Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Interactive Proofs and the PSPACE Landscape: A Practical Investigation of the Space-Time Barrier

Ayush Saini
Independent Researcher

Abstract

The relationship between deterministic polynomial time (P) and polynomial space (PSPACE) is one of the foundational open problems in computational complexity theory. While proving P = PSPACE remains elusive and is widely believed to be false, the characterizations of PSPACE have yielded profound insights into modern computer science, specifically cryptography and zero-knowledge proofs. This paper surveys the landscape of PSPACE, examines the three fundamental barriers preventing resolution, and presents original systems-level experiments in C and Python that make the space-time tradeoff at the heart of the problem tangible and measurable.

1 Introduction

In the landscape of computational complexity, PSPACE represents the set of all decision problems solvable by a deterministic Turing machine using a memory footprint that is polynomial in the input size n. Formally, we define:

$$\mathsf{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathsf{SPACE}(n^k)$$

The significance of PSPACE stems from its surprising robustness. A cornerstone result, Savitch's Theorem (1970), establishes that NPSPACE = PSPACE, implying that nondeterministic polynomial space is no more powerful than its deterministic counterpart—a sharp contrast to the widely held belief that P $\neq$ NP.

The relationship between time and space is captured in the standard inclusion chain:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$$

While it is known that NL $\subsetneq$ PSPACE and P $\subsetneq$ EXP by the Space and Time Hierarchy Theorems, respectively, the exact separation between P and PSPACE remains one of the deepest mysteries in the field. If P = PSPACE, it would imply that memory is essentially interchangeable with time, and that complex existential and universal quantifications (inherent in PSPACE-complete problems like TQBF) can be resolved in polynomial time. This paper explores the landscape of this boundary through both theoretical survey and original experimental instrumentation.

2 Interactive Proofs and Arithmetization

The proof that IP = PSPACE (Shamir, 1992) is widely regarded as one of the most beautiful results in complexity theory. It demonstrates that a verifier with limited time but access to randomness can verify the solution to any PSPACE problem by interacting with an untrusted, all-powerful prover.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

2.1 Arithmetization

The transformation begins with *arithmetization*: converting a boolean formula ϕ into a multivariate polynomial P over a finite field $\mathbb{F}$. This is achieved via a recursive mapping:

- $x \to x$
- $\neg x \to 1 - x$
- $x \wedge y \to x \cdot y$
- $x \vee y \to x + y - x \cdot y$

Quantifiers are then treated as operators: $\forall x \phi(x)$ becomes $\prod_{x \in \{0,1\}} P(x)$, and $\exists x \phi(x)$ becomes $\sum_{x \in \{0,1\}} P(x)$ (after suitable linearized adjustments to maintain low degree).

2.2 The Sum-Check Protocol

The Sum-Check protocol allows a verifier to check the sum of a v-variable polynomial $P(x_1, \ldots, x_v)$ over the boolean hypercube $\{0,1\}^v$.

1. **Initial Claim:** The Prover claims that $S = \sum_{x_1,\ldots,x_v \in \{0,1\}} P(x_1, \ldots, x_v)$.
2. **Round 1:** The Prover sends a univariate polynomial $g_1(X) = \sum_{x_2,\ldots,x_v \in \{0,1\}} P(X, x_2, \ldots, x_v)$. The Verifier checks if $g_1(0) + g_1(1) = S$.
3. **Interaction:** The Verifier sends a random challenge $r_1 \in \mathbb{F}$. The new goal is to verify $g_1(r_1)$, which reduces the problem to $v - 1$ variables.
4. **Termination:** After v rounds, the Verifier makes a single query to an oracle (or evaluates the polynomial itself) at $P(r_1, \ldots, r_v)$ to ensure the Prover has not cheated.

This reduction from an exponential sum to a linear number of rounds is the technical engine that empowers the $\mathsf{IP} = \mathsf{PSPACE}$ characterization.

3 Theoretical Barriers to Resolution

The fact that P vs PSPACE remains unresolved is not due to a lack of effort, but due to deep mathematical barriers that invalidate standard proof techniques.

3.1 The Relativization Barrier

Baker, Gill, and Solovay (1975) proved that there exist oracles A and B such that $\mathsf{P}^A = \mathsf{PSPACE}^A$ and $\mathsf{P}^B \neq \mathsf{PSPACE}^B$. This implies that any technique that "relativizes" (i.e., remains valid if the Turing machines are given access to an oracle) cannot possibly resolve the question. Most techniques in early complexity theory, such as diagonalization and simulation, are relativizing. A successful proof must therefore exploit properties of the computation model that do not persist under oracle access.

3.2 The Natural Proofs Barrier

Razborov and Rudich (1993) identified that most attempts to prove lower bounds on circuit complexity follow a pattern they called "Natural Proofs." They showed that if strong pseudo-random generators exist (a widely held belief in cryptography), then no "natural" proof can separate complexity classes like P from PSPACE. A "natural" proof is one that can efficiently identify a large set of "hard" functions. The existence of PRGs implies that "hard" functions are indistinguishable from "easy" ones by efficient algorithms, thus blocking this path.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

3.3 The Algebrization Barrier

Following the success of arithmetization in proving IP = PSPACE, Aaronson and Wigderson (2008) showed that even these algebraic techniques are subject to a new barrier. They defined "algebrization" as a generalization of relativization that accounts for the power of polynomials. They proved that algebrizing techniques are also insufficient to separate P from PSPACE. This suggests that arithmetization, while powerful, still does not capture the essential difference between polynomial time and polynomial space.

Synthesis: These barriers collectively suggest that any resolution will require a fundamentally new insight into the nature of computation that transcends both simple logic (relativization), structural properties (natural proofs), and algebraic structure (algebrization).

4 Software Architecture and System Design

To make these abstract concepts tangible, we developed a suite of high-performance tools in C and Python. The suite is designed to demonstrate the space-time tradeoff and the mechanics of interactive proofs.

4.1 TQBF Solver Architecture (C)

The TQBF solver is implemented in C for maximum performance. It utilizes a recursive DPLL-style algorithm adapted for quantifiers. Key features include:

- **Bitwise State Tracking:** Current variable assignments are stored in a single `uint64_t` bitmask, allowing for $O(1)$ state updates and minimal memory overhead.
- **Pruning:** The solver implements alpha-beta style pruning. For existential quantifiers ($\exists x$), the search short-circuits as soon as a TRUE assignment is found. For universal quantifiers ($\forall x$), it short-circuits on FALSE.
- **Instrumentation:** The code tracks peak recursion depth and total node count to empirically measure $O(n)$ space and $O(2^n)$ time complexity.

4.2 Interactive Proof Simulation (Python)

The Sum-Check protocol is simulated in Python to leverage its arbitrary-precision integers and readability.

- **Finite Field Arithmetic:** All calculations are performed modulo a large prime $(10^9 + 7)$ to simulate a finite field $\mathbb{F}_p$.
- **Lagrange Interpolation:** The verifier uses Lagrange interpolation to reconstruct univariate polynomials from a minimal set of evaluation points, ensuring $O(v)$ verification time.
- **Adversarial Simulation:** The `soundness_analysis.py` script extends this framework by implementing cheating prover strategies to test the protocol's robustness.

5 Experimental Investigations

While we cannot resolve P vs PSPACE by writing programs, we *can* build systems that produce original experimental data about the mechanisms underpinning PSPACE. We present five experiments, the first of which constitutes our primary original contribution.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

5.1 Original Contribution: Soundness Analysis of the Sum-Check Protocol

We conduct an empirical investigation into the *soundness* of the Sum-Check Protocol the core mechanism of IP = PSPACE by simulating four dishonest prover strategies and measuring their rejection rates across thousands of randomized trials.

5.1.1 Research Questions

RQ1. Does the empirical rejection rate match the theoretical soundness bound $1 - vd/|\mathbb{F}|$?

RQ2. How does finite field size affect detection probability?

RQ3. Which cheating strategies are hardest for the verifier to detect?

RQ4. At which round is a cheating prover most commonly caught?

5.1.2 Cheating Prover Strategies

We implemented four distinct dishonest prover strategies:

- **Random Polynomial**: sends a uniformly random polynomial satisfying the sum constraint $g(0) + g(1) = S'$
- **Honest-then-Lie**: computes the honest polynomial in round 1 but shifts it to match a false sum, then sends random polynomials in subsequent rounds
- **Perturb One Evaluation**: computes the honest polynomial but perturbs a single non-binary evaluation point
- **Constant Shift**: adds a uniform offset to all evaluations of the honest polynomial

5.1.3 Results: Field Size Impact (RQ2)

$\lvert\mathbb{F}\rvert$	$vd/\lvert\mathbb{F}\rvert$	Theoretical Rej. Rate	Empirical Rej. Rate	Δ
7	1.286	0.000	0.873	+0.873
13	0.692	0.308	0.917	+0.609
31	0.290	0.710	0.973	+0.263
127	0.071	0.929	0.986	+0.057
1021	0.009	0.991	0.999	+0.008
$10^6 + 3$	0.000009	0.999991	1.000	+0.000

Table 1: Empirical rejection rate vs. theoretical bound across field sizes. The empirical rate consistently *exceeds* the theoretical lower bound, confirming that the bound $1 - vd/|\mathbb{F}|$ is not tight.

5.1.4 Key Findings

1. **Soundness confirmed**: Across all 4 strategies and 2000 trials each over $\mathbb{F}_{10^9+7}$, the verifier achieves 100% rejection, consistent with the near-zero theoretical error bound (9×10^{-9}).
2. **Field size is critical**: Over $\mathbb{F}_7$, cheating provers escape detection $\sim$12.7% of the time. Over $\mathbb{F}_{1021}$, only $\sim$0.1% escape. This has **practical implications** for choosing parameters in real-world interactive proof deployments (e.g., in zkSNARKs).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

3. **The bound is loose**: Empirical rejection rates consistently exceed the theoretical bound by a positive margin, suggesting the Schwartz-Zippel-based analysis yields a conservative (non-tight) bound for structured cheating strategies.

4. **Detection concentrates at the final check**: Over large fields, all cheating strategies trivially pass the per-round sum checks (since they are constructed to satisfy $g(0)+g(1) = S'$). Detection occurs at the final oracle evaluation, where the verifier directly queries the polynomial.

5.2 Experiment 2: Sum-Check Protocol Implementation

Our base simulation demonstrates $\mathsf{IP} = \mathsf{PSPACE}$ by implementing the Sum-Check Protocol with a polynomial-time Verifier using Lagrange interpolation over $\mathbb{F}_p$. Prover/Verifier timing confirms the fundamental asymmetry: at $v = 7$ variables, the Prover takes $\sim 1.5\,\text{ms}$ while the Verifier takes < 0.001 ms — a ratio exceeding $1000\times$.

5.3 Experiment 3: TQBF Solver Instrumentation

We developed a high-performance solver for True Quantified Boolean Formulas (TQBF) to empirically study the $O(2^n)$ time vs. $O(n)$ space gap. The solver implements a recursive evaluation based on the standard identity:

$$TQBF(\exists x\phi) = TQBF(\phi[x=0]) \vee TQBF(\phi[x=1])$$
$$TQBF(\forall x\phi) = TQBF(\phi[x=0]) \wedge TQBF(\phi[x=1])$$

The solver utilizes a recursive DPLL-style search with alpha-beta pruning tailored for quantifiers.

5.3.1 Scaling Results

As shown in Table 2, the number of nodes explored remains manageable for structured or pruned formulas, but the stack depth strictly follows the number of variables, confirming the PSPACE bound.

Variables	Nodes Explored	Peak Stack Depth	Memory Usage (est.)
10	11	10	280 B
14	58	14	392 B
18	53	18	504 B
20	95	20	560 B

Table 2: TQBF solver performance. While time complexity is $O(2^n)$ in the worst case, space usage remains strictly linear ($O(n)$) as each stack frame stores only a few bytes of state.

5.4 Experiment 4: Savitch's Theorem in Practice

Savitch's Theorem proves that any nondeterministic polynomial space computation can be simulated deterministically in $O(\log^2 n)$ space. We implemented the constructive proof using the $\texttt{CanReach}(u, v, k)$ recursive algorithm, which is defined by:

$$\texttt{CanReach}(u, v, k) = \exists z \in V(\texttt{CanReach}(u, z, k-1) \wedge \texttt{CanReach}(z, v, k-1))$$

This algorithm checks if node v is reachable from u in 2^k steps by trying all possible middle nodes z.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

5.4.1 Space Comparison

We compared the memory footprint of Savitch's algorithm against a standard Breadth-First Search (BFS). BFS requires $O(n)$ space to store the visited set and queue, whereas Savitch's algorithm uses only $O(\log^2 n)$ space for the recursion stack.

Nodes (n)	BFS Space Usage	Savitch Space Usage	BFS Time (ms)
8	64 B	128 B	0.01
16	128 B	160 B	0.02
32	256 B	192 B	0.05
64	512 B	224 B	0.12

Table 3: Savitch's algorithm vs. BFS. As n increases, Savitch's space usage grows much slower than BFS, empirically verifying the quadratic logarithmic bound.

The tradeoff is severe: while BFS runs in polynomial time ($O(n+m)$), Savitch's algorithm runs in super-polynomial time ($O(n^{\log n})$) because it recomputes reachability for midpoints repeatedly.

5.5 Experiment 5: The "Memory Wall" and Space-Time Tradeoff

This experiment directly visualizes the hard physical limit of space-bounded computation. We count simple paths in a graph using three strategies: pure backtracking (minimal space), full Dynamic Programming (minimal time), and bounded memoization.

5.5.1 The Memory Wall

For a graph with n nodes, the DP table requires $O(n \cdot 2^n)$ space. As shown in Table 4, once n reaches 24, the memory requirement exceeds 512 MB, leading to allocation failure on standard hardware.

n	Backtrack Time (ms)	DP Time (ms)	DP Space
16	57	6	160 KB
18	1,497	48	640 KB
20	89,943	442	2.5 MB
24	*Timed Out*	*Allocation Failed*	$>$ 512 MB

Table 4: The Memory Wall. When memory is exhausted, we are forced to revert to exponential-time backtracking. This confirms that polynomial time is often a luxury afforded only by exponential space.

This experiment provides a tangible metaphor for the P vs PSPACE question: if we could always solve these problems in polynomial time using only polynomial space, the "memory wall" would not exist.

6 Conclusion

Our primary contribution is an empirical soundness analysis of the Sum-Check Protocol that produces three original findings: (1) the theoretical bound $1 - vd/|\mathbb{F}|$ is consistently non-tight, (2) field size selection has measurable practical consequences for security, and (3) detection dynamics concentrate at the final oracle query rather than intermediate rounds. These experimental results are supported by systems-level investigations in C that make the space-time

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

barrier tangible. While resolving $\mathsf{P} \stackrel{?}{=} \mathsf{PSPACE}$ requires fundamentally new mathematics, this work demonstrates that experimental methodology can yield novel insights into the mechanisms that make the problem hard.

References:

- Shamir, A. (1992). $\mathsf{IP} = \mathsf{PSPACE}$. *Journal of the ACM*, 39(4), 869-877.
- Arora, S., & Barak, B. (2009). *Computational Complexity: A Modern Approach.*
- Baker, T., Gill, J., & Solovay, R. (1975). Relativizations of the $\mathsf{P} =?\mathsf{NP}$ question. *SIAM Journal on Computing*, 4(4), 431-442.
- Razborov, A. A., & Rudich, S. (1997). Natural proofs. *Journal of Computer and System Sciences*, 55(1), 24-35.
- Aaronson, S., & Wigderson, A. (2009). Algebrization: A new barrier in complexity theory. *ACM TOCT*, 1(1), 1-54.
- Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *JCSS*, 4(2), 177-192.