A SPACE-COMPRESSION ARGUMENT FOR THE BEST-MOVE FUNCTION IN EXPTIME-COMPLETE GAMES

LIGHTMAN CHANG

Abstract. For two-player perfect-information games whose game-value problem is EXPTIME-complete, the natural approach to lower-bounding the *best-move function* m_1 is via self-reduction: simulate optimal play move by move using an FP algorithm for m_1, recover the game value, and contradict P $\neq$ EXPTIME. This approach fails because the optimal-play trajectory can have superpolynomial length; the simulator cannot follow it within polynomial time.

We propose a different reduction that uses *polynomial space* rather than polynomial time. Given an FP algorithm $\mathcal{A}$ for m_1, we simulate optimal play using only two position pointers and the $\mathcal{A}$ workspace, detecting cycles in the position graph by the Floyd two-pointer method. The simulator runs in PSPACE, producing a PSPACE decision algorithm for any EXPTIME-complete game-value language. We obtain **Theorem A:** for generalized $n \times n$ Go under the basic ko rule, $m_1 \in$ FP implies EXPTIME = PSPACE.

The argument uses no special property of Go beyond polynomial position size and the EXPTIME-completeness of the value problem; we therefore state and prove it for a general framework of *compactly representable EXPTIME-complete games*. As a structural by-product we record **Theorem B:** every EXPTIME-complete two-player perfect-information game with polynomial-size positions has superpolynomial game length, conditional only on EXPTIME $\neq$ PSPACE.

The paper is independent of the bottleneck-encoding construction used for the second-best move and uses no Robson-specific gadget hypothesis.

1. Introduction

Generalized Go on an $n \times n$ board under the basic ko rule is EXPTIME-complete [7], sharpening the earlier PSPACE-hardness result of Lichtenstein and Sipser [6]. Generalized chess is also EXPTIME-complete [3], as is generalized checkers [8]. For each of these games, the natural folklore expectation is that the function $m_1(S)$—the best move from position S under minimax—admits no polynomial-time algorithm. The expectation is not a theorem.

The standard self-reduction template would proceed as follows. Suppose $m_1 \in$ FP via an algorithm $\mathcal{A}$. To decide whether $V(S_0) > 0$, simulate optimal play by repeatedly setting $S \leftarrow \text{apply}(S, \mathcal{A}(S))$ until S is terminal, then return the sign of score(S). The argument yields $V \in$ P, hence EXPTIME = P, contradicting the time hierarchy theorem [5]. *This argument fails.* The optimal-play trajectory in Go (and in any other EXPTIME-complete game) can have superpolynomial length; the simulator runs out of polynomial time before reaching a terminal position. The same obstruction defeats every naive attempt to get a polynomial-time reduction from the value problem to m_1.

The space-compression idea. The simulator does not need polynomial time; polynomial space suffices. Each step requires only the current position S (size $O(\text{poly}(n))$), the $\mathcal{A}$ workspace (size $O(\text{poly}(n))$), and constant bookkeeping. Cycles in the position graph are detected by Floyd's two-pointer method [4], requiring one extra position pointer of the same size. The total space is $O(\text{poly}(n))$. Polynomial time is replaced by polynomial *space*, and the contradiction becomes EXPTIME = PSPACE instead of EXPTIME = P. The former is widely believed false but is not a theorem; nevertheless this reduction sharpens the obstruction by exactly one complexity-class inclusion.

Date: May 5, 2026.

2020 *Mathematics Subject Classification.* Primary 91A46; Secondary 68Q15, 68Q17, 68Q05.

Key words and phrases. EXPTIME-complete games, generalized Go, best move, space complexity, Floyd cycle detection, time hierarchy theorem, game-length lower bound.

Contributions.

(C1) **Compactly representable game framework (Definition 2.1).** A clean abstraction of the structural features common to Go, chess, and checkers that the argument needs.
(C2) **Space-compression theorem (Theorem 3.1).** For any compactly representable EXPTIME-complete game, $m_1 \in \mathrm{FP}$ implies EXPTIME = PSPACE.
(C3) **Specialization to Go (Corollary 3.2).** For generalized $n \times n$ Go under the basic ko rule, $m_1 \in \mathrm{FP}$ implies EXPTIME = PSPACE.
(C4) **Game-length meta-theorem (Theorem 4.2).** Every compactly representable EXPTIME-complete game has superpolynomial game length, conditional on EXPTIME $\neq$ PSPACE. This identifies game length as the structural reason why the time-based self-reduction fails.

What this paper does not claim. We do not prove $m_1 \notin \mathrm{FP}$ unconditionally. The conclusion has the form "$m_1 \in \mathrm{FP} \Rightarrow$ EXPTIME = PSPACE," which is a conditional negative result. Whether EXPTIME = PSPACE is open; both equality and strict containment are consistent with current knowledge. We also do not address the second-best move function m_2; that is the topic of a separate companion paper.

Paper structure. Section 2 fixes notation and the framework. Section 3 states and proves the space-compression theorem. Section 4 proves the game-length meta-theorem. Section 5 discusses scope, comparison with known reductions, and open problems.

2. Framework: Compactly Representable Games

We axiomatize the structural features of Go (and of chess, checkers, and other EXPTIME-complete board games) that the space-compression argument uses.

Definition 2.1 (Compactly representable game)**.** A *compactly representable two-player perfect-information game with parameter n* consists of:

(i) a set $\mathcal{P}_n$ of positions with $|\mathcal{P}_n| \le 2^{p(n)}$ for some polynomial p;
(ii) a position encoding such that each $S \in \mathcal{P}_n$ has bit-length $|S| = O(\mathrm{poly}(n))$;
(iii) a set $M(S)$ of legal moves at S with $|M(S)| \le \mathrm{poly}(n)$ and a legal-move enumerator computable in time $O(\mathrm{poly}(n))$;
(iv) a transition function apply: $\mathcal{P}_n \times M \to \mathcal{P}_n$ computable in time $O(\mathrm{poly}(n))$;
(v) a terminality predicate term: $\mathcal{P}_n \to \{0,1\}$ computable in time $O(\mathrm{poly}(n))$;
(vi) a score function score: $\{S : \mathrm{term}(S) = 1\} \to \mathbb{Z}$ computable in time $O(\mathrm{poly}(n))$;
(vii) a tie-breaking total order on M for cycles, used to define the value V on infinite optimal-play paths.

The minimax value $V\colon \mathcal{P}_n \to \mathbb{Z}$ is defined by Definition 2.2 below, and the best-move function is $m_1(S) = \arg\max_{m \in M(S)} \big[-V(\mathrm{apply}(S,m))\big]$ with ties broken by the fixed total order.

Definition 2.2 (Game value)**.** For a compactly representable game,

$$V(S) = \begin{cases} \mathrm{score}(S) & \text{if } \mathrm{term}(S) = 1, \\ 0 & \text{if optimal play from } S \text{ enters an infinite cycle}, \\ \max\limits_{m \in M(S)} \big[-V(\mathrm{apply}(S,m))\big] & \text{otherwise.} \end{cases}$$

The cycle case captures triple-ko situations in Go and the threefold repetition rule in chess, both of which we treat as draws of value 0.

Example 2.3 (Generalized $n \times n$ Go)**.** Generalized $n \times n$ Go under the basic ko rule is a compactly representable game with the position encoding from [7]: a position consists of a board configuration in $\{\bullet, \circ, \emptyset\}^{n^2}$, a side-to-move bit, and a ko-forbidden coordinate. Hence $|\mathcal{P}_n| \le 3^{n^2} \cdot 2 \cdot (n^2+1)$ and $|S| = O(n^2)$. The legal-move enumerator runs in time $O(n^2)$, apply in time $O(n^2)$, term checks for two consecutive passes, and score counts territory in $O(n^2)$.

Example 2.4 (Generalized chess and checkers)**.** The generalized $n \times n$ versions of chess [3] and checkers [8] fit the framework with the obvious encodings of position, move set, and score.

2.1. **The value language.** We work with the decision version of the value problem.

Definition 2.5 (Value language)**.** For a compactly representable game $\mathcal{G}$, define $L_{\mathcal{G}} = \{S : V(S) > 0\}$.

We say $\mathcal{G}$ is *EXPTIME-complete* if $L_{\mathcal{G}}$ is EXPTIME-complete under polynomial-time many-one reductions. By [7, 3, 8], generalized Go (basic ko), generalized chess, and generalized checkers are EXPTIME-complete in this sense.

3. The Space-Compression Theorem

Theorem 3.1 (Space compression for m_1)**.** *Let $\mathcal{G}$ be a compactly representable EXPTIME-complete game with parameter n. If $m_1 \in$ FP, then $L_{\mathcal{G}} \in$ PSPACE. Consequently* EXPTIME $\subseteq$ PSPACE*, hence* EXPTIME = PSPACE*.*

Proof. Let $\mathcal{A}$ be a polynomial-time algorithm computing m_1, with running time bounded by some polynomial $q(n)$ and workspace bounded by $q(n)$ as well. We construct a PSPACE decision algorithm $\mathcal{B}$ for $L_{\mathcal{G}}$.

Algorithm $\mathcal{B}$ on input $S_0 \in \mathcal{P}_n$.

- Initialize two position registers $S_{\text{slow}} \leftarrow S_0$ and $S_{\text{fast}} \leftarrow S_0$.
- **Loop.**
  - If $\text{term}(S_{\text{slow}}) = 1$, return $\text{sign}(\text{score}(S_{\text{slow}}))$.
  - $S_{\text{slow}} \leftarrow \text{apply}(S_{\text{slow}}, \mathcal{A}(S_{\text{slow}}))$ (one tortoise step).
  - For $i = 1, 2$: if $\text{term}(S_{\text{fast}}) = 1$, return $\text{sign}(\text{score}(S_{\text{fast}}))$; else $S_{\text{fast}} \leftarrow \text{apply}(S_{\text{fast}}, \mathcal{A}(S_{\text{fast}}))$ (two hare steps).
  - If $S_{\text{slow}} = S_{\text{fast}}$, return 0.

We verify that $\mathcal{B}$ is correct, terminates on every input, and runs in polynomial space.

Step 1: Space. At any moment, $\mathcal{B}$ stores

(a) S_{slow} and S_{fast}, each of size $O(\text{poly}(n))$;
(b) the workspace of one in-flight call to $\mathcal{A}$, of size $\leq q(n) = O(\text{poly}(n))$ (workspace is reused after each call returns);
(c) the workspace of apply, term, score, and the position-equality test, each $O(\text{poly}(n))$.

The total space usage is $O(\text{poly}(n))$. Hence $\mathcal{B}$ is a PSPACE machine.

Step 2: Termination. Consider the deterministic walk in the position graph defined by $S \mapsto \text{apply}(S, \mathcal{A}(S))$. Starting from S_0, the orbit of S_{slow} either (i) reaches a terminal position in finitely many steps, or (ii) enters a cycle of length $\ell \geq 1$ after a transient of length $t \geq 0$, by the pigeonhole principle applied to the finite state set $\mathcal{P}_n$.

In case (i), $\mathcal{B}$ returns at the terminal-position check.

In case (ii), Floyd's algorithm [4] guarantees that for some step k with $t \leq k \leq t + \ell$, $S_{\text{slow}} = S_{\text{fast}}$. To see this, note that after k steps of $\mathcal{B}$, S_{slow} has taken k steps in the orbit and S_{fast} has taken $2k$ steps. The two coincide when $k \equiv 2k \pmod{\ell}$ and both $k, 2k \geq t$, i.e., when $k \equiv 0 \pmod{\ell}$ and $k \geq t$, which happens at $k = \ell \cdot \lceil t/\ell \rceil$. Hence $\mathcal{B}$ terminates after at most $t + \ell \leq |\mathcal{P}_n| \leq 2^{p(n)}$ outer iterations.

Step 3: Correctness. We claim that $\mathcal{B}(S_0)$ outputs $\text{sign}(V(S_0))$ when both players follow the deterministic optimal-play strategy $\mathcal{A}$. Since $\mathcal{A}$ is assumed to compute m_1 correctly, the trajectory of S_{slow} traces out the optimal-play path from S_0.

In case (i) of Step 2, the trajectory reaches a terminal position S_T along the optimal path, and $V(S_0) = (-1)^{(\text{parity of } T)} \cdot \text{score}(S_T)$ in the conventional perspective shift; the sign returned by $\mathcal{B}$ equals $\text{sign}(V(S_0))$ when score is interpreted from the original side-to-move at S_0. (Standard convention: define score so that positive values favor the side to move at the terminal position; the alternation of signs in the recursion of V yields the same sign on both sides when both play optimally to a terminal.)

In case (ii), the trajectory cycles, and by Definition 2.2 the value of every position on the cycle is 0; in particular $V(S_0) = 0$, matching $\mathcal{B}$'s output.

Step 4: Conclusion. $\mathcal{B}$ is a polynomial-space algorithm deciding $L_{\mathcal{G}}$. Hence $L_{\mathcal{G}} \in$ PSPACE. Since $L_{\mathcal{G}}$ is EXPTIME-complete, EXPTIME $\subseteq$ PSPACE. Combined with the known inclusion PSPACE $\subseteq$ EXPTIME, we obtain EXPTIME = PSPACE. □

Corollary 3.2 (Specialization to generalized Go)**.** *For generalized $n \times n$ Go under the basic ko rule, $m_1 \in$ FP implies* EXPTIME = PSPACE*. Equivalently, if* EXPTIME $\neq$ PSPACE*, then $m_1 \notin$ FP.*

Proof. Apply Theorem 3.1 to Example 2.3, using the EXPTIME-completeness of the value problem from [7]. □

Corollary 3.3 (Specialization to generalized chess and checkers)**.** *For generalized $n \times n$ chess and generalized $n \times n$ checkers, $m_1 \in$ FP implies* EXPTIME = PSPACE*.*

Proof. Apply Theorem 3.1 to Example 2.4, using the EXPTIME-completeness results of [3] for chess and [8] for checkers. □

Remark 3.4 (Why two pointers and not one). A naive simulator could detect cycles by storing all visited positions in a set, but the set could grow exponentially. Floyd's two-pointer method is essential to keep the space bound polynomial. The cost is constant-factor in time per step (which is irrelevant for PSPACE membership) and a factor of two in position-pointer storage (which is also negligible).

Remark 3.5 (Memoryless optimal play is the key). The argument relies on the fact that under the basic ko rule, the optimal move at S depends only on S itself, not on the history of play (since the position encoding includes everything needed to determine legal moves). This is what allows simulation by two scalar pointers rather than by a stack of moves. Under super-ko, the position encoding would have to include all previously visited configurations, breaking the polynomial position-size assumption of Definition 2.1.

4. The Game-Length Meta-Theorem

The space-compression argument exposes a structural reason why the time-based self-reduction fails for EXPTIME-complete games: the game itself must take superpolynomial time to play out. We make this precise.

Definition 4.1 (Game length)**.** For a compactly representable game $\mathcal{G}$ and a position $S \in \mathcal{P}_n$, let $L(S)$ be the length of the optimal-play path from S until the first occurrence of either a terminal position or a repeated position. The *worst-case game length at parameter n* is $L_{\mathcal{G}}(n) = \max_{S \in \mathcal{P}_n} L(S)$.

Theorem 4.2 (Game-length meta-theorem)**.** *Let $\mathcal{G}$ be a compactly representable EXPTIME-complete game. If $L_{\mathcal{G}}(n) = \mathrm{poly}(n)$, then* EXPTIME = PSPACE*. Equivalently, conditional on* EXPTIME $\neq$ PSPACE*, every compactly representable EXPTIME-complete game has worst-case game length that is not bounded by any polynomial in n.*

Proof. Suppose $L_{\mathcal{G}}(n) \leq r(n)$ for some polynomial r. We construct a PSPACE decision algorithm for $L_{\mathcal{G}}$ by direct depth-first minimax search.

Algorithm $\mathcal{C}$ on input S:

- If $\mathrm{term}(S) = 1$, return $\mathrm{score}(S)$.
- If the search depth equals $r(n)$, return 0 (cycle limit reached).
- Enumerate $m \in M(S)$. For each m, recursively compute $V(\mathrm{apply}(S, m))$, and return $\max_m[-V(\mathrm{apply}(S, m))]$.

The recursion has depth $\leq r(n) = \mathrm{poly}(n)$. Each stack frame stores one position ($O(\mathrm{poly}(n))$ bits), one move pointer ($O(\log \mathrm{poly}(n))$ bits), and one running maximum ($O(\mathrm{poly}(n))$ bits). The total space is $O(\mathrm{poly}(n) \cdot r(n)) = O(\mathrm{poly}(n))$.

Hence $L_{\mathcal{G}} \in$ PSPACE. Since $L_{\mathcal{G}}$ is EXPTIME-complete, EXPTIME $\subseteq$ PSPACE, hence EXPTIME = PSPACE. □

Corollary 4.3. *Conditional on* EXPTIME $\neq$ PSPACE*, generalized $n \times n$ Go under the basic ko rule has worst-case optimal-play length that grows faster than every polynomial in n.*

Proof. Combine Theorem 4.2 with Example 2.3 and [7]. □

Remark 4.4 (Why this is a meta-theorem). Theorem 4.2 is independent of the specific game and of the specific reduction used to prove EXPTIME-hardness. Any future candidate EXPTIME-complete game with polynomial position size and polynomial game length would force EXPTIME = PSPACE. The contrapositive is the structural takeaway: any path to a PSPACE algorithm for an EXPTIME-complete game value must avoid direct depth-bounded minimax search, since the game tree itself is necessarily deep.

5. Discussion

5.1. **Comparison with Robson's reduction.** Robson's reduction [7] establishes EXPTIME-hardness of the value problem V by encoding alternating polynomial-space computations into Go positions. The reduction yields hardness of V, hence of m_1 via a single oracle query: given a value oracle, one trivially recovers m_1 by querying $V(\text{apply}(S, m))$ for each m. However, the converse—reducing V to m_1—is the question we address. The naive oracle reduction (simulate optimal play and read off the terminal score) fails because the game length is superpolynomial. Theorem 3.1 fixes this by replacing the polynomial time bound with a polynomial space bound.

5.2. **Comparison with PSPACE-complete games.** For PSPACE-complete games such as generalized Othello [1], the time-based self-reduction from value to best move succeeds because game length is bounded by a polynomial in the board size (at most n^2 moves for Othello). The simulator can follow the optimal-play trajectory in polynomial time, recover the terminal score, and contradict the PSPACE-hardness assumption. Our Theorem 4.2 shows why the analogous trick cannot work for EXPTIME-complete games: the game length is provably not polynomial under the standard assumption EXPTIME $\neq$ PSPACE.

5.3. **What is and is not excluded.** Theorem 3.1 excludes a polynomial-time algorithm for m_1 unless EXPTIME = PSPACE. It does *not* exclude:

- polynomial-space algorithms for m_1, which always exist by depth-bounded minimax search (although the depth bound is provided by Theorem 4.2 only conditionally);
- heuristic engines (AlphaGo, KataGo, Stockfish) that produce approximate best moves;
- polynomial-time algorithms for restricted classes of positions, e.g., Berlekamp–Wolfe [2] for decomposable Go endgames;
- polynomial-time approximation schemes whose error is allowed to grow with n.

The conditional nature of Corollary 3.2 reflects a genuine gap in current knowledge. Whether EXPTIME = PSPACE holds is open; both equality and strict containment are consistent with all unconditional theorems known to date. Our contribution is to crystallize the obstruction at exactly that gap: the best-move function for generalized Go cannot be in FP unless this collapse occurs.

5.4. **Open problems.**

(1) Strengthen the conclusion of Theorem 3.1 to $m_1 \in \text{FP} \Rightarrow \text{P} = \text{PSPACE}$ (or even P = EXPTIME). This would require a different simulation that uses both polynomial time and polynomial space; we know of no such argument.
(2) Apply Theorem 3.1 to other EXPTIME-complete games: Othello variants, Reversi variants, Hex variants under appropriate rule sets.
(3) Quantify the game-length lower bound from Theorem 4.2. The current statement is that $L_{\mathcal{G}}(n)$ is not polynomially bounded; can we improve to an explicit superpolynomial lower bound?
(4) Extend Corollary 3.2 to super-ko rules, where the position encoding (which must include game history) violates the polynomial-size assumption of Definition 2.1.

References

[1] S. Iwata and T. Kasai. The Othello game on an $n \times n$ board is PSPACE-complete. *Theoretical Computer Science*, **123**(2): 329–340, 1994.

[2] E. R. Berlekamp and D. Wolfe. *Mathematical Go: Chilling Gets the Last Point*. A. K. Peters, Wellesley, MA, 1994.

[3] A. S. Fraenkel and D. Lichtenstein. Computing a perfect strategy for $n \times n$ chess requires time exponential in n. *Journal of Combinatorial Theory, Series A*, **31**(2): 199–214, 1981.

[4] R. W. Floyd. Nondeterministic algorithms. *Journal of the ACM*, **14**(4): 636–644, 1967.

[5] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, **117**: 285–306, 1965.

[6] D. Lichtenstein and M. Sipser. GO is polynomial-space hard. *Journal of the ACM*, **27**(2): 393–401, 1980.

[7] J. M. Robson. The complexity of Go. In *Information Processing 83 (Proceedings of the IFIP 9th World Computer Congress)*, pages 413–417. North-Holland, Amsterdam, 1983.

[8] J. M. Robson. N by N checkers is EXPTIME-complete. *SIAM Journal on Computing*, **13**(2): 252–267, 1984.

Independent Researcher
Email address: lightman.chang@gmail.com