CSCI 361 Lecture 21:

Time and Space Complexity

Shikha Singh

Announcements & Logistics

- No HW due this week due to Thanksgiving break 🎉
- **HW 10** will be released today and due following Wed (Dec 3)
    - **Optional** and grade will **replace** your lowest HW grade so far
    - But problems are good to practice for the final exam
- Final exam will be held Dec 12 (Friday) at 1 pm in this room
    - Cumulative with a emphasis on final 3-4 weeks

Last Time

- Wrapped up NP completeness reductions
  - VertexCover $\leq_p$ SubsetSum

Today

- Wrap up Time Complexity classes
- Discuss Space Complexity & the relationship between time and space

NP hard Problems Recap

- Different "classes" of NP hard problems to choose from:
    - **Satisfiability**: SAT/ 3-SAT
    - **Adjacency checks**: INDEPENDENT SET and CLIQUE
    - **Covering problems**: VERTEX COVER
    - **Coloring problems**: 3-COLOR
    - **Sequencing problems:** Hamiltonian cycle / path
    - **Packing problems:** SubsetSum/ Knapsack
- Reduction on the final exam will give you a few options that you can reduce from

Other NP-hard Problems

- **BIN-PACKING.** Given a set of items $I = \{1, \ldots, n\}$ where item i has size $s_i \in (0,1]$, bins of capacity c, find an assignment of items to bins that minimizes the number of bins used?
- **PARTITION.** Given a set S of n integers, are there subsets A and B such that $A \cup B = S$, $A \cap B = \emptyset$ and $\sum_{a \in A} a = \sum_{b \in B} b$

- **MAXCUT.** Given an undirected graph $G = (V, E)$, find a subset $S \subset V$ that maximizes the number of edges with exactly one endpoint in S.
- **MAX-2-SAT.** Given a Boolean formula in CNF, with exactly two literals per clause, find a variable assignment that maximizes the number of clauses with at least one true literal. (**2-SAT** on the other hand is in **P**)
- **3D-MATCHING**. Given n instructors, n courses, and n times, and a list of the possible courses and times each instructor is willing to teach, is it possible to make an assignment so that all courses are taught at different times?

Many More hard computational problems

Aerospace engineering. Optimal mesh partitioning for finite elements.

Biology. Phylogeny reconstruction.

Chemical engineering. Heat exchanger network synthesis.

Chemistry. Protein folding.

Civil engineering. Equilibrium of urban traffic flow.

Economics. Computation of arbitrage in financial markets with friction.

Electrical engineering. VLSI layout.

Environmental engineering. Optimal placement of contaminant sensors.

Financial engineering. Minimum risk portfolio of given return.

Game theory. Nash equilibrium that maximizes social welfare.

Mathematics. Given integer $a_1, \ldots, a_n$, compute

Mechanical engineering. Structure of turbulence in sheared flows.

Medicine. Reconstructing 3d shape from biplane angiocardiogram.

Operations research. Traveling salesperson problem.

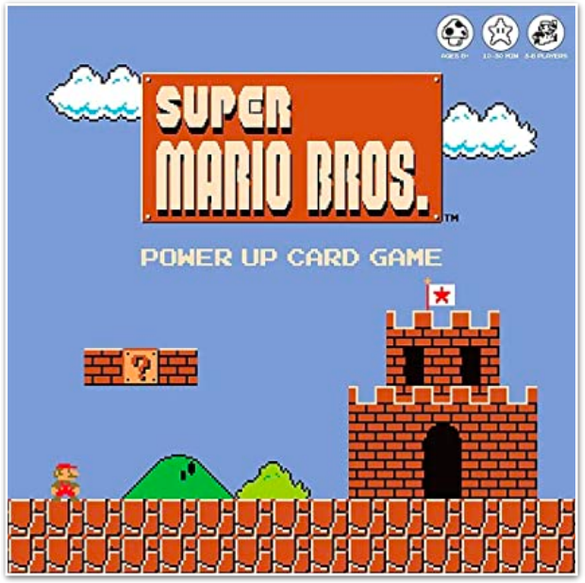
Physics. Partition function of 3d Ising model.

Politics. Shapley–Shubik voting power.

Recreation. Versions of Sudoku, Checkers, Minesweeper, Tetris, Rubik's Cube.

Fun NP-hard Games

- **MINESWEEPER** (from CIRCUIT-SAT)
- **SODUKO** (from 3-SAT)
- **TETRIS** (from 3PARTITION)
- **SOLITAIRE** (from 3PARTITION)
- **SUPER MARIO BROTHERS** (from 3-SAT)
- **CANDY CRUSH SAGA** (from 3-SAT variant)
- **PAC-MAN** (from Hamiltonian Cycle)
- **RUBIC's CUBE** (recent 2017 result, from Hamiltonian Cycle
- **TRAINYARD** (from Dominating Set)



Brief Look at Complexity Beyond NP and NP Completeness

Complexity Class: CoNP

- Not all problems are easily verifiable
- For example,

  UNSAT $= \{\phi \mid \phi$ is a 3SAT formula with no satisfying assignments$\}$
- No way to easily verify a yes instance. However, some such problems, no instances are easily verifiable
    - Given a "no" instance to UNSAT, a certificate that "refutes" its inclusion is a satisfying assignment
- coNP is the class of languages whose complement is in NP, that is,

  $\text{coNP} = \{L \mid \overline{L} \in \mathbf{NP}\}$
- Thus, UNSAT $\in$ **NP**

Complexity Class: CoNP

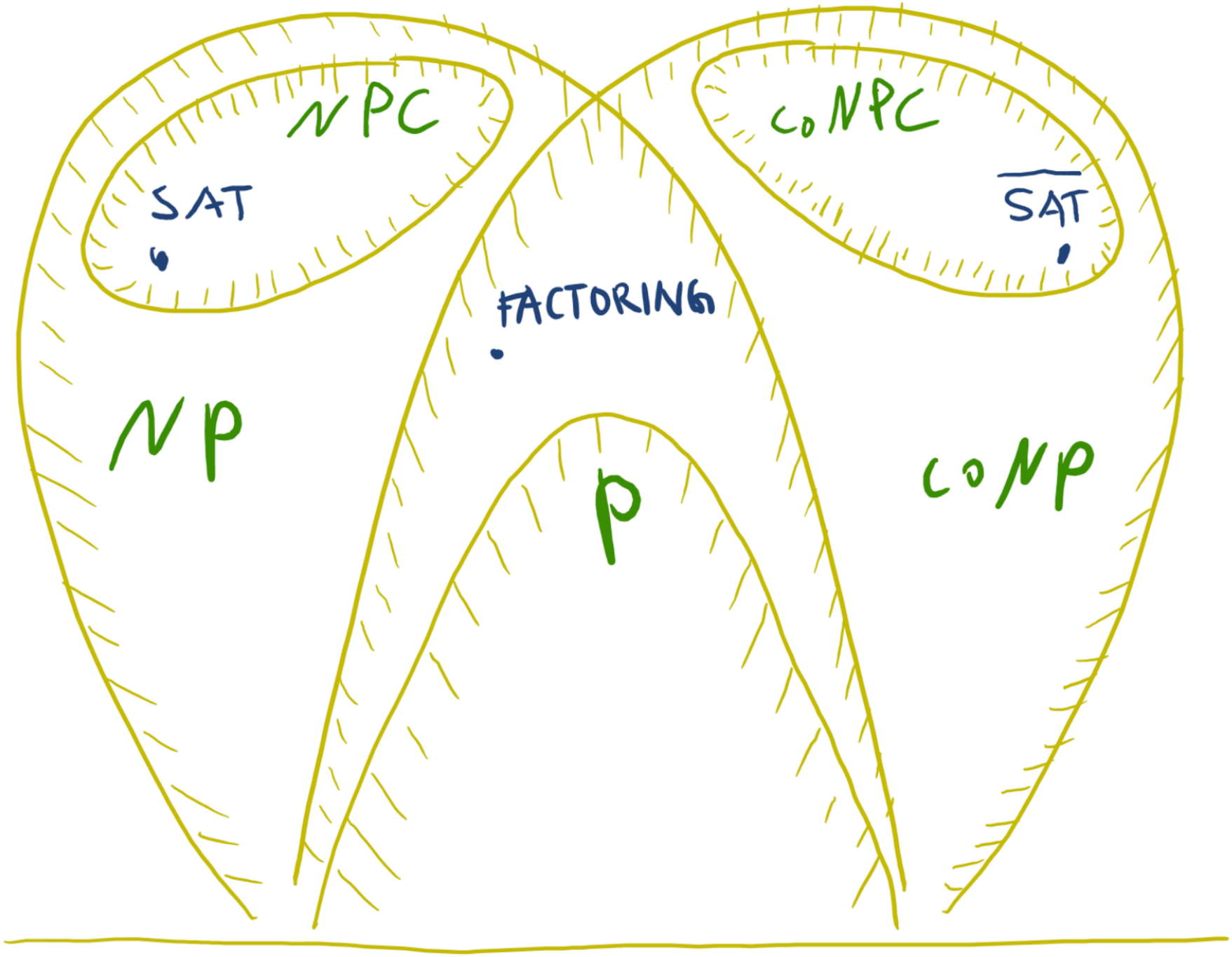
- Another example: given the binary representation of a number n, is n prime?
- Why is this in coNP?
    - Want to verify: "no, n is not prime"
    - Certificate is two factors x and y of n
    - Can verify that $x \cdot y = n$ in polynomial time
- Just so you know: this problem is not "hard" or "complete" for coNP
    - In fact, you can solve this problem in polynomial time

Complexity Class: CoNP

- $L \in$ **P**: Both yes & no instances can be decided in (deterministic) polynomial time
- $L \in$ **NP**: Every yes instance can be verified in polynomial time
- $L \in$ **coNP**: Every no instance can be verified in polynomial time

- Note that **P** $\subseteq$ **NP** $\cap$ **coNP**
- Many believe that **NP** $\neq$ **coNP**
- Proving this would also solve P versus NP problem, that is,
    - If **NP** $\neq$ **coNP**, then **P** $\neq$ **NP**

Conjectured Landscape

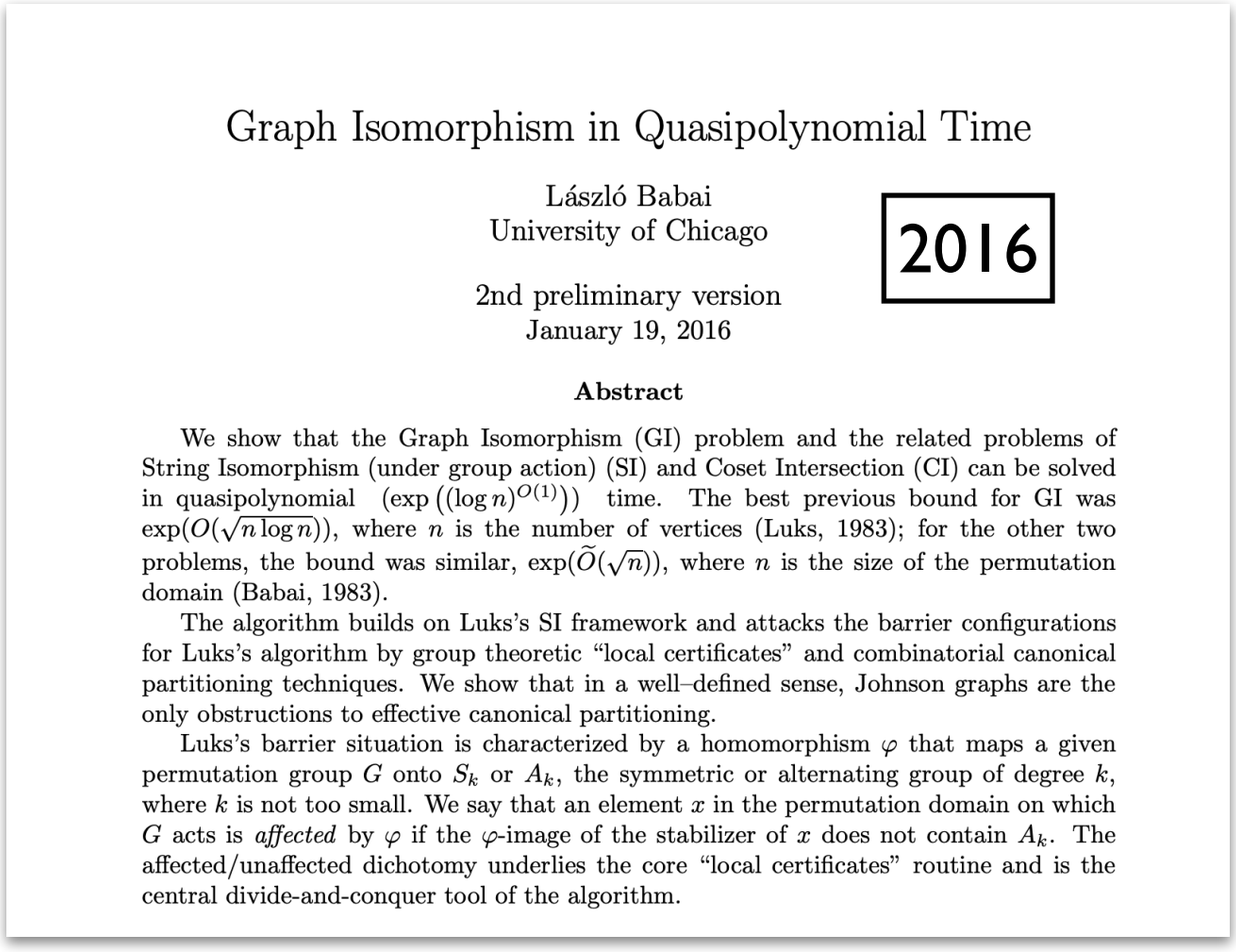


NP Intermediate Problems

- An NP intermediate problem is a problem that is neither in **P**, nor is it NP complete.
- You proved on HW 9 that **P = NP** if and only there are no NP intermediate problems
- We have problems that considered good "candidates" to be NP intermediate
    - Don't know if they are in **P** and or if they are NP complete

NP Intermediate Candidates

- **Factoring** (finding the prime factorization of a positive integer) is widely believed to be NP intermediate
- A lot of cryptographic protocols rely on Factoring being hard
- **Graph isomorphism** is another such problem
- Subexponential algorithms (but not polynomial-time) algorithm are known for both



Graph Isomorphism in Quasipolynomial Time

László Babai
University of Chicago

2016

2nd preliminary version
January 19, 2016

Abstract

We show that the Graph Isomorphism (GI) problem and the related problems of String Isomorphism (under group action) (SI) and Coset Intersection (CI) can be solved in quasipolynomial $(\exp\left((\log n)^{O(1)}\right))$ time. The best previous bound for GI was $\exp(O(\sqrt{n \log n}))$, where n is the number of vertices (Luks, 1983); for the other two problems, the bound was similar, $\exp(\widetilde{O}(\sqrt{n}))$, where n is the size of the permutation domain (Babai, 1983).

The algorithm builds on Luks's SI framework and attacks the barrier configurations for Luks's algorithm by group theoretic "local certificates" and combinatorial canonical partitioning techniques. We show that in a well–defined sense, Johnson graphs are the only obstructions to effective canonical partitioning.

Luks's barrier situation is characterized by a homomorphism φ that maps a given permutation group G onto S_k or A_k, the symmetric or alternating group of degree k, where k is not too small. We say that an element x in the permutation domain on which G acts is *affected* by φ if the φ-image of the stabilizer of x does not contain A_k. The affected/unaffected dichotomy underlies the core "local certificates" routine and is the central divide-and-conquer tool of the algorithm.

PRIMEs are in P

- For a while the problem of determining whether an integer is prime was considered NP intermediate
- Breakthrough in 2002: AKS primality testing poly-time algorithm

Efficient solution for primality testing was found in 2002

PRIMES is in P

Manindra Agrawal Neeraj Kayal
Nitin Saxena*

Department of Computer Science & Engineering
Indian Institute of Technology Kanpur
Kanpur-208016, INDIA
Email: {manindra,kayaln,nitinsa}@iitk.ac.in

Abstract

We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.

1 Introduction

Prime numbers are of fundamental importance in mathematics in general, and number theory in particular. So it is of great interest to study different properties of prime numbers. Of special interest are those properties that allow one to efficiently determine if a number is prime. Such efficient tests are also useful in practice: a number of cryptographic protocols need large prime numbers.

Let PRIMES denote the set of all prime numbers. The definition of prime numbers already gives a way of determining if a number n is in PRIMES: try dividing n by every number $m \leq \sqrt{n}$—if any m divides n then it is composite, otherwise it is prime. This test was known since the time of the ancient Greeks—it is a specialization of the *Sieve of Eratosthenes* (ca. 240 BC) that generates all primes less then n. The test, however, is inefficient: it takes $\Omega(\sqrt{n})$ steps to determine if n is prime. An efficient

Do NP hard Problems Require EXPTime?

- So far, we have said that if **P ≠ NP**, then NP complete problems cannot be solved in polynomial time
    - But this does not tell if they really require exponential time
    - E.g., can SAT be solved in super-polynomial (but sub-exponential time) such as $O\left(2^{\sqrt{N}}\right)$ time?
    - Experts believe that this is unlikely
- **ETH (Exponential-time hypothesis).** There is a constant $c > 1$ such that every algorithm solving 3SAT requires time at least c^n

Circumventing NP Hardness

- To solve NP hard problems, must compromise on one of the following guarantees:
  - ***General-purpose.*** The algorithm accommodates all possible inputs of the computational problem
    - Compromise: design for domain-specific special instances
  - ***Correct.*** For every input, the algorithm correctly solves the problem.
    - Correct on "most" inputs or "mostly correct" on all inputs
  - ***Fast.*** For every input, the algorithm runs in polynomial time.
    - Algorithms that improve on exhaustive search but are not poly-time or algorithms that run quickly on relevant instances

Space Complexity

Space Complexity

- **Definition.** Let M be a deterministic TM that halts on all inputs. The space complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of tape cells that M scans on any input of length n; we say M runs in $f(n)$ space.

  If M is a non-deterministic TM wherein all branches halt on all inputs, then $f(n)$ is the maximum number of tape cells that M scans on any branch of its computing for any input of length n.

Space Complexity Classes

- SPACE($f(n)$) is the set of all languages decided by an $O(f(n))$-space deterministic TM
- NSPACE($f(n)$) is the set of all languages decided by an $O(f(n))$ -space non-deterministic TM

- **Question.** How does time and space complexity compare with each other?
    - If a DTM uses $f(n)$ space, how much time must it use?
- **Observation 1.** DTIME($f(n)$) $\subseteq$ SPACE($f(n)$)

Space Time Comparison

- Suppose a DTM uses $f(n)$ space, how much time can it take?
    - Can reuse each cell over and over again
    - E.g. consider a DTM that uses $f(n)$ space to increment a counter from 1 to $2^{f(n)-1}$ takes $\Omega(2^{f(n)})$ time

- We can show that a DTM that uses $f(n)$ space, takes at most $O(2^{d(f(n)})$ time.
    - Consider a configuration of a TM uqv where uv is on the tape and the head is on the first symbol of v
    - A TM that halts must not repeat configurations, at most $2^{O(f(n))}$ configurations if space is at most $f(n)$

Savitch's Theorem

- Surprisingly Savitch's theorem states that:
  $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{SPACE}(f(n)^2)$
- ***Proof idea.*** A NTM N that uses $f(n)$ space must run in time at most $2^{O(f(n))}$: go from start to accept configuration in $t = 2^{d \cdot f(n)}$ steps
  - Given N, create a DTM that recursively searches for an intermediate configuration c_m s.t. N goes from $c_1 \to c_m$ in $t/2$ steps and $c_m \to c_{\text{accept}}$ in $t/2$ steps (reuse the space for recursive calls)
  - Depth of recursion: $\log_2 2^{df(n)} = O(f(n))$; each recursive call uses $O(f(n))$ space
  - Overall can do simulation in $O(f(n)^2)$ space

Space Complexity

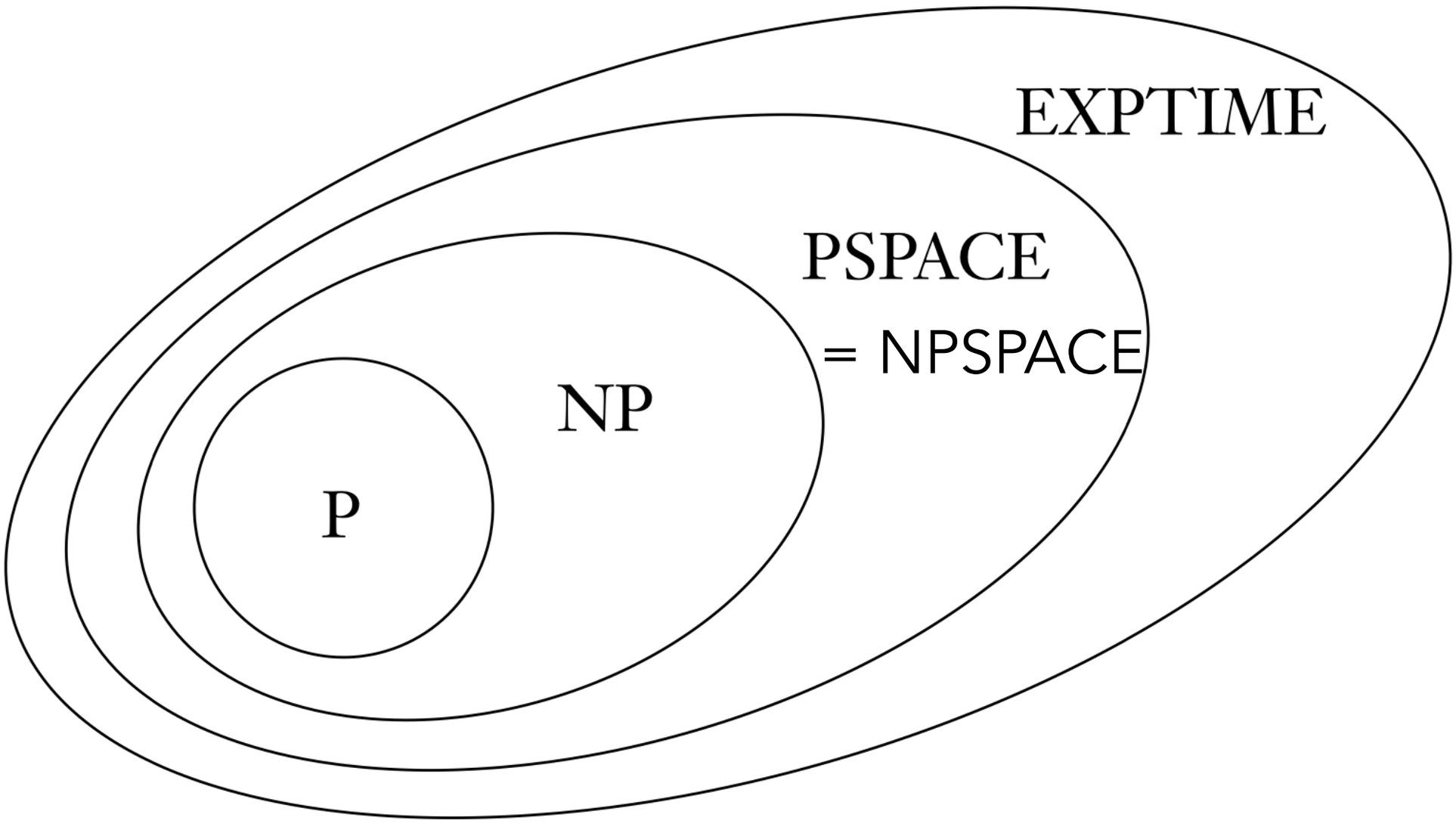
- PSPACE $= \cup_k$ SPACE(n^k)
- NPSPACE $= \cup_k$ NSPACE(n^k)

- **Theorem (Savitch's Theorem).** PSPACE = NPSPACE

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}.$$

We know P ≠ EXPTIME, so one of these containments is proper but we don't know which one

Belief: All Containments are Proper



P versus PSPACE

- Can a polynomial space algorithm be simulated in polynomial time?
    - We strongly suspect that **P ≠ PSPACE** but there had not been much progress towards in over five decades
- Over 50 years ago,

[Hopcroft and Valiant, 1975]

$$\text{DTIME}(f(n)) \subseteq \text{SPACE}(f(n)/\log n)$$

- **Takeaways.** Space is a more powerful resource than time!
    - A linear space algorithm requires $\Omega(n \log n)$ time
- One way to show **P ≠ PSPACE**, is to show that a linear-space algorithm requires super-polynomial $\Omega(n^k)$ time

Last Year: Biggest Breakthrough

Simulating Time with Square-Root Space

R. Ryan Williams*
Massachusetts Institute of Technology
Cambridge, USA
rrw@mit.edu

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]. Among other results, our simulation implies that bounded fan-in circuits of size s can be evaluated on any input in only $\sqrt{s} \cdot poly(\log s)$ space, and that there are explicit problems solvable in $O(n)$ space which require at least $n^{2-\varepsilon}$ time on every multitape Turing machine for all $\varepsilon > 0$, thereby making a little progress on the P versus PSPACE problem.

Our simulation reduces the problem of simulating time-bounded multitape Turing machines to a series of implicitly-defined Tree Evaluation instances with nice parameters, leveraging the remarkable space-efficient algorithm for Tree Evaluation recently found by Cook and Mertz [STOC 2024].

there is a linear-space problem that cannot be solved in $O(n^k)$ time for every constant $k \geq 1$.

In this paper, we make another step towards separating P from PSPACE, by showing that there are problems solvable in $O(n)$ space that cannot be solved in $n^2/\log^c n$ time for some constant $c > 0$. This is the first generic polynomial separation of time and space in a robust computational model (namely, the multitape Turing machine). The separation is accomplished by exhibiting a surprisingly space-efficient simulation of generic time-bounded algorithms. More formally, let $\mathsf{TIME}[t(n)]$ be the class of decision problems decided by $O(t(n))$-time multitape Turing machines, and let $\mathsf{SPACE}[s(n)]$ the class decided by $O(s(n))$-space multitape Turing machines.

Theorem 1.1. *For every function $t(n) \geq n$,*

$$\mathsf{TIME}[t(n)] \subseteq \mathsf{SPACE}[\sqrt{t(n) \log t(n)}].$$

Last Year: Biggest Breakthrough

- Ryan Williams in 2025 proved that any algorithm that runs in t time can be simulated in $\sqrt{t}$ space, that is,

$$\mathsf{DTIME}(f(n)) \subseteq \mathsf{SPACE}(\sqrt{f(n)\log n})$$

- Surprisingly, he gave an explicit construction to do this

- Biggest step towards $\mathsf{P} \neq \mathsf{PSPACE}$ since this shows that a linear space algorithm requires (nearly) quadratic time

    - If this could be applied recursively to boost the polynomial degree, then $\mathsf{P} \neq \mathsf{PSPACE}$!

PSPACE Complete Problem

- A language B is **PSPACE-complete** if it satisfies two conditions:
  - B is in PSPACE, and
  - every language A in PSPACE is polynomial-time reducible to B
- PSPACE complete problems are the hardest problems in PSPACE
  - An efficient solution to one means efficient solution to all

PSPACE Completeness Intuition

- 2-Player Games where we want to know if either player has a winning strategy capture the essence of PSPACE completeness
- Consider a generalized $n \times n$ Chess game
  - What does it mean for player 1 to have a winning strategy?
    - There exists a move for player 1 s.t. for all moves of player 2, there exists another move of player, s.t. for all moves of player 2,.... and so on, at the end of which player 1 wins
    - ∃ Player 1 move ∀ Player 2 moves ∃ Player 1 move …. such that Player 1 wins
- Such games are PSPACE complete

PSPACE Completeness Intuition

- 2-Player Games where we want to know if either player has a winning strategy capture the essence of PSPACE completeness
- Consider a generalized $n \times n$ Chess game
  - ∃ Player 1 move ∀ Player 2 moves ∃ Player 1 move ….
    such that Player 1 wins
- Notice, checking if a player has a winning strategy, requires searching through an exponential-size game tree (can be done in poly-space by reusing space)
  - Similar to NP hard problems BUT
  - No short certificate showing winning strategy exists!

PSPACE Completeness Intuition

- Essentially, PSPACE is the class of 2-player zero-sum games with perfect information
- PSPACE complete problem:
  TQBF = $\{\phi \mid \phi$ is a true quantified Boolean formula$\}$
- Example of a quantified Boolean formula

$$\exists x_1 \, \forall x_2 \, \exists x_3 \, \left[(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3}) \right].$$

PSPACE and Hardness of Games

- A lot of games a PSPACE complete!