Computational Complexity Theory

Fall 2025

Finishing the Time-space lower bounds for SAT and Savitch's Theorem

September 11, 2025

Lijie Chen

University of California, Berkeley

✉ lijiechen@berkeley.edu

The Speedup Lemma

$$\mathrm{DTS}[n^c] = \mathrm{TISP}[n^c, n^{o(1)}].$$

Lemma (Speedup Lemma)

$$DTS[n^d] \subseteq (\exists n^x)(\forall \log n)DTS[n^{d-x}].$$
$$DTS[n^d] \subseteq (\forall n^x)(\exists \log n)DTS[n^{d-x}].$$

Definition

A language $L \in (\exists f(n))(\forall g(n))\mathrm{DTS}[n^k]$ if there is an $n^{o(1)}$ space machine M such that for all $x \in \{0,1\}^n$,

- $x \in L$ if and only if there exists a witness w with $|w| = f(n)^{1+o(1)}$ such that for every y with $|y| = g(n)^{1+o(1)}$, $M(x, w, y)$ accepts in $n^{k+o(1)}$ time.

Proof of the Speedup Lemma: see the white board!

The Slowdown Lemma

$\mathrm{DTS}[n^c] = \mathrm{TISP}[n^c, n^{o(1)}]$.

Lemma (Slowdown Lemma)

If $\mathit{NTIME}[n] \subseteq \mathit{DTS}[n^c]$, then $\Sigma_2\mathit{TIME}[n^d] \subseteq \mathit{NTIME}[n^{d \cdot c + o(1)}]$.

Proof: see the white board!

The Padding Lemma

Lemma (Padding Lemma)

If $NTIME[n] \subseteq DTS[n^c]$, then $NTIME[n^d] \subseteq DTS[n^{d \cdot c}]$.

Proof: see the white board!

Main theorem: The ϕ Lower Bound

Theorem

$NTIME[n] \not\subseteq DTS[n^{\phi-\epsilon}]$ *for any* $\epsilon > 0$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$.

Lemma

$DTS[n^a] \not\subseteq DTS[n^b]$ *for any* $a > b > 1$.

Main theorem: The ϕ Lower Bound

Theorem

NTIME$[n] \not\subseteq$ *DTS*$[n^{\phi-\epsilon}]$ *for any* $\epsilon > 0$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$.

- Again, Proof by contradiction.

Lemma

DTS$[n^a] \not\subseteq$ *DTS*$[n^b]$ *for any* $a > b > 1$.

Main theorem: The ϕ Lower Bound

Theorem

$NTIME[n] \not\subseteq DTS[n^{\phi-\epsilon}]$ for any $\epsilon > 0$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$.

- Again, Proof by contradiction.
- Assume $\mathrm{NTIME}[n] \subseteq \mathrm{DTS}[n^{\phi-\epsilon}]$, we will deduce $\mathrm{DTS}[n^a] \subseteq \mathrm{DTS}[n^b]$ for some $a > b > 1$, this is a contradiction.

Lemma

$DTS[n^a] \not\subseteq DTS[n^b]$ for any $a > b > 1$.

Main theorem: The ϕ Lower Bound

Theorem

NTIME$[n] \not\subseteq$ *DTS*$[n^{\phi-\epsilon}]$ *for any* $\epsilon > 0$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$.

Key lemma:

Lemma

For all $k \geqslant 0$, *if* *NTIME*$[n] \subseteq$ *DTS*$[n^c]$, *then*

$$DTS\left[n^{2+\sum_{i=1}^{k} c^i}\right] \subseteq \Sigma_2 TIME\left[n^{c^k+o(1)}\right]$$

and

$$DTS\left[n^{2+\sum_{i=1}^{k} c^i}\right] \subseteq \Pi_2 TIME\left[n^{c^k+o(1)}\right]$$

Proof: see the white board!

Today's plan

- Deterministic and nondeterministic **space** complexity

Today's plan

- Deterministic and nondeterministic **space** complexity
- Defining $\mathrm{SPACE}(f(n))$ and $\mathrm{NSPACE}(f(n))$

Today's plan

- Deterministic and nondeterministic **space** complexity
- Defining $\text{SPACE}(f(n))$ and $\text{NSPACE}(f(n))$
- Savitch's theorem: "P = NP" for space!

Today's plan

- Deterministic and nondeterministic **space** complexity
- Defining $\text{SPACE}(f(n))$ and $\text{NSPACE}(f(n))$
- Savitch's theorem: "P = NP" for space!
- A proof overview of Savitch's theorem.

Today's plan

- Deterministic and nondeterministic **space** complexity
- Defining $\text{SPACE}(f(n))$ and $\text{NSPACE}(f(n))$
- Savitch's theorem: "P = NP" for space!
- A proof overview of Savitch's theorem.
- Corollaries: $\mathbf{PSPACE} = \mathbf{NPSPACE}$ and $\mathbf{NL} \subseteq \text{SPACE}(\log^2 n)$

Today's plan

- Deterministic and nondeterministic **space** complexity
- Defining SPACE$(f(n))$ and NSPACE$(f(n))$
- Savitch's theorem: "P = NP" for space!
- A proof overview of Savitch's theorem.
- Corollaries: **PSPACE** $=$ **NPSPACE** and **NL** $\subseteq$ SPACE$(\log^2 n)$
- PSPACE-completeness and TQBF (if time permits)

Quick recap: Deterministic space complexity

- For a multi-tape Turing machine M and input x, the **space complexity** of M on x is the number of tape cells visited by any head during the computation.

Quick recap: Deterministic space complexity

- For a multi-tape Turing machine M and input x, the **space complexity** of M on x is the number of tape cells visited by any head during the computation.
- We say M uses space $S(n)$ if for every input x of length n, M uses at most $S(n)$ space.

Quick recap: Deterministic space complexity

- For a multi-tape Turing machine M and input x, the **space complexity** of M on x is the number of tape cells visited by any head during the computation.
- We say M uses space $S(n)$ if for every input x of length n, M uses at most $S(n)$ space.
- The input tape is **read-only**,

Quick recap: Deterministic space complexity

- For a multi-tape Turing machine M and input x, the **space complexity** of M on x is the number of tape cells visited by any head during the computation.
- We say M uses space $S(n)$ if for every input x of length n, M uses at most $S(n)$ space.
- The input tape is **read-only**,
- There is a designated **output tape** which is **write-only**.

Quick recap: Deterministic space complexity

- For a multi-tape Turing machine M and input x, the **space complexity** of M on x is the number of tape cells visited by any head during the computation.
- We say M uses space $S(n)$ if for every input x of length n, M uses at most $S(n)$ space.
- The input tape is **read-only**,
- There is a designated **output tape** which is **write-only**.
- Only the **work tapes** count towards space complexity.

Quick recap: Deterministic space complexity

- For a multi-tape Turing machine M and input x, the **space complexity** of M on x is the number of tape cells visited by any head during the computation.
- We say M uses space $S(n)$ if for every input x of length n, M uses at most $S(n)$ space.
- The input tape is **read-only**,
- There is a designated **output tape** which is **write-only**.
- Only the **work tapes** count towards space complexity.
- $\mathrm{SPACE}(s(n))$: set of languages that can be decided by a deterministic multi-tape Turing machine using $O(s(n))$ space.

Quick recap: Deterministic space complexity

- For a multi-tape Turing machine M and input x, the **space complexity** of M on x is the number of tape cells visited by any head during the computation.
- We say M uses space $S(n)$ if for every input x of length n, M uses at most $S(n)$ space.
- The input tape is **read-only**,
- There is a designated **output tape** which is **write-only**.
- Only the **work tapes** count towards space complexity.
- $\mathrm{SPACE}(s(n))$: set of languages that can be decided by a deterministic multi-tape Turing machine using $O(s(n))$ space.
- $\mathbf{L} = \mathrm{SPACE}(\log n)$ and $\mathbf{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathrm{SPACE}(n^k)$.

Recap: Multi-tape non-deterministic TM

Definition (Multi-tape non-deterministic Turing machine)

- A *multi-tape non-deterministic Turing machine* is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:

Recap: Multi-tape non-deterministic TM

Definition (Multi-tape non-deterministic Turing machine)

- A *multi-tape non-deterministic Turing machine* is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:
  - Q is a finite set of states

Recap: Multi-tape non-deterministic TM

Definition (Multi-tape non-deterministic Turing machine)

- A *multi-tape non-deterministic Turing machine* is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:
  - Q is a finite set of states
  - Σ is the input alphabet

Recap: Multi-tape non-deterministic TM

Definition (Multi-tape non-deterministic Turing machine)

- A *multi-tape non-deterministic Turing machine* is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:
  - Q is a finite set of states
  - Σ is the input alphabet
  - Γ is the tape alphabet (with $\Sigma \subseteq \Gamma$)

Recap: Multi-tape non-deterministic TM

Definition (Multi-tape non-deterministic Turing machine)

- A *multi-tape non-deterministic Turing machine* is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:
  - Q is a finite set of states
  - Σ is the input alphabet
  - Γ is the tape alphabet (with $\Sigma \subseteq \Gamma$)
  - $\delta : Q \times \Gamma^k \to \mathcal{P}(Q \times \Gamma^k \times \{L, R, S\}^k)$ is the transition function

Recap: Multi-tape non-deterministic TM

Definition (Multi-tape non-deterministic Turing machine)

- A *multi-tape non-deterministic Turing machine* is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:
  - Q is a finite set of states
  - Σ is the input alphabet
  - Γ is the tape alphabet (with $\Sigma \subseteq \Gamma$)
  - $\delta : Q \times \Gamma^k \to \mathcal{P}(Q \times \Gamma^k \times \{L, R, S\}^k)$ is the transition function
  - $q_0 \in Q$ is the initial state

Recap: Multi-tape non-deterministic TM

Definition (Multi-tape non-deterministic Turing machine)

- A *multi-tape non-deterministic Turing machine* is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:
  - Q is a finite set of states
  - Σ is the input alphabet
  - Γ is the tape alphabet (with $\Sigma \subseteq \Gamma$)
  - $\delta : Q \times \Gamma^k \to \mathcal{P}(Q \times \Gamma^k \times \{L, R, S\}^k)$ is the transition function
  - $q_0 \in Q$ is the initial state
  - $q_{acc}, q_{rej} \in Q$ are the accepting and rejecting states

Recap: Multi-tape non-deterministic TM

Definition (Multi-tape non-deterministic Turing machine)

- A *multi-tape non-deterministic Turing machine* is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:
  - Q is a finite set of states
  - Σ is the input alphabet
  - Γ is the tape alphabet (with $\Sigma \subseteq \Gamma$)
  - $\delta : Q \times \Gamma^k \to \mathcal{P}(Q \times \Gamma^k \times \{L, R, S\}^k)$ is the transition function
  - $q_0 \in Q$ is the initial state
  - $q_{acc}, q_{rej} \in Q$ are the accepting and rejecting states

- M accepts x **if and only if** there is a sequence of transitions that leads to q_{acc}.

Non-deterministic space complexity

Definition (Non-deterministic space complexity)

- For a (non-deterministic) multi-tape Turing machine M and input x, the **space complexity** of M on x is the maximum number of tape cells visited by any head during the computation, over all possible sequences of transitions.

Non-deterministic space complexity

Definition (Non-deterministic space complexity)

- For a (non-deterministic) multi-tape Turing machine M and input x, the **space complexity** of M on x is the maximum number of tape cells visited by any head during the computation, over all possible sequences of transitions.
- We say M uses space $S(n)$ if for every input x of length n, M uses at most $S(n)$ space.

Non-deterministic space complexity

Definition (Non-deterministic space complexity)

- For a (non-deterministic) multi-tape Turing machine M and input x, the **space complexity** of M on x is the maximum number of tape cells visited by any head during the computation, over all possible sequences of transitions.
- We say M uses space $S(n)$ if for every input x of length n, M uses at most $S(n)$ space.
- The input tape is **read-only** and does not count towards space usage.

Non-deterministic space complexity

Definition (Non-deterministic space complexity)

- For a (non-deterministic) multi-tape Turing machine M and input x, the **space complexity** of M on x is the maximum number of tape cells visited by any head during the computation, over all possible sequences of transitions.
- We say M uses space $S(n)$ if for every input x of length n, M uses at most $S(n)$ space.
- The input tape is **read-only** and does not count towards space usage.
- There is a designated **output tape** which is **write-only** and does not count towards space usage.

Non-deterministic space complexity

Definition (Non-deterministic space complexity)

- For a (non-deterministic) multi-tape Turing machine M and input x, the **space complexity** of M on x is the maximum number of tape cells visited by any head during the computation, over all possible sequences of transitions.
- We say M uses space $S(n)$ if for every input x of length n, M uses at most $S(n)$ space.
- The input tape is **read-only** and does not count towards space usage.
- There is a designated **output tape** which is **write-only** and does not count towards space usage.
- Only the **work tapes** count towards space complexity.

Space vs. Nondeterministic space

NSPACE$(s(n))$: set of languages that can be decided by a non-deterministic multi-tape Turing machine using $O(s(n))$ space.

$\mathbf{NL} = \text{NSPACE}(\log n)$ and $\mathbf{NPSPACE} = \bigcup_{k\in\mathbb{N}} \text{NSPACE}(n^k)$.

Basic relationships

- $\text{SPACE}(s(n)) \subseteq \text{NSPACE}(s(n))$ (determinism is a special case).

Space vs. Nondeterministic space

NSPACE$(s(n))$: set of languages that can be decided by a non-deterministic multi-tape Turing machine using $O(s(n))$ space.

NL $=$ NSPACE$(\log n)$ and **NPSPACE** $= \bigcup_{k \in \mathbb{N}}$ NSPACE(n^k).

Basic relationships

- SPACE$(s(n)) \subseteq$ NSPACE$(s(n))$ (determinism is a special case).
- NSPACE$(s(n)) \subseteq$ TIME$(2^{O(s(n))})$.

Space vs. Nondeterministic space

NSPACE$(s(n))$: set of languages that can be decided by a non-deterministic multi-tape Turing machine using $O(s(n))$ space.

NL $= \text{NSPACE}(\log n)$ and **NPSPACE** $= \bigcup_{k \in \mathbb{N}} \text{NSPACE}(n^k)$.

Basic relationships

- $\text{SPACE}(s(n)) \subseteq \text{NSPACE}(s(n))$ (determinism is a special case).
- $\text{NSPACE}(s(n)) \subseteq \text{TIME}(2^{O(s(n))})$.
- $\text{SPACE}(s(n)) = \text{coSPACE}(s(n))$.

Space vs. Nondeterministic space

NSPACE$(s(n))$: set of languages that can be decided by a non-deterministic multi-tape Turing machine using $O(s(n))$ space.

NL $=$ NSPACE$(\log n)$ and **NPSPACE** $= \bigcup_{k\in\mathbb{N}}$ NSPACE(n^k).

Basic relationships

- SPACE$(s(n)) \subseteq$ NSPACE$(s(n))$ (determinism is a special case).
- NSPACE$(s(n)) \subseteq$ TIME$(2^{O(s(n))})$.
- SPACE$(s(n)) =$ coSPACE$(s(n))$.
- Surprisingly: NSPACE$(s(n)) =$ coNSPACE$(s(n))$ (next lecture).

Why Savitch's theorem is interesting

- In time complexity, nondeterminism *seems* powerful (**P** vs. **NP**).

Why Savitch's theorem is interesting

- In time complexity, nondeterminism *seems* powerful (**P** vs. **NP**).
- **Savitch (1970):** for space, nondeterminism is much less powerful:

$$\mathrm{NSPACE}(s(n)) \subseteq \mathrm{SPACE}(s(n)^2) \qquad \text{for } s(n) \geqslant \log n.$$

Why Savitch's theorem is interesting

- In time complexity, nondeterminism *seems* powerful (**P** vs. **NP**).
- **Savitch (1970):** for space, nondeterminism is much less powerful:

$$\mathrm{NSPACE}(s(n)) \subseteq \mathrm{SPACE}(s(n)^2) \qquad \text{for } s(n) \geqslant \log n.$$

- Immediate corollary: **PSPACE** = **NPSPACE**.

Why Savitch's theorem is interesting

- In time complexity, nondeterminism *seems* powerful (**P** vs. **NP**).
- **Savitch (1970):** for space, nondeterminism is much less powerful:

$$\mathrm{NSPACE}(s(n)) \subseteq \mathrm{SPACE}(s(n)^2) \qquad \text{for } s(n) \geqslant \log n.$$

- Immediate corollary: **PSPACE** $=$ **NPSPACE**.
- Another corollary: **NL** $\subseteq \mathrm{SPACE}(\log^2 n)$. The **L** vs. **NL** question remains open.

Why Savitch's theorem is interesting

- In time complexity, nondeterminism *seems* powerful (**P** vs. **NP**).
- **Savitch (1970):** for space, nondeterminism is much less powerful:

$$\mathrm{NSPACE}(s(n)) \subseteq \mathrm{SPACE}(s(n)^2) \qquad \text{for } s(n) \geqslant \log n.$$

- Immediate corollary: **PSPACE** $=$ **NPSPACE**.
- Another corollary: **NL** $\subseteq \mathrm{SPACE}(\log^2 n)$. The **L** vs. **NL** question remains open.
- The proof is a clean *divide–and–conquer on paths in a configuration graph*, reusing space via recursion.

Configuration graph

- For a fixed machine M and input x, a *configuration* encodes the state, heads, and work-tape contents.

Configuration graph

- For a fixed machine M and input x, a *configuration* encodes the state, heads, and work-tape contents.
- If M uses $s(n)$ space, the number of distinct configurations is

$$N = 2^{O(s(n))}.$$

Configuration graph

- For a fixed machine M and input x, a *configuration* encodes the state, heads, and work-tape contents.
- If M uses $s(n)$ space, the number of distinct configurations is

$$N = 2^{O(s(n))}.$$

- Build the directed graph $G_{M,x}$ whose nodes are configurations and whose edges represent one valid move.

Configuration graph

- For a fixed machine M and input x, a *configuration* encodes the state, heads, and work-tape contents.
- If M uses $s(n)$ space, the number of distinct configurations is

$$N = 2^{O(s(n))}.$$

- Build the directed graph $G_{M,x}$ whose nodes are configurations and whose edges represent one valid move.
- M accepts x iff there exists a path from the start configuration c_{start} to some c_{acc}.

Configuration graph

- For a fixed machine M and input x, a *configuration* encodes the state, heads, and work-tape contents.
- If M uses $s(n)$ space, the number of distinct configurations is

$$N = 2^{O(s(n))}.$$

- Build the directed graph $G_{M,x}$ whose nodes are configurations and whose edges represent one valid move.
- M accepts x iff there exists a path from the start configuration c_{start} to some c_{acc}.
- Any accepting path has length at most N (no need to repeat configurations).

Savitch's theorem (statement)

Theorem (Savitch)

For $s(n) \geqslant \log n$,

$$NSPACE(s(n)) \subseteq SPACE(s(n)^2).$$

Intuition

Compute the function $\text{Reach}(u, v, t)$ that decides if there is a path from u to v of length at most t in $G_{M,x}$ in $O(s(n)^2)$ space.

Proof: see the white board!

PSPACE and PSPACE-completeness (recap)

Recall: Many-one reduction

A language A many-one reduces to B (written $A \leqslant_m^p B$) if there exists a poly-time computable function f such that

$$x \in A \iff f(x) \in B \quad \text{for all } x.$$

PSPACE-hard / complete

A language L is PSPACE-hard if every $A \in$ PSPACE many-one reduces to L in polytime.
L is PSPACE-complete if $L \in$ PSPACE and L is PSPACE-hard.

TQBF (a.k.a. QSAT)

Problem

TQBF = the set of *true*, fully-quantified Boolean formulas.
Instance: a closed formula $Q_1 x_1\, Q_2 x_2 \cdots Q_m x_m \,.\, \varphi(x_1, \ldots, x_m)$ with $Q_i \in \{\forall, \exists\}$ and propositional φ.
Question: is the formula *true* under the standard semantics of quantifiers?

Example

$\forall x\, \exists y\, \forall z.\ (x \lor y) \land (\neg y \lor z)$ is true: for each x, pick $y = 1$; then for all z the matrix holds.

Why it matters

TQBF is the *canonical* PSPACE-complete problem (the “SAT” of PSPACE).

TQBF $\in$ PSPACE

Depth-first evaluation uses only polynomial space

Evaluate the prefix left-to-right with a recursive procedure that reuses space:

- For Qx at the front, branch on $x \in \{0, 1\}$ and recurse on the shorter prefix.
- On an $\exists$, accept if *some* branch accepts; on a $\forall$, accept if *all* branches accept.
- Stop at matrix φ and evaluate it in polytime.

The recursion depth is the number of variables m, so total space is $O(m + |\varphi|)$ = polynomial; time may be exponential.

TQBF is PSPACE-hard (proof sketch)

From any $A \in$ PSPACE to TQBF

Let M be a poly-space TM deciding A. For input x, consider the *configuration graph* $G_{M,x}$ whose nodes are configurations; its size is $2^{p(|x|)}$ for some polynomial p.

Proof: see the white board!

A handy template: reductions from TQBF

Game/constraint viewpoint

Evaluate a QBF as a two-player, perfect-information game with moves for $\exists$ (Eve) and $\forall$ (Adam). The formula is true iff Eve has a winning strategy.

To show a problem B is PSPACE-complete

1. Show $B \in$ PSPACE (often via DFS with polynomial memory or via a succinct dynamic program).

A handy template: reductions from TQBF

Game/constraint viewpoint

Evaluate a QBF as a two-player, perfect-information game with moves for $\exists$ (Eve) and $\forall$ (Adam). The formula is true iff Eve has a winning strategy.

To show a problem B is PSPACE-complete

1. Show $B \in$ PSPACE (often via DFS with polynomial memory or via a succinct dynamic program).
2. Reduce TQBF to B by letting players / constraints *simulate* quantifiers and the matrix φ.

A handy template: reductions from TQBF

Game/constraint viewpoint

Evaluate a QBF as a two-player, perfect-information game with moves for $\exists$ (Eve) and $\forall$ (Adam). The formula is true iff Eve has a winning strategy.

To show a problem B is PSPACE-complete

1. Show $B \in$ PSPACE (often via DFS with polynomial memory or via a succinct dynamic program).
2. Reduce TQBF to B by letting players / constraints *simulate* quantifiers and the matrix φ.
3. Ensure the game/instance size is polynomial and the play length (or search depth) is polynomially bounded.

Other PSPACE-complete problems (a sampler)

Logic / verification

- QSAT/TQBF (validity of fully-quantified formulas).
- LTL satisfiability and model checking.
- QBF with unrestricted alternations; bounded alternations capture levels of PH.
- NFA universality / language inclusion.

Planning / search

- STRIPS Plan-Existence (Bylander '94).
- Corridor tiling problem.

Games / puzzles (generalized to $n \times n$)

- Generalized Geography.
- Hex, Othello/Reversi, Node Kayles.
- Rush Hour, Sokoban.

General meta-theorem

Two-player, perfect-information games with polynomially bounded plays and polytime-checkable moves are typically PSPACE-complete via a reduction from TQBF.