUBSET-SUM is NP-complete

Claim 1: SUBSET-SUM is in NP

NTM:

On input $\langle w_1, \ldots, w_m, t\rangle$

1. Nondeterministically guess $S \subseteq [m]$
2. Accept iff $\sum_{i \in S} w_i = t$

Claim 2: SUBSET-SUM is NP-hard

Prove by reducing 3-SAT $\leq_p$ SUBSET-SUM

$3SAT \leq_p$ SUBSET−SUM

Perform this in (deterministic) poly-time

Goal: Given a 3CNF formula φ on v variables and k clauses, construct a SUBSET-SUM instance $w_1, \dots, w_m, t$ such that

φ is satisfiable iff there exists a subset of $w_1, \dots, w_m$ that sum to t

$x_1, \overline{x_1}, x_2, \overline{x_2}, \dots$

First attempt: Encode each literal ℓ of φ as a k-digit *decimal* number $w_\ell = c_1 \dots c_k$ where

$$c_i = \begin{cases} 1 \text{ if } \ell \text{ appears in clause } i \\ 0 \text{ otherwise} \end{cases}$$

Example first attempt reduction

$$\varphi = \underbrace{(\overline{x_1} \vee x_2 \vee x_3)}^{\text{Clause 1}} \wedge \underbrace{(x_1 \vee \overline{x_2} \vee x_3)}^{\text{Clause 2}}$$

ℓ	$w_\ell = c_1$	c_2
$\boxed{x_1}$	0	1
$\overline{x_1}$	1	0
$\boxed{x_2}$	1	0
$\overline{x_2}$	0	1
$\boxed{x_3}$	1	1
$\overline{x_3}$	0	0

Encode literal ℓ as $w_\ell = c_1 \dots c_k$ where

$$c_i = \begin{cases} 1 \text{ if } \ell \text{ appears in clause } i \\ 0 \text{ otherwise} \end{cases}$$

Ex. Satisfying assignment to φ

$x_1 = 1$
$x_2 = 1$
$x_3 = 1$

$$\begin{array}{r} 01 \\ 10 \\ +\ 11 \\ \hline 22 \end{array}$$

"digitwise"

$3 \quad 11$

$3\text{SAT} \leq_p$ SUBSET–SUM

First attempt: Encode each literal ℓ of φ as a k-digit *decimal* number $w_\ell = c_1 \dots c_k$ where

$$c_i = \begin{cases} 1 \text{ if } \ell \text{ appears in clause } i \\ \qquad 0 \text{ otherwise} \end{cases}$$

Claim: If φ is satisfiable, then there exists a subset of the w_ℓ's that "digit-wise" add up to "at least" $111 \dots 11$

<u>Two issues:</u>

1) Need to enforce that exactly one of $\ell, \overline{\ell}$ is set to 1
2) Need the subset to add up to **exactly** some target

$3\text{SAT} \leq_p$ SUBSET−SUM

Actual reduction: Encode each literal ℓ of φ as a $(v+k)$-digit *decimal* number $w_\ell = b_1 \dots b_v |\ c_1 \dots c_k$ where

$$b_i = \begin{cases} 1 \text{ if } \ell \in \{x_i, \overline{x_i}\} \\ 0 \text{ otherwise} \end{cases} \qquad c_i = \begin{cases} 1 \text{ if } \ell \text{ appears in clause } i \\ 0 \text{ otherwise} \end{cases}$$

Enforce choosing exactly one of each ℓ or $\overline{\ell}$

Also, include two copies each of 000…0|100…0, 000…0|010…0, … 000…0|0…01

Ask for exactly summing to a target

Claim: φ is satisfiable if and only if there exists a subset of the numbers that add up to $t = 111 \dots 11|333 \dots 33$

Example of the reduction

$\varphi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3)$

Encode literal ℓ as $w_\ell = b_1 \dots b_v | \, c_1 \dots c_k$ where

$$b_i = \begin{cases} 1 \text{ if } \ell \in \{x_i, \overline{x_i}\} \\ 0 \text{ otherwise} \end{cases}$$

$$c_i = \begin{cases} 1 \text{ if } \ell \text{ appears in clause } i \\ 0 \text{ otherwise} \end{cases}$$

Include two copies each of 000…0|100…0, 000…0|010…0, … 000…0|0…01

"Padding" w's

p_{11} = 0 0 0 1 0
p_{12} = 0 0 0 1 0
clause 1

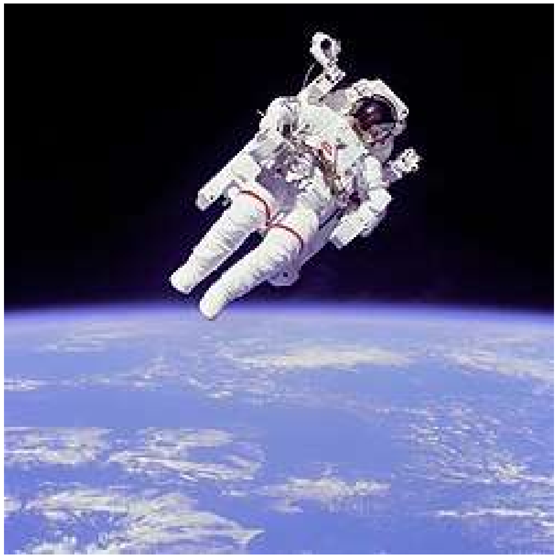
p_{21} = 0 0 0 0 1
p_{22} = 0 0 0 0 1
clause 2

ℓ	w_ℓ = b_1	b_2	b_3	c_1	c_2
x_1	1	0	0	0	1
$\overline{x_1}$	1	0	0	1	0
x_2	0	1	0	1	0
$\overline{x_2}$	0	1	0	0	1
x_3	0	0	1	1	1
$\overline{x_3}$	0	0	1	0	0

Final SUBSET-SUM Instance:
⟨10001, 10010, …, 00100, 00010, …, 00001, 11133⟩

Sat. Assignment: $x_1 = 1$ $x_2 = 1$ $x_3 = 1$

A Brief Tour of Space (Complexity)



Space analysis

Space complexity of a TM (algorithm) = maximum number of tape cells it uses on a worst-case input

Formally: Let $f : \mathbb{N} \to \mathbb{N}$. A TM M runs in space $f(n)$ if on every input $w \in \Sigma^*$, M halts on w using at most $f(|w|)$ cells

For nondeterministic machines: Let $f : \mathbb{N} \to \mathbb{N}$. An NTM N runs in space $f(n)$ if on every input $w \in \Sigma^*$, N halts on w using at most $f(|w|)$ cells on every computational branch

Space complexity classes

Let $f : \mathbb{N} \to \mathbb{N}$

A language $A \in \mathrm{SPACE}(f(n))$ if there exists a basic single-tape (deterministic) TM M that

1) Decides A, and
2) Runs in space $O(f(n))$

A language $A \in \mathrm{NSPACE}(f(n))$ if there exists a single-tape nondeterministic TM N that

1) Decides A, and
2) Runs in space $O(f(n))$

Example: Space complexity of SAT

Theorem: $SAT \in \text{SPACE}(n)$

Proof: The following deterministic TM decides SAT using linear space

On input $\langle \varphi \rangle$ where φ is a Boolean formula:

1. For each truth assignment to the variables $x_1, \dots, x_m$ of φ:
2. Evaluate φ on $x_1, \dots, x_m$
3. If any evaluation $= 1$, accept. Else, reject.

Runs in time $2^{O(m)}$ but can reuse the same workspace to evaluate each assignment

Space vs. Time

$$\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$$
$$\subseteq \mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$$

Simulating Time With Square-Root Space*

Ryan Williams†
MIT

Abstract

We show that for all functions $t(n) \geq n$, every multitape Turing machine running in time t can be simulated in space only $O(\sqrt{t \log t})$. This is a substantial improvement over Hopcroft, Paul, and Valiant's simulation of time t in $O(t/\log t)$ space from 50 years ago [FOCS 1975, JACM 1977].

$\mathrm{TIME}(f(n)) \subseteq$

$\mathrm{SPACE}(\sqrt{f(n) \log f(n)})$

[defining everything for multi-tape TMs]

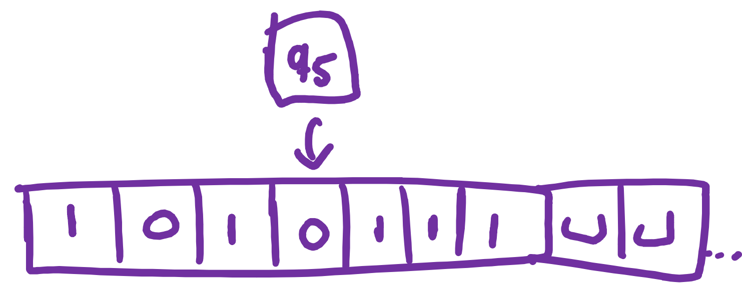
How about the opposite direction? Can low-space algorithms be simulated by low-time algorithms?

Reminder: Configurations

A configuration is a string uqv where $q \in Q$ and $u, v \in \Gamma^*$

- Tape contents = uv (followed by blanks $\sqcup$)
- Current state = q
- Tape head on first symbol of v

Example: $101q_5 0111$



Start configuration: $q_0 w$

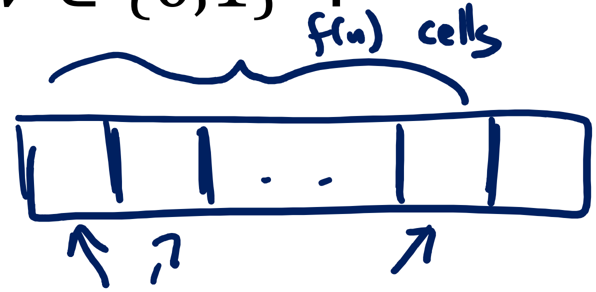
Accepting configuration: $q = q_{\text{accept}}$

Rejecting configuration: $q = q_{\text{reje}}$

Reminder: Configurations

Consider a TM with k states, tape alphabet $\{0, 1, \sqcup\}$, and space bound $f(n)$. How many configurations are possible when this TM is run on an input $w \in \{0,1\}^n$?

Tape contents:



each either 0, 1, ⊔

$3^{f(n)}$

a) $3k \cdot f(n)$

b) $k + f(n)$

c) $k \cdot 3^{f(n)}$

d) $\boxed{k \cdot f(n) \cdot 3^{f(n)}}$

$= 2^{O(f(n))}$

State: k choices

Head location: f(n) choices

Observation: If a TM enters the same configuration twice when run on input w, it loops forever

Corollary: A TM running in space $f(n)$ also runs in time $2^{O(f(n))}$

Space vs. Time

$$\begin{aligned}\mathrm{TIME}(f(n)) &\subseteq \mathrm{NTIME}(f(n)) \\ &\subseteq \mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n)) \\ &\subseteq \mathrm{TIME}\left(2^{O(f(n))}\right)\end{aligned}$$

Now how about the relationship between SPACE and NSPACE?

Savitch's Theorem: Deterministic vs. Nondeterministic Space

~1970

Theorem: Let f be a function with $f(n) \geq n$. Then $NSPACE(f(n)) \subseteq SPACE\left((f(n))^2\right)$.

Proof idea:

- Let N be an NTM deciding A in space $f(n)$
- We construct a deterministic TM M deciding A in space $O\left((f(n))^2\right)$
- Actually solve a more general problem:
  - Given configurations c_1, c_2 of N and natural number t, decide whether N can go from configuration c_1 to c_2 in $\leq t$ steps on some nondeterministic path
  - Procedure called CANYIELD(c_1, c_2, t)

Savitch's Theorem

Theorem: Let f be a function with $f(n) \geq n$. Then $NSPACE(f(n)) \subseteq SPACE\left((f(n))^2\right)$.

Proof idea:

- Let N be an NTM deciding A in space $f(n)$

M = "On input w:

Start config of N Accept config of N

1. Output the result of $\underbrace{\text{CANYIELD}(c_1, c_2, 2^{Cf(n)})}_{\text{uses deterministic space}}$"

$O(f(n))^2)$

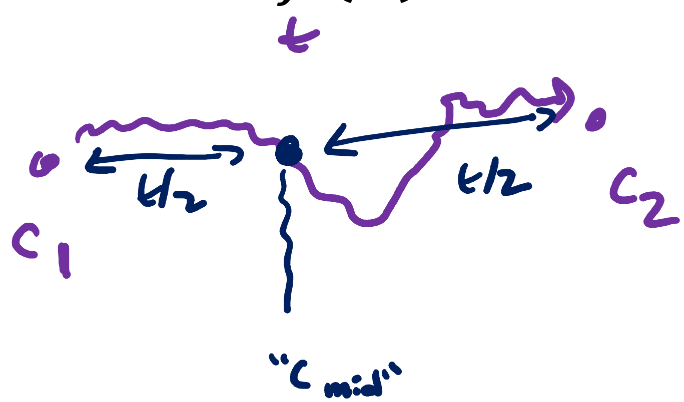
where CANYIELD(c_1, c_2, t) decides whether N can go from configuration c_1 to c_2 in $\leq t$ steps on some nondeterministic path

Savitch's Theorem

CANYIELD(c_1, c_2, t) decides whether N can go from configuration c_1 to c_2 in $\leq t$ steps on some nondeterministic path:

CANYIELD(c_1, c_2, t) =

1. If $t = 1$, accept if $c_1 = c_2$ or c_1 yields c_2 in one transition.
   Else, reject.
2. If $t > 1$, then for each config c_{mid} of N with $\leq f(n)$ cells:
3. Run CANYIELD$(c_1,\ c_{mid},\ t/2)$.
4. Run CANYIELD$(c_{mid},\ c_2,\ t/2)$.
5. If both runs accept, accept.
6. Reject.



$S(t) :=$ space usage for step count t

$= S(t/2) + O(f(n))$ ← keep track of c_{mid}

$\approx \log t \cdot O(f(n))$

Complexity class PSPACE

Definition: PSPACE is the class of languages decidable in polynomial space on a basic single-tape (deterministic) TM

$$\text{PSPACE} = \bigcup_{k=1}^{\infty} \text{SPACE}(n^k)$$

Definition: NPSPACE is the class of languages decidable in polynomial space on a single-tape (nondeterministic) TM

$$\text{NPSPACE} = \bigcup_{k=1}^{\infty} \text{NSPACE}(n^k)$$

PSPACE = NSPACE by Savitch's Thm

Relationships between complexity classes

1. $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$

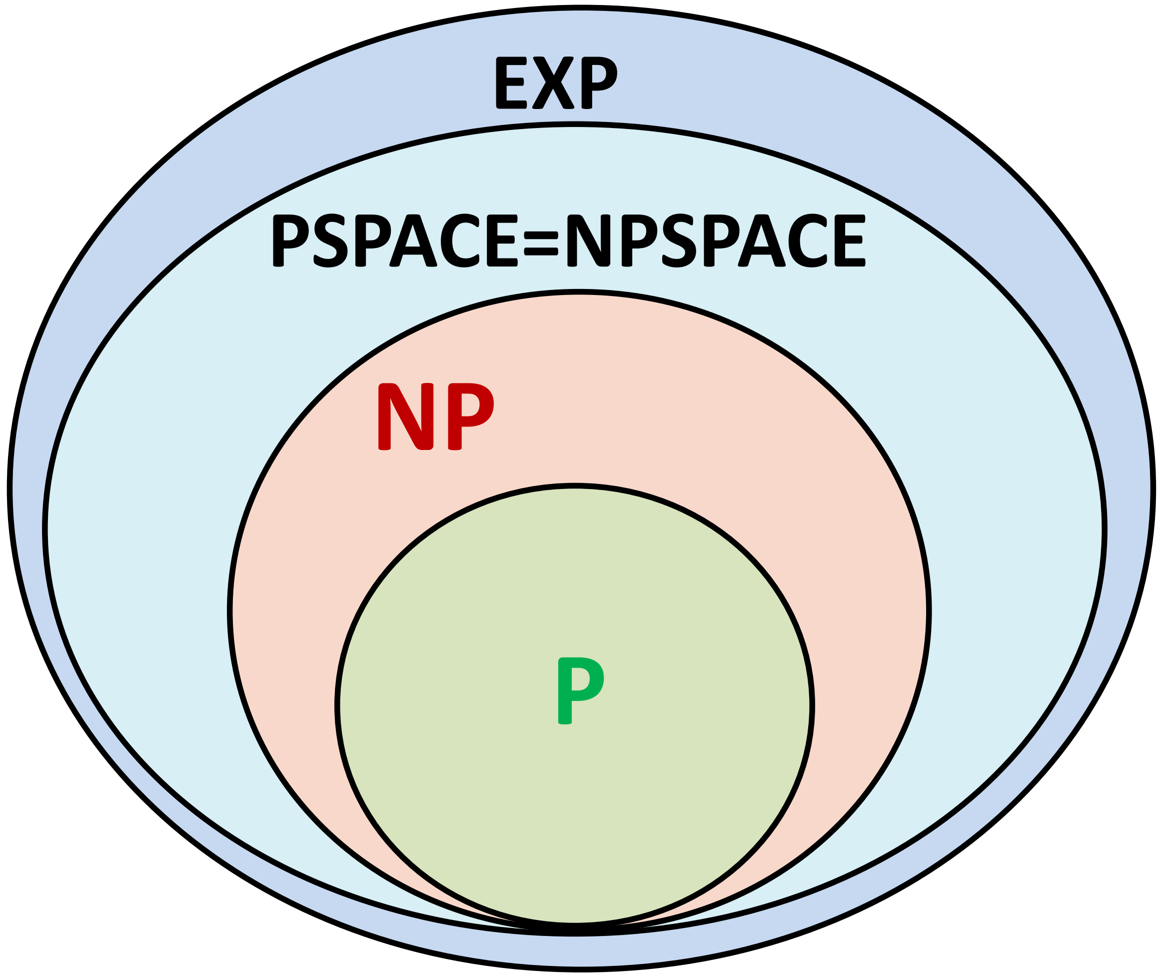
since $\mathrm{SPACE}\big(f(n)\big) \subseteq \mathrm{TIME}(2^{O(f(n))})$

2. $\mathrm{P} \neq \mathrm{EXP}$

   (via time hierarchy)

Which containments in (1) are proper?

Unknown!



Things we didn't get to talk about

- Details of polynomial and logarithmic space
- Alternating TMs, polynomial hierarchy
- Relativization and the limits of diagonalization
- Boolean circuits
- Randomized algorithms / complexity classes
- Interactive and probabilistic proof systems
- Complexity of counting

For all of this and more, Mark is teaching CS 535 in Fall '26
https://cs-people.bu.edu/mbun/courses/535_F23

Course Evaluations

bu.edu/courseeval/