Combinatorial Games and Computational Complexity

Anirudh Bharadwaj

August 2021

Abstract

In this paper, we briefly introduce the idea of computational complexity and its applications in Combinatorial Game Theory. We will examine several games, and their respective complexities such as the Redwood Furniture Problem derived from the game of Hackenbush. This provides an exposition on the idea of complexity when it comes to Combinatorial Games.

1 Introduction and Definitions

Computational complexity is a theoretical subfield of computer science which aims at classifying problems into how hard they are to solve given a certain amount of combinatorial objects. In the context of CGT, one such problem would be how "hard" it is to determine the winning strategy of a game.

Now, we introduce the concept of complexity classes. There are quite a few complexity classes, but the notable classes are $\texttt{P}, \texttt{NP}$, and $\texttt{PSPACE}$. It is well known that $\texttt{P} \subseteq \texttt{NP} \subseteq \texttt{PSPACE}$, meaning that problems that are in $\texttt{P}$ are also in $\texttt{NP}$ by the definition of a subset. We now define the classes mentioned previously.

Definition 1. *The complexity class P contains problems which can be solved in polynomial time by a Deterministic Turing Machine.*

Definition 2. *The complexity class NP contains problems which can be solved in polynomial time by a Non-Deterministic Turing Machine. More formally, this can be written as $\texttt{NP} = \bigcup_{k \in N} \texttt{NTIME}(n^k)$*

Definition 3. *The complexity class $PSPACE$ contains the problems which can be solved by Turing Machines in Polynomial **space**. Similar to the NP class, this can be written as $\texttt{PSPACE} = \bigcup_{k \in N} \texttt{SPACE}(n^k)$.*

Definition 4. *If a given problem is class-X hard, it is at least as hard as the problems in class-X.*
*If the aforementioned problem is both class-X hard, and is located in class-X, it is **class-X complete**.*

This definition is important as it gives us an important idea - a lower bound for a problem's complexity. Additionally, note that if a problem is class-X complete, it *must* be class-X hard.

2 Computational Complexity of Go

So far, we have given a simple overview of complexity in a general sense. Now, we expound upon the subject with respect to combinatorial games.

The first game we analyze is Go. Before we examine the game, it is imperative to understand a standard way to prove that a certain game belongs to a certain complexity class. This method is known as reduction. The general idea is starting with a conjectured complexity class that a game could belong to. We then start with a problem known to belong to that complexity class. From there, we reduce said problem consecutively until we are able to reduce it to the game we want to prove.

A simple exposition of proving Go's complexity will now be given.

Theorem 1. *Go is* `PSPACE` *hard*

Proof. We start with the Quantified Boolean Formula Problem. Quantified Boolean Formula (QBF) $= \{Q_1 v_1 Q_2 v_2 ... Q_n v_n F(v_1, v_2, ..., v_n)\}$ where $Q_1 = \exists$ and $Q_n = \forall$ and $Q_i \neq Q_{i+1}$. F is a boolean function using the variables.

Lemma 1. *QBF is* $\texttt{PSPACE} - complete$

We use this lemma without proof in this paper. Readers may refer to *A.R. Meyer, L.J. Stockmeyer, Word Problems Requiring Exponential Time* for a detailed proof.

Generalized Geography .

We assume prior knowledge of this game. We will use this game as our starting point of reduction after being given a QBF formula.

Theorem 2. *GG is* $\texttt{PSPACE} - complete$

A sketch of the proof is as follows

Proof. Given a QBF formula X, we can construct a graph corresponding to the Generalized Geography game of X, which we will refer to as GG(X). The graph is constructed as follows. Each variable gets its own diamond structure, and each clause in the F function is represented by a node with each node pointing to all variables within that clause.

The first part of the game is passing through all the diamond structures until the game arrives at the last named node
There are two players: if-any and the for-all player. One player will choose the variables corresponding to the $\forall$ and another player will choose the variables corresponding to the $\exists$.
Without loss of generality, suppose that the for-all player is the player that is responsible for determining which node to visit after the last variable node. Note once again that these nodes represent *clauses* rather than variables now. By the definition of $\exists$, if even one node is open, then the if-any player wins. Thus, the winning strategy for for-all player is to make sure that the node they visit has a dead end.

This should seem familiar. Indeed, the if-any player's strategy can be informally noted as discovering the variables that will satisfy the QBF.

Thus we have shown that GG is also PSPACE-complete. QED. We have therefore completed the reduction from QBF to GG.

Planar GG .

Lemma 2. *Planar GG is PSPACE − complete*

Proof. There exists a substitution which can transform every GG game into a Planar GG game. The substitution is as follows: We do not dive deeper into

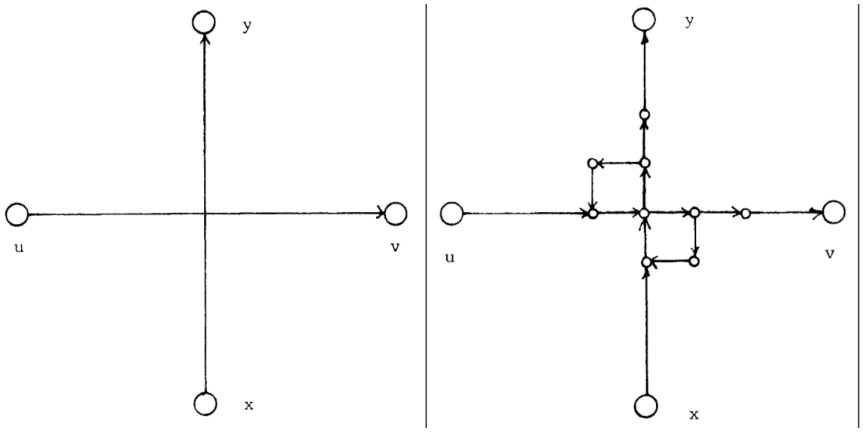


Figure 1: The substitution needed to transform a GG to PGG

the proof and construction of this equivalent PGG game, but readers may refer to *D. Lichtenstein, M. Sipser, GO is Pspace Hard* for a more detailed proof.

Go .

The idea behind this final reduction is to show that the game Go can be used to solve any PGG problem, thus putting PGG as a lower bound on complexity for Go. Trivially, this will mean that Go is at least as hard as PGG which will provide us the result wanted.

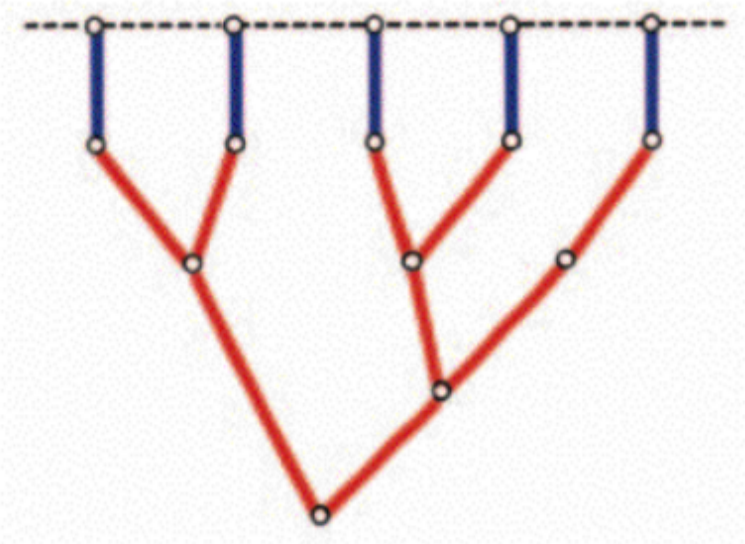
3 Hackenbush

This section assumes a basic knowledge of the game Hackenbush. The problem we examine is the computational complexity of the Red-Blue Hackenbush.

Theorem 3. *Hackenbush is an NP-hard problem*

Proof. We first examine a specific problem in Red-Blue Hackenbush known as the Redwood Furniture Problem.

Figure 27. A Redwood Tree.



Lemma 3. *The Redwood Furniture Problem is NP-hard.*

Proof. We start with the fact that a piece of redwood furniture has a value of $\frac{1}{2^n}$. We also utilize the Don't-Break-It-Up theorem, which says that if there is a move for Right which leaves a piece of redwood furniture connected, then

one of those moves which does so is worthwhile for Right.

For a redwood furniture with value $\frac{1}{2^n}$ this means that Right has a move to $\{0|\frac{1}{2^{n-1}}\}$. Now consider a redwood bed, which is a redwood furniture such that all the mattress edges each have one end at the top of a leg.

If you make a succession of worthwhile moves, you can prove that this redwood bed, that is a tree has value $\frac{1}{2}$

If you make these worthwhile moves for as long as you can without disconnecting it, we have that a Bed $= \frac{1}{2^m} \cdot \frac{1}{2} = \frac{1}{2^{m+1}}$. In other words, to figure out the size of a redwood bed, you have to know the smallest redwood tree in the bed which contains all its legs.

Thus, it follows that this problem is NP-Hard as it has been reduced.

References

[1] Elwyn Berlekamp (1982), *Winning Ways for Your Mathematical Plays*

[2] Vasiliki - Despoina Mitsou (2014), *THE COMPUTATIONAL COMPLEXITY OF SOME GAMES AND PUZZLES WITH THEORETICAL APPLICATIONS*

[3] David Lichtenstein, Michael Sipser (1978), *GO is pspace hard*

[4] T.J. Schaefer (1976), *Complexity of Decision Problems Based on Finite Two-Person Perfect-Information Games*