PSPACE-hardness of Two-Player Games

Ryan Catullo, Tanvi Deshpande

February 2024

Contents

Acknowledgments

The authors would like to thank Professor Li-Yang Tan for his guidance on navigating this project, as well as our peers in CS254 who reviewed and provided useful comments this paper.

1. Introduction

One of the more interesting applications of complexity theory is its application to unexpected languages. A good example of this idea is applying the theory to languages of complex two-player games, like chess.

One game of interest is Go, a board game played by over 46 million people worldwide for the past 2,500 years. Go is played on a 19×19 board, and, at a high level, involves two players claiming territory by placing stones on the board in sequence (Fig. 1).

Games such as chess and Go are typically played on constant-size boards and thus are theoretically solvable in constant space. However, even given the existence of machine-learning algorithms such as AlphaGo that have become competitive against human players, we still have yet to find practical algorithms (beyond considering every possible sequence of

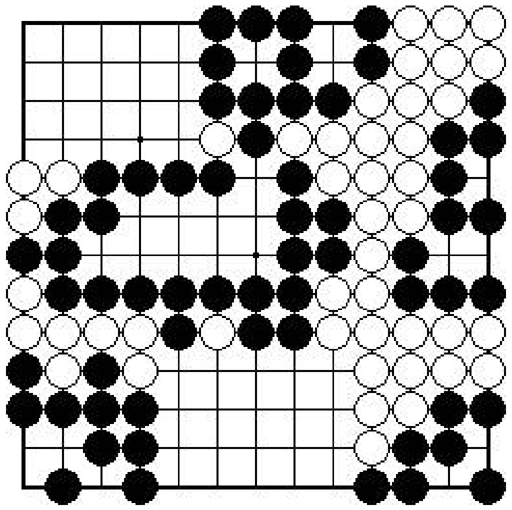


Figure 1: An example Go game.

moves) for many such games that can determine the existence of winning strategies given a game position. As the ability to play complex reasoning games is considered by many to be a crucial step towards artificial general intelligence, understanding how "difficult" these games are is not only theoretically compelling but also pertinent to real-life applications.

While the set of possible real-life game positions typically has constant size due to fixed board sizes, we can consider generally for a game G the language consisting of inputs (like the position of pieces on a chess board or stones on a Go board) of *variable size*, such that Player 1 has a winning strategy.

Some natural questions arise regarding these languages. What time or space complexity classes do they lie in? Are any of them complete with respect to that class? This is akin to asking "how computationally hard is it to develop a winning strategy for these games?" Such questions can help us understand the difficulty of creating algorithms for these games in real-life settings, and, at the very least, lead to compelling questions and results—such as which of these games are PSPACE-hard or complete.

In this project, we will examine several two-player games, including TQBF, Geography, and Go. Amazingly, each of these games is either PSPACE-hard or complete (depending on whether the number of moves in the game is polynomially bounded: in some games, like chess and Go, the number of moves can be exponential).

The general strategy for doing this is by reducing the problem to G_ω, which is the language of Boolean formulas where, when players take turns assigning values to variables, Player 1 has a strategy for assigning variables such that the formula is satisfied after all variables are assigned. This is like playing a game on a Boolean formula in the following sense. Given a formula φ on variables $x_1 \dots x_n$, a winning strategy is the existence of some assignment for x_1 such that for all assignments x_2, there exists x_3 such that for all x_4, etc... such that $\varphi(x_1, \dots, x_n) = 1$.

Essentially, Player 1 plays the odd variables and Player 2 the even ones, and for any choice Player 2 makes, Player 1 should be able to make a choice that leads to a satisfying assignment. In the language of Boolean operators,

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \dots x_n \colon \varphi(x_1, \dots, x_n) = 1$$

This language is an analogue to SAT for PSPACE (the canonical PSPACE-complete problem); it is easy to see how this could, intuitively, be extended to all two-player turn-based games: if we are considering positions for which Player 1 has a winning strategy, they should have a valid (winning) response for all possible moves played by Player 2.

1.1. **Definitions**

We start by standardizing the definitions and terminology that we use throughout the paper.

We restrict our attention to two-player games, with players denoted by Player 1 and Player 2. Let G be a game: constraints on the set of inputs, and a set of rules that Player 1 and Player 2 must adhere to. We use the term *general game* to refer to all possible inputs over a rule set, for example all possible positions of pieces on the chess board is a general chess game. By a *specific game*, we mean one where a particular input A is provided.

Definition 1.1 For a general game G, we define

$$L(G) = \{A : \text{There is a winning strategy for Player 1 using the rules of } G \text{ on } A,\}$$

where A is a specific game input.

We consider games in which there are no draws, meaning that $\overline{L}(G)$ is the set of inputs for a game on which there is a winning strategy for Player 2.

Like NP-complete problems, PSPACE-complete problems are problems in PSPACE that every other problem in PSPACE can be polynomial-time reduced to.

Definition 1.2 (Log-space reducibility) Given two problems A, B, we say that B is log-space reducible (or log-reducible) to A if there is some log-space function f such that for all w, $w \in B \iff f(w) \in A$.

Remark Log-space reducibility implies polynomial-time reducibility; therefore, showing that a given game is PSPACE-complete with respect to log-space reductions is a stronger condition than showing PSPACE-completeness (with poly-time reductions).

2. Two-player Formula Games

2.1. **True Quantified Boolean Formulas**

As was described in the introduction, G_ω is a game played using a Boolean formula ϕ and a list of its variables $x_1, \dots, x_n$. Players take turns assigning each subsequent variable to a value, and Player 1 wins if, at the end, the given assignment of variables satisfies ϕ.

Formally, let $x_1, \dots, x_n$ be Boolean variables for $n \geq 1$ and define

$$G_n = \{\varphi \mid \exists x_1 \forall x_2 \exists x_3 \dots Q_n x_n \text{ such that } \varphi(x_1, \dots, x_n) = 1\}$$

Here, $Q_n = \exists$ if n is odd and $Q_n = \forall$ if n is even. Additionally, φ is assumed to be a Boolean formula in $x_1, \dots, x_n$ where the only operations used are $\vee, \wedge, \neg$. More, generally, we define the following set.

$$G_\omega = \bigcup_{n=1}^{\infty} G_n \tag{1}$$

This language is also called TQBF, which stands for true quantified Boolean formula, or appropriately QSAT, since it is the analogue to SAT for formulas with quantifiers. We show that, in further analogy to SAT, this problem is the "canonical" PSPACE-complete problem. Note that any quantified Boolean formula is in G_ω by introducing "dummy variables." For example, the formula $\exists x_1 \exists x_2 \varphi(x_1, x_2)$ is the same as $\exists x_1 \forall y_1 \exists x_2 \varphi(x_1, x_2) \in G_\omega$, so we can think of G_ω as just all quantified Boolean formulas.

We want to show first that G_ω is in PSPACE in order to show completeness. Thus, we have the following theorem.

Theorem 2.1 $G_\omega \in$ PSPACE.

Proof. It suffices to give a polynomial space algorithm that determines G_ω. First, we can reduce any quantified Boolean formula φ to one in CNF, since SAT $\equiv_P$ 3SAT and this is just a subset of that problem and polynomial time reducibility preserves polynomial space algorithms.

Suppose φ has n variables and m clauses. Note that each clause has at most n variables, so we wish to show the satisfiability of φ can be decided in poly(n, m) space. Let $S(\varphi, i)$ be the space required to determine

$$\varphi_i(\varepsilon_1, \dots, \varepsilon_{n-i}) = Q_{n-i+1} x_{n-i+1} \cdots Q_n x_n \varphi(\varepsilon_1, \dots, \varepsilon_{n-i}, x_{n-i+1}, \dots, x_n)$$

That is, φ where we determine the first $n - i$ variables to be $x_i = \varepsilon_i$. Here, as before, $Q_j = \exists$ if j is odd and $\forall$ if j is even. The total space complexity is then $S(\varphi, n)$. In the base case, where $i = 0$, we have fixed $x_1 = \varepsilon_1, \dots, x_n = \varepsilon_n$ so we just have to evaluate an n-variable, m-clause Boolean formula, which can be done in space $S(\varphi, 0) = O(mn)$.

For the inductive step, suppose $\varphi_0, \dots, \varphi_{i-1}$ are all computable in polynomial space. Then we recursively evaluate φ_i with $\varepsilon_{n-i} = 1$ and $\varepsilon_{n-i} = 0$. If $Q_{n-i} = \forall$, then φ is true if and only if φ_{i-1} is true with both $\varepsilon_{n-i} = 1$ and $\varepsilon_{n-i} = 0$, and if $Q_{n-i} = \exists$ then it is true if and only if at least one of the recursive calls is true. However, we reuse the space we used to compute φ_i with $x_{n-i} = 1$ to compute φ_i with $x_{n-i} = 0$. Therefore, the space complexity satisfies

$$S(\varphi, i) = S(\varphi, i - 1) + O(mn)$$

since it takes $O(mn)$ space to write down the partially evaluated formula for φ_{i-1} with our choices of $\varepsilon_1, \dots, \varepsilon_{n-i}$. Therefore, $S(\varphi, i) = O(mn^2)$ is polynomial in the input length, so $G_\omega \in$ PSPACE. □

For the next part, we wish to show completeness. We will follow the proof given in [SM73]. The general idea is as follows. Suppose $A \in$ PSPACE, so that A is determined by some deterministic Turing machine M with $O(n^k)$ space on inputs of length n for some $k \geq 1$. The key insight is that configurations of the Turing machine can be encoded as Boolean formulas.

Let C, C' be two configurations of M, and $t \in \mathbb{N}$. We want to define $\varphi_t(C, C')$ to be a quantified Boolean formula which is true if and only if M can go from C to C' in at most 2^t timesteps. The reason why is the following. If C_{start} is the starting configuration, and C_{accept} is the accepting configuration, then there is some T bounded by a function in n which is the most number of steps it takes to get from C_{start} to C_{accept}. Then $x \in A$ if and only if $\varphi_T(C_{\text{start}}, C_{\text{accept}}) = 1$, which gives us our reduction $A \leq_P G_\omega$.

Theorem 2.2 G_ω is PSPACE-complete with respect to poly-time reductions.

Proof. As before, let $A \in$ PSPACE be determined by M, with alphabet Γ, states Q, transition function δ, a single read tape with input x written on it of length n, and a single write tape. We will define $\varphi_t(C, C')$ recursively, which is a quantified Boolean formula satisfied if and only if M transitions from C to C' in at most 2^t timesteps.

A configuration C describes the contents of M at a timestep i on an input x. Specifically, $C = (q, P, \gamma_1, \dots, \gamma_{s(n)})$ where $s(n) = O(n^k)$ is the space of M, $q \in Q$ is the state at time i, P is the position of the head at time i, and $\gamma_1, \dots, \gamma_{s(n)}$ are the symbols on M's write tape at time i. Note that $\varphi_0(C, C')$ is 1 if and only if δ applied to M with C's configuration yields C''s configuration, i.e. they are 1 timestep away. It will be a conjunction of a constant number of CNF formulas, each of which is polynomial length in n and computable in polynomial time.

How do we check that C' is one timestep away from C? First, we want that for $j \neq P$, we have $\gamma_j = \gamma'_j$ since transitioning only writes to the part of the tape that the head points to, i.e. P. In the case $j = P$, we want that the state is moved to q' when δ is applied to q, x_P, γ_P, where x_P is the symbol on the read-only input tape at position P, and that $\gamma_{P'}$ is written on the tape, and the head moves to P'. We can represent these requirements as Boolean formulas which are computable in polynomial time, since the same result which is used in the proof of Cook-Levin applies here.

For example, if we represent γ_j and γ'_j as bitstrings, which we can do in a constant number of bits since Γ is finite, and let x_i and y_i be Boolean variables representing the bits of γ_j and γ'_j, then

$$F(\gamma_j, \gamma'_j) = \neg(x_0 \oplus y_0) \wedge \dots \wedge \neg(x_r \oplus y_r) = 1$$

if and only if $\gamma_j = \gamma'_j$, and this can be expressed as a CNF formula. Similarly, since

δ, q, x_P, γ_P can all be specified in a constant number of bits, there is a constant-length CNF formula F such that $\delta(q, x_P, \gamma_P) = q'$ if and only if $F(q', \delta, q, x_P, \gamma_P) = 1$, and the same holds for $\delta(q, x_P, \gamma_P)$ writing $\gamma_{P'}$ on the tape if and only if $F(\gamma_{P'}, \delta, q, x_P, \gamma_P) = 1$.

Similar logic holds for computing P', except now it is encoded by $\log s(n) = O(\log n)$ bits so $F(P', \delta, q, x_P, \gamma_P)$ is length $O(\log n)$. By taking the conjuction of all of these formulas, we obtain a CNF formula $\varphi_0(C, C')$ of length $O(n^k)$ such that $\varphi_0(C, C') = 1$ if and only if C is 1 timestep away from C'.

Now for the recursive step, the key is to guess some state in the middle of C and C'. Namely, we let φ_t be the following quanitied Boolean formula.

$$\varphi_t(C, C') = \exists C'' \varphi_{t-1}(C, C'') \wedge \varphi_{t-1}(C'', C')$$

That is, there is some configuration C'' that is 2^{t-1} timesteps away from C, and 2^{t-1} timesteps away from C', such that C is $2 * 2^{t-1} = 2^t$ timesteps away from C' by definition. The issue is that the length of this formula does not grow like a polynomial in n. Instead, we make the following equivalent definition.

$$\begin{aligned} \varphi_t(C, C') = \exists C'' \, \forall D_1 \, \forall D_2 & \text{ such that} \\ & [(D_1 = C \wedge D_2 = C'') \vee (D_1 = C'' \wedge D_2 = C')] \implies \varphi_{t-1(D_1, D_2)} \end{aligned}$$

Why is this equivalent? Note that the above implication is true if and only if *both* implications below hold.

$$\begin{aligned} (D_1 = C \wedge D_2 = C'') &\implies \varphi_{t-1}(D_1, D_2) \\ (D_1 = C'' \wedge D_2 = C') &\implies \varphi_{t-1}(D_1, D_2) \end{aligned}$$

Therefore, this is saying if $D_1 = C$ and $D_2 = C''$, then there is a path of length 2^{t-1} from C to C'', and if $D_1 = C''$ and $D_2 = C'$ then there is a path of length 2^{t-1} from C'' to C'. By the same logic as before, this is true if and only if C'' is the midpoint, i.e. there is a path of length 2^t from C to C'. Further, the length of this formula

$$L(t) = L(t-1) + O(n^k) = O(tn^k)$$

where the latter part is the length required to specify the configurations C, C'', C'.

Now we can consider what the maximum value of T is, i.e. the most steps it takes to get from C_{start} to C_{accept}. There are $O(n^k)$ bits of memory used by M, and therefore $2^{O(n^k)}$ different configurations. Therefore, there at most that many steps between start and accept configurations if there is such a path for a given input x, and so it follows that $T = O(n^k)$. That is, the length of $\varphi_T(C_{\text{start}}, C_{\text{accept}})$ is $O(n^{2k})$ which is polynomial in n. Therefore, we have shown the existence of a function f such that $x \in A$ if and only if $f(x) \in G_\omega$, namely $f(x) = \varphi_T(C_{\text{start}}, C_{\text{accept}})$, and further that f is computable in polynomial time. It follows that G_ω is PSPACE-complete. □

Recall that, for a game G, $L(G)$ is the language of inputs for G such that there is a winning strategy for Player 1. This is where the notation for G_ω comes in, since any Boolean formula φ can be turned into an input for G_ω where Player 1 chooses the odd variables and Player 2 chooses the even ones. We denote this language by $L_\omega = L(G_\omega)$, and the above theorem tells us in fact that L_ω is PSPACE-complete.

2.2. **Geography**

In this subsection, we introduce *Geography*, a game inspired by the verbal game in which players take turns naming cities, with each subsequent city's first letter matching the previous city's last letter. For example, the following would be a valid Geography game.

$$\text{Austin} \to \text{Nashua} \to \text{Albany} \to \text{York} \to \text{Kansas} \to \ldots$$

The game continues until one of the players can't name a city, and whoever can't name a city loses.

We can generalize the game in the following way. For us, Geography is a game played on a directed graph G with a given starting vertex s, which we represent by an ordered pair (G, s). Players take turns moving from the current vertex along an edge to a new vertex, where each directed edge can be used exactly once until one player has no valid moves (in which case they lose). In the analogy with the above game, for instance, G would consist of vertices labeled with city names, where we have a directed edge from City A to City B if the first letter of B is the last letter of A.

We first want to show that L(GEOGRAPHY) is in PSPACE by demonstrating some algorithm for it. This is essentially the same as the algorithm given for G_ω, and the reason is that it applies in full generality to all games of this form.

Theorem 2.3 $L(\text{GEOGRAPHY}) \in \text{PSPACE}$.

Proof. Let (G, s) be our graph and starting vertex. Let n be the largest integer such that there is a finished game

$$s \to v_1 \to v_2 \to \ldots \to v_n$$

where v_n has no outgoing edges. There is some finite n for which this is true since the path is of length at most $|E|$ if no two edges are used twice. As per usual, Player 1 has chosen v_i for i odd and Player 2 has chosen for i even. If n is odd, we want to verify Player 2 has no more moves, and if n is even we want to verify Player 1 has no more moves.

Suppose without loss of generality that n is odd. Then to verify that Player 1 has won, we just need to check that there are no outgoing edges from v_n that are not already used. This can be done in polynomial space, since there are $n = O(|E|)$ edges used and we can loop through all edges outgoing from v_n and check if that edge is already used, hence the total algorithm runs in $O(|E|)$ space. Therefore, $S(G, n) = O(|E|)$ where $S(G, i)$ is the space required to compute whether Player 1 wins with a path of length i already decided.

Now we proceed by induction to compute $S(G, 0)$, the space required to determine if Player 1 wins. Suppose we have a sequence of moves

$$s \to v_1 \to v_2 \to \ldots \to v_i$$

where Player 1 plays odd indices and Player 2 even indices, and suppose without loss of generality that i is odd. Then we can compute all outgoing unused edges from v_i in $O(|E|)$ space as before. If there are none, Player 1 wins. Otherwise, for every outgoing edge, add it to the path and compute whether Player 1 wins recursively, which can be done in space $S(G, i+1)$. If none of the outgoing paths result in Player 1 winning, then Player 2 wins. The space then satisfies

$$S(G, i) = S(G, i+1) + O(|E|)$$

since we can reuse the space for every call to a path of length $i+1$, and therefore $S(G, 0) = O(|E|^2)$ which is polynomial in the length of the input. □

We conclude this section with a proof that $L(\text{GEOGRAPHY})$ is in fact PSPACE-complete, using a reduction from L_ω.

Theorem 2.4 $L(\text{GEOGRAPHY})$ is PSPACE-complete.

Proof. We will follow the proof given by [Sch78], which shows that $L(\text{GEOGRAPHY})$ is log-space complete in PSPACE (a stronger condition than showing PSPACE-complete using polynomial-time reductions), by showing a reduction from L_ω, i.e. $L_\omega \leq_L L(\text{GEOGRAPHY})$.

Let φ be some input to L, and assume without loss of generality it is in CNF with m clauses $\varphi = C_1 \wedge \ldots \wedge C_m$ where each C_i is a disjunction of literals, in n variables $x_1, \ldots, x_n$, where n is odd (we can always add dummy variables). Define the directed graph $G = (V, E)$ with vertices equal to the following set.

$$V = \{x_i, \neg x_i, u_i, v_i \mid 1 \leq i \leq n\} \cup \{u_{n+1}\} \cup \{y_k \mid 1 \leq k \neq m\}$$

For edges, we let E be the following.

$$\begin{aligned} E = \{&(u_i \to x_i), (u_i \to \neg x_i), (x_i \to v_i), (\neg x_i \to v_i), (v_i \to u_{i+1}) \mid 1 \leq i \leq n\} \\ \cup \{&(u_{n+1} \to y_k) \mid 1 \leq k \leq m\} \cup \{(y_k \to x_i) \mid x_i \text{ occurs unnegated in } C_k\} \\ \cup \{&(y_k \to \neg x_i) \mid x_i \text{ occurs negated in } C_k\} \end{aligned}$$

Figure 2 gives a good idea of what the graph looks like. The x_i and $\neg x_i$ represent choices of x_i to be 1 or 0, the u_i and v_i are auxiliary nodes, and the y_k represent the k clauses. We start the game of Geography on u_1. Player 1 moves to x_1 or $\neg x_1$, which corresponds to $\exists x_1$ being either 1 or 0 in φ. Then Player 2 has to move to v_1, and Player 1 has to move to u_2. Now Player 2 can either move to x_2 or $\neg x_2$, corresponding to $\forall x_2$ in φ. The game continues in this way until we get to u_{n+1}. Since we assumed n is odd, it is Player 2's turn.

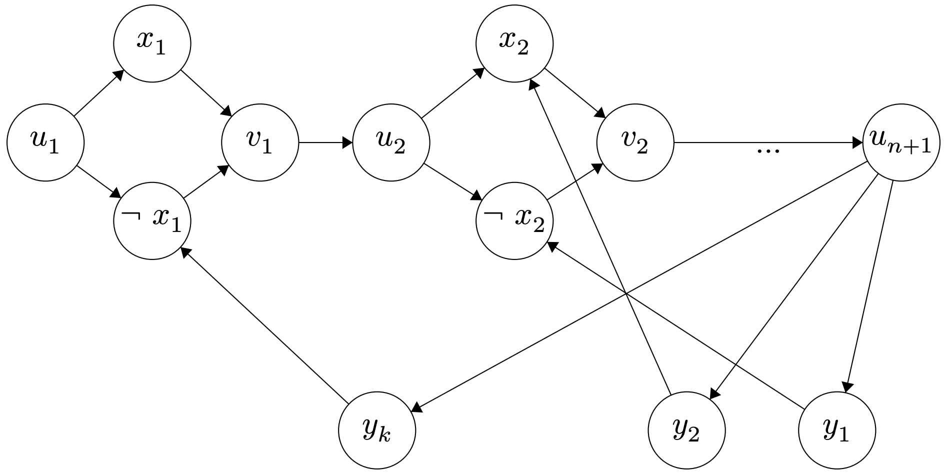


Figure 2: Graph associated to quantified Boolean formula φ. Here, the arrows departing from y_k, y_2, y_1 show us that $C_k = \neg x_1 \vee ...$, $C_2 = x_2 \vee ...$, $C_1 = \neg x_2 \vee ...$

Now suppose φ is not satisfiable, and let C_k be a clause which is not satisfied. Then Player 2 can win by moving from u_{n+1} to y_k, since the paths out of y_k go to x_i or $\neg x_i$ depending on which appears in C_k. Suppose x_i appears in C_k, and Player 1 chooses the path $y_k \to x_i$. Then since C_k is not satisfied, x_i is false which means we chose the path

$$u_i \to \neg x_i \to v_i \to u_{i+1}$$

Then Player 2 can choose $x_i \to v_i$ since this path has not been chosen yet, and Player 1 has no moves since the only outgoing edge is $v_i \to u_{i+1}$ which has already been used. Thus, Player 2 wins.

Now suppose φ is satisfiable. Then Player 2 will move from $u_{n+1} \to y_k$ for some k. Then by assumption, C_k is true so there is some x_i or $\neg x_i$ which is true in C_k. Suppose it is x_i, so that x_i appears in C_k and is set to true. Then there is an edge $y_k \to x_i$, so Player 1 chooses this edge. Then since x_i was set to true, the previous paths include

$$u_i \to x_i \to v_i \to u_{i+1}$$

and the only outgoing edge of x_i is to v_i. Therefore, Player 2 has no moves, so Player 1 wins. □

This is our first amazing example of a real game that is complete in PSPACE. In the next section, we use this example to tackle a slightly more complex game of Go.

3. PSPACE-hardness of Go

Lastly, we show that Go is PSPACE-hard, following the proof in [LS80]. This is done through a reduction from Planar Geography (a variant of Geography), by encoding planar graphs as Go positions.

3.1. **Rules**

We begin by describing the rules of Go at a high level; for more detailed explanation, see here.

Go involves two players, White and Black, placing stones on a (typically 19×19, but for our purposes, $n \times n$) grid of points. Black moves first, meaning they are Player 1.

Two stones are adjacent if they occupy lines one horizontal unit over or vertical unit over, and two diagonal stones are not adjacent (thus, every stone has 4 adjacent spaces). If a stone is surrounded on all adjacent sides by the opposing stones, it is captured. Similarly, if a player's stones surround a contiguous region of the board, that territory temporarily belongs to them. However, if one player's stones ever surround a group of the other's stones and every empty space on the inside is filled, the surrounded stones are removed, meaning that territory belonging to one player could no longer belong to them. Also, players are not allowed to "sacrifice" a piece: stones that they place must be adjacent to at least one empty space ("liberty"), and they can't place it in a surrounded area, such that their piece is immediately thereafter removed, unless they are capturing the other player's stones.

Therefore, the only way for a player to guarantee that territory belongs to them is through *eyes*, which are empty points surrounded by one player's stones. For example, in Figure 3, Black has a group of stones with two eyes; observe that White cannot surround any of Black's territory, since they need to have stones in both empty points to do so, and this is impossible. White cannot place a stone in either eye since it would be captured immediately (sacrificing, which is not allowed), and even if it surrounded Black's stones it could not place a white stone since there is another liberty within the group, so White would be sacrificing a piece. Therefore, this territory is safe.

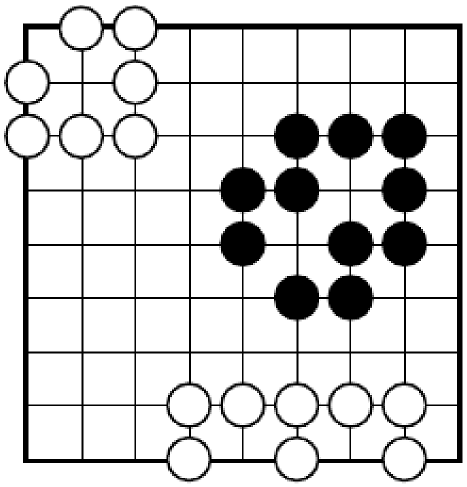


Figure 3: Eyes on a Go board

Unlike previously discussed games, the game ends by consensus (when both players agree to end the game), and each player's score is calculated as the number of empty points within the territory that belongs to them, minus the number of their stones that were captured. The player with the highest score wins. Though Go technically ends by agreement, for many practical (and our) purposes, this is when one player has definite control over most of the territory (through the use of eyes), so that the other player can't gain more territory than them.

Remark Note that because a game of Go ends by agreement of the two players, rather than whenever one player can no longer move, as in games we have previously considered, the number of moves and possible game positions is not polynomially bounded. Therefore, rather than showing that Go is PSPACE-complete, we only show that it is PSPACE-hard.

To show PSPACE-hardness, we ensure that the Go position we construct as part of our reduction is a position for which one player can control the majority of the territory in a polynomial number of moves.

3.2. **Hardness**

We will show that Go is PSPACE-hard through a reduction from *Planar* Geography, which is played on bipartite graphs with maximum degree 3. This allows us to restrict to a small set of cases while constructing Go positions corresponding to specific Geography games.

Remark Showing that PLANAR GEOGRAPHY is PSPACE-complete involves making Geography graphs planar by replacing the points where their edges cross with a cluster of several new nodes, such that there are exactly two paths through the cluster, each corresponding to the edges in the original graph. We omit the proof for brevity, but it can be found in section 4 of [LS80].

Theorem 3.1 $L(\text{GO})$ is PSPACE-hard.

Proof. We show this by representing a Planar Geography game as a Go position. We can consider Planar Geography games that correspond to inputs to TQBF, since this would be equivalent to showing a reduction from TQBF to Planar Geography, then to Go. Since TQBF and Planar Geography are PSPACE-complete, this suffices to show that Go is PSPACE-hard. For our construction, we will create a board position where there is one large region of territory that definitely belongs to White (as in, it has two eyes), as well as another (larger) region that is currently White's, but could be captured by Black; attached to this region, we have a region of stones corresponding to the Planar Geography game. We will show that, in this particular position, Black wins by capturing the large contested region of territory iff Player 1 has a winning strategy in the Geography game.

Since we are only considering graphs with maximum degree 3, and because vertices with only incoming edges result in an instant loss for the next player (and vertices with only outgoing edges can never be reached by a Geography game), we need only restrict to several cases, given in Table 1, along with the corresponding Go configuration. In particular, any vertex consisting of a singular incoming edge and outgoing edge—essentially a "dummy vertex"—was meant to correspond to flip to the player making the next assignment to a literal. However, in Go, we distinguish between Player 1 and Player 2's choices by constructing a different position (note the distinction between (a) and (b) in the table), allowing us to collapse dummy vertices.

Through these constructions encoding various types of vertices (referred to as *pipes* and *junctions*, we construct a Go position corresponding to the input Planar Geography game, away from the two large existing pieces of territory. In order to win, Black must capture the contested piece of territory, which they have almost surrounded, except for the portion corresponding to the Geography game. Also, in each junction, White has two eyes: they want to connect the contested territory to these eyes to guarantee that it belongs to them. We illustrate this in Figure 4; the portion of the Geography graph corresponding to the first literal (x_1) to be assigned is the junction that is attached to the contested territory.

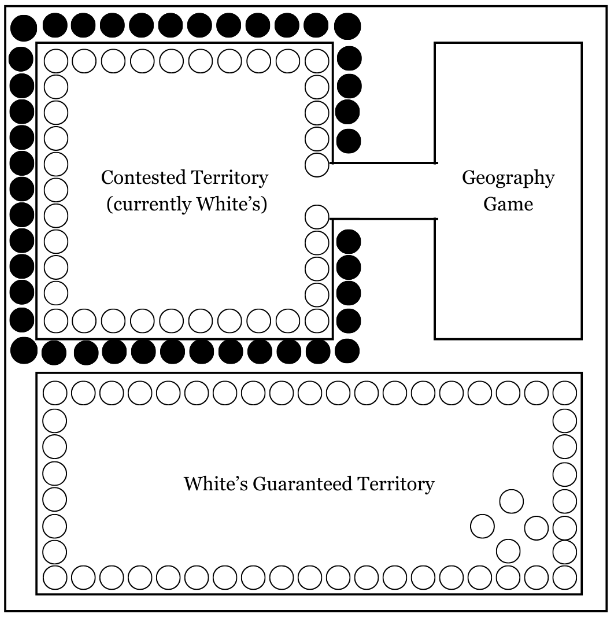


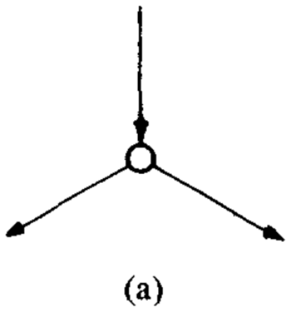
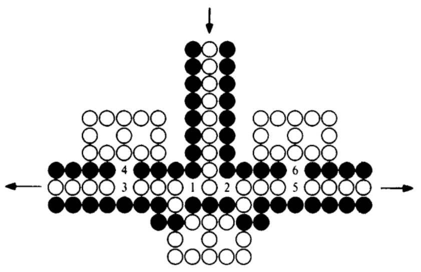
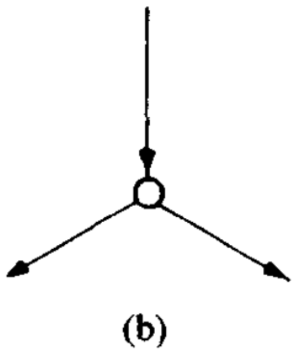
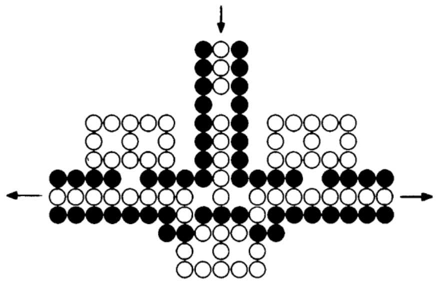
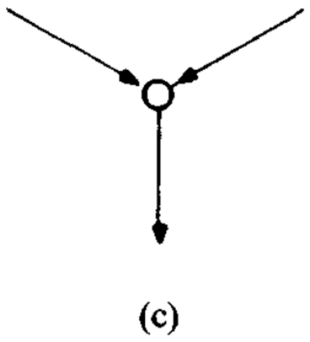
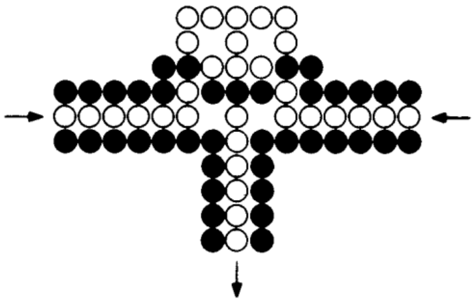
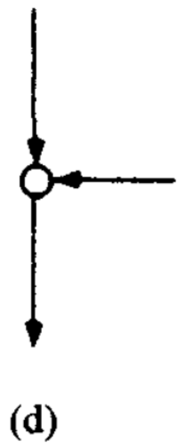
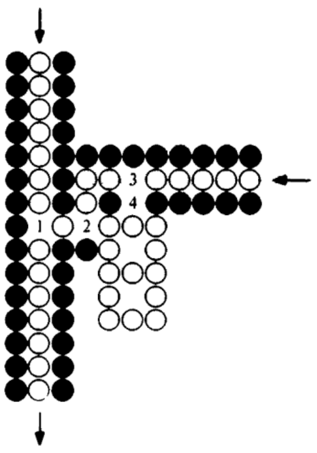
Figure 4: Construction of the Go board from a Geography game.

Now, we will show that Black can capture the contested territory iff Player 1 has a winning strategy in the Geography game, through induction. Assume that, when we enter a new junction, Black has almost surrounded the contested territory (as in, for example, Figure 4) and that it's White's turn.

Consider case (a) in Table 1, corresponding to Player 2 assigning a literal. In the context of Go, this means they choose whether to go right or left out of the junction. Suppose without loss of generality that they go left. We claim that the only valid sequence of moves is 1 (White)—2 (Black)—3 (White)—4 (Black). Note that this results in the players entering the junction to the left, again with Black almost surrounding the contested territory and it being White's turn, satisfying the inductive hypothesis.

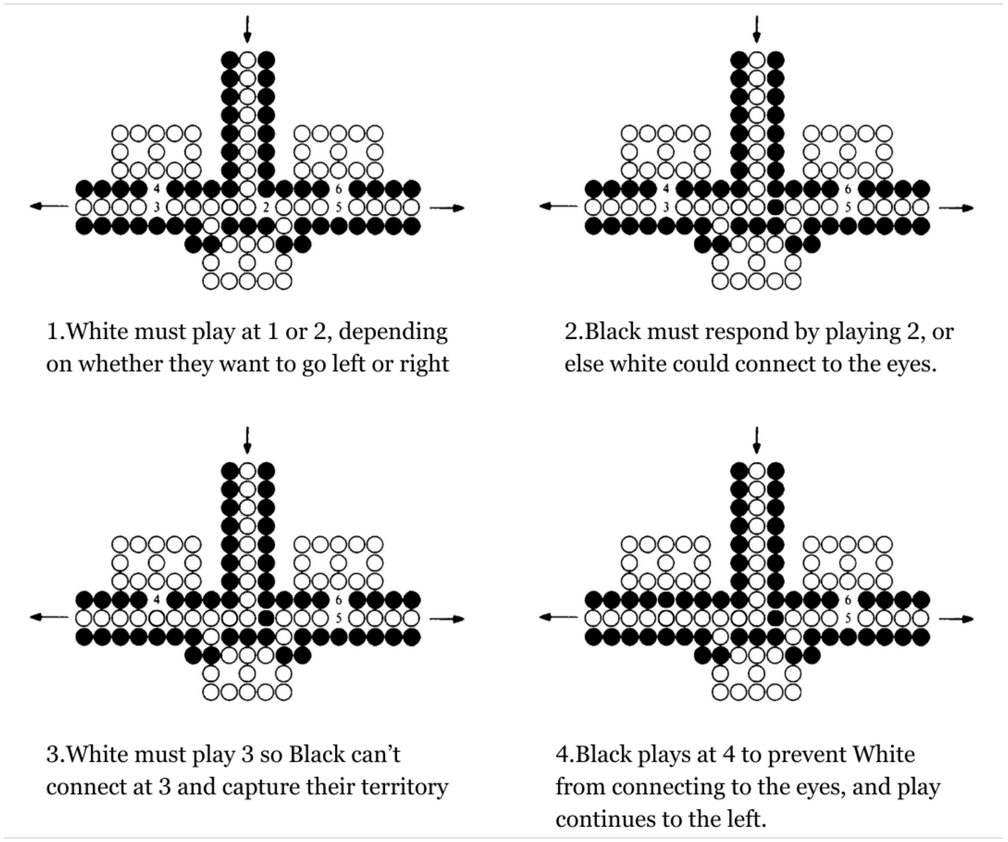
Further, this is the only valid sequence of moves. First, White's first move has to be at position 1 or 2 in the diagram. If not, Black could play at 2, meaning White has to block

Table 1: Geography vertices and corresponding Go positions [LS80]

Type of vertex	Geography diagram	Go junction
Player 2 choice		
Player 1 choice		
Joint intersection		
"Test" intersection		

them at 1, leading to Black winning by playing 3 (if White had played 3 earlier, then Black could again play at 2, forcing White to play at 1, then win by playing 5.)

After White's first move, Black has to play at point 2, or else White could play there and connect to the two eyes in the junction. By similar logic, White is forced to play at 3, and Black is forced to play at 4. This path of play is illustrated in Figure 5. Cases (b) and (c) can also be shown through a similar argument. Note that while (a) corresponds to White choosing to go left or right (analogous to assigning a literal), (b) corresponds to Black making the choice.



1.White must play at 1 or 2, depending on whether they want to go left or right

2.Black must respond by playing 2, or else white could connect to the eyes.

3.White must play 3 so Black can't connect at 3 and capture their territory

4.Black plays at 4 to prevent White from connecting to the eyes, and play continues to the left.

Figure 5: The only valid sequence of moves through the junction in case (a).

Lastly, we show case (d). If the players pass through the junction at the top first, then White wins: White would first play at 1, Black would then block them at 2, and then White must play 3, resulting in Black winning by playing 4. However, if the players pass through the right first, then we can observe that White plays 3, and regardless of what Black plays, can win by playing 2 or 4.

We have shown that, through this construction given a Geography game corresponding to an input to TQBF, we obtain a Go board for which Black can win iff Player 1 can win the Geography game. Hence we have completed our reduction, and have shown that Go is PSPACE-hard. □

References

[LS80] David Lichtenstein and Michael Sipser. "GO Is Polynomial-Space Hard". In: *J. ACM* 27.2 (Apr. 1980), pp. 393–401. ISSN: 0004-5411. DOI: 10.1145/322186.322201. URL: https://doi.org/10.1145/322186.322201.

[Sch78] Thomas J. Schaefer. "On the complexity of some two-person perfect-information games". In: *Journal of Computer and System Sciences* 16.2 (1978), pp. 185–225. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(78)90045-4. URL: https://www.sciencedirect.com/science/article/pii/0022000078900454.

[SM73] L. J. Stockmeyer and A. R. Meyer. "Word problems requiring exponential time(Preliminary Report)". In: *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*. STOC '73. Austin, Texas, USA: Association for Computing Machinery, 1973, pp. 1–9. ISBN: 9781450374309. DOI: 10.1145/800125.804029. URL: https://doi.org/10.1145/800125.804029.