Some Mathematics of Go

Jeff Kaufman

April 29, 2006

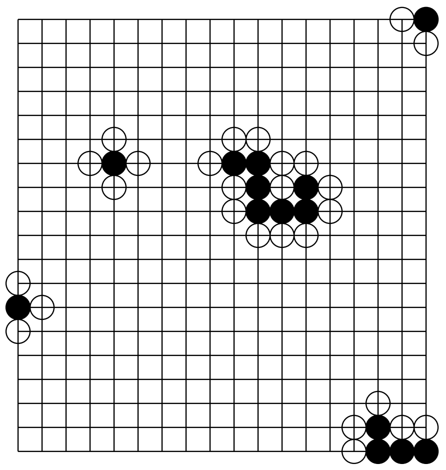
1 Introduction

The game of Go is very old and very simple. It has been played for at least 2500 years, and the complexity level, for humans, is similar to that of chess. For computers chess is substantially easier; while programs have beat the best chess players, no computer program is on the level of even the weakest professional Go player.

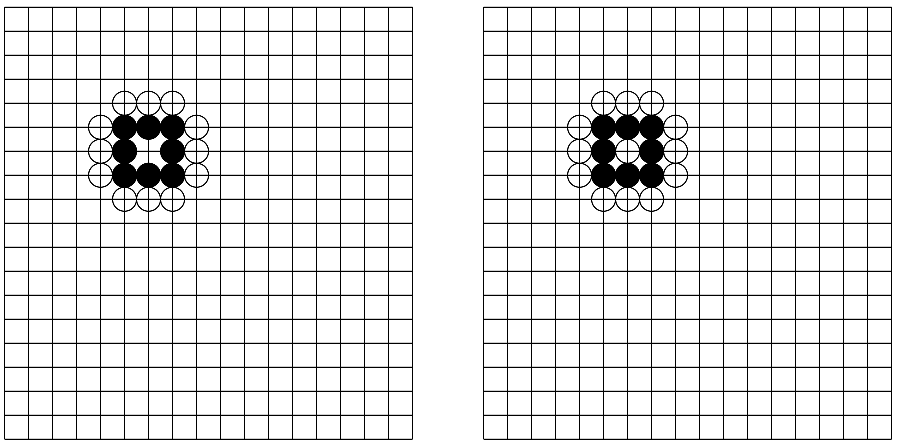
In this paper I show how we can apply simple graph theory techniques to Go, proving some simple results. While these results are original to this paper, none of them depart very far from approaches we've used in class.

2 Non-Mathematical Explanation of Go

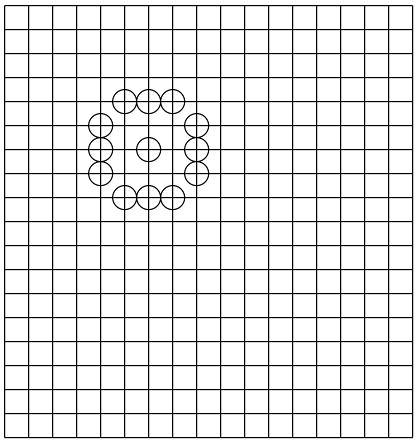
Go is a game played on a 19 by 19 grid. Two players alternate placing black and white pieces on the empty intersections. Pieces do not move once placed, except to be removed from the board in a process called *lifting*. A group of pieces is lifted if, considered as a group, it borders no empty spaces. These pieces are then removed from the board as *prisoners* to count against their original owner in the final scoring. In the example board below, all the black pieces should be lifted.



In an actual game you would never encounter a configuration like the above, with multiple groups of pieces that should be lifted, because after every piece played lifts are determined. Consider the two boards below, then. In the first, white is about to make a move, in the second white has made the move.



White played where it did in order to lift black's circle of eight. By our current definition of lifting, all nine middle pieces, black and white should be lifted. Instead there is a different rule, which says that when player A places you first consider if any of player B's pieces are lifted, lift them if need be, and then check for player A's. So in this case only the black eight are lifted, giving a board that looks like:



To ensure that the game has an end, there is an additional rule that prohibits making a move that would result, after lifts, in a board configuration that is identical to any previous configuration. This is usually referred to as the *coe rule*, after the *coe* shape.[1]

The game ends when both players agree it is over, at which point they decide who surrounded what territory and agree on scores.

3 A Graph Theory Explanation

Go is traditionally played on a 19 by 19 grid, but we can make a more general definition and consider the 19 by 19 version as a special case. Specifically, we define a *Go-board* to be a graph G whose verticies can take on three values; empty, black, and white. A *Go-game* then can be an ordered quadruple (G_{cur}, S, P_b, P_w),

[1]A coe is a shape where black and white could, if not prohibited from doing so, alternate back and forth lifting each other's pieces. The standard one and it's alternate are:

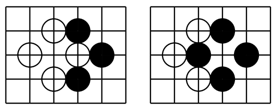


Figure 1: Algorithm Lift

```
Algorithm Lift
Input $(G_{cur}, S, P_b, P_w)$, $v$                    [$v$ is the vertex at which we just placed.]
Output $(G_{cur}, S, P_b, P_w)$
    function isSafe($L, p$)              [$L$ is the set of verticies we've looked at so far.]
                                                            [$p$ is the next one to look at.]
        isSafe $\leftarrow$ $false$
        for $a$ in adjacent($p$)
            if empty($a$) or ($a \notin L$ and sameColor($p$, $a$) and isSafe($L \cup \{p\}, a$))
                isSafe $\leftarrow$ $true$
            endif
        endfor
    endfunc

    procedure removeBlock(in $L, p$, inout $P_c$)
        for $a$ in adjacent($p$)
            if $a \notin L$ and sameColor($p$, $a$)
                removeBlock($L \cup \{p\}, a$)
            endif
        endfor
        setEmpty($p$)
        $P_c \leftarrow P_c + 1$
    endproc

    for $a$ in adjacent($v$)                                          [Main Program]
        if colorOf($a$) $\neq$ colorOf($v$) and !isSafe($\varnothing$, $a$)
            removeBlock($\varnothing$, $a$, $P_{\text{colorOf}(v)}$)
        endif
    if !isSafe($\varnothing$, $v$)
        removeBlock($\varnothing$, $v$, $P_{\text{colorOf}(v)}$)
    endif
```

where G_{cur} is the current board, S is the set of past boards[2] $\{G_0, G_1, \ldots, G_{n-1}\}$, and P_b and P_w are the amounts of prisoners possessed by each of black and white.

Implementing the rules on this structure requires a formal definition of the lifting and coe rule checking processes. For the former we first note that we have a nice property of lifting: if any pieces are to be lifted at all they must be or border the piece just played, or else they would have been lifted on a previous turn. We then only have to branch our recursion out from the piece just played.

This algorithm is implemented in figure 1 as Lift. It uses two subroutines, the first of which, isSafe, determines recursively if a block of pieces of like color is safe from lifting. If any piece within the block borders an empty space then the whole block is declared safe. The second is removeBlock, a procedure that gets invoked once we decide that a block of pieces should be lifted. It does the actual removal from the board and updates the prisoner counts. It would be possible to combine these two routines into one procedure for an algorithm faster by a constant, but that would give messier code.

The main routine uses isSafe to check each opposite color piece adjacent to the one that was just played to see if it forms a block that should be lifted, using removeBlock on any that should be. Then it looks at

[2]Without these we could not tell if we were breaking the coe rule.

the piece just played to see if it is in a block of same color pieces that should be lifted. If so, it goes ahead and lifts them. This leaves us with updated board and prisoner counts.

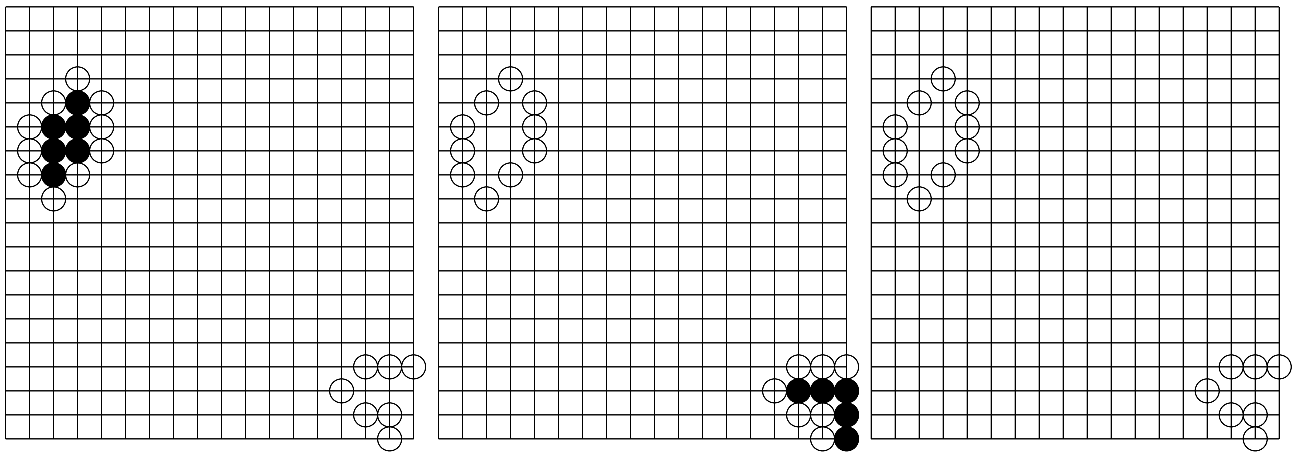
4 Upper Bounds

It would be nice to have some idea how long a game could be. We ought to be able to set some bounds on this.

Initial Upper Bound. *No game of Go (G_{cur}, S, P_b, P_w) can take more than $3^{|V|}$ moves, where V is the set of verticies of G_{cur}.*

Proof. There are $|V|$ places on the board one could play, and each at all times contains a black piece, a white piece, or no piece. There are then $3^{|V|}$ distinct configurations for a Go board. Because of the coe rule, where the board can't ever look as it looked in the past, each of these $3^{|V|}$ configurations can appear no more than once. This limits us to $3^{|V|}$ moves. □

Unfortunately, this is not a very close upper bound. The coe rule applies after LIFT has run, which means that while we would have counted the first two board positions below separately, they both reduce to the third one:



Any board in which some pieces should be lifted, then, is one that we don't want to be counting towards our number of games but are, and so we are over counting the maximum number of moves by a decent margin.

Unfortunately, without running an algorithm like our ISSAFE function, we don't have a way to tell if a board is going to reduce in the lifting process or not. This makes it hard to have a general idea of how much of an effect these reductions will have on the total number of distinct configurations. It is unlikely[3] that we would be reducing by enough to take us out of the exponential range, so our theoretical upper bound on game length is order $3^{|V|}$ and hence exponential.

This upper bound, though, is fantastically large. On a 19 by 19 board we would have the bound on the order of $3^{19\times 19}$, or 1.74×10^{172}. Games almost never go anywhere near that long. Unless people are learning the game or trying to make it last a long time, they almost never run out of pieces. As there are $|V|$ pieces in a typical Go set, we now have an (empirical) linear bound.

5 That Go Has a Winner

Go is finite, so for any starting board B_s we can build a tree of all possible configurations. This tree would have the empty board as root, all intermediate boards as internal nodes, and finished games as leaves. A

[3]NB: this is speculation.

board is put in as a child of another board if you can get from one to the other in one move. Note that a board configuration can show up multiple times in the tree, but never twice in any downward path from the root node to a leaf.[4] Call this the *move tree* for B_s.

We can then say that player x can force a win on board B if no matter what the other player, y, does, x will always have choices that will result in winning the game.

Go Has a Winner. *One of white or black can force a win on every board.*

Proof. Given an initial board B_s, construct the move tree. We can declare the *forcible winner* of a node where it is person x's turn to be y if the *forcible winner* of at least one child node is x and y otherwise. Nodes that have no children are leaves and represent finished games, so there we simply have the *forced winner* of that node be the winner of the finished game represented by that node.

This recursively defined forced winnerness will propagate up the tree until we know who wins every node, including the root node. The winner of the root node, player z, can then force a win by always choosing the moves represented by transitions to child nodes that are marked with z as their winner, and due to our definition of the forced winner of a node, there will always be at least one such transition and accompanying move. ☐

For a simple board we might actually be able to calculate the forced winner this way, by building the tree and recursing over it. This tree has a branching rate of around $\frac{1}{2}|V|$, however, as you can play in almost any open space and there are lots of open spaces. Additionally, we have a depth on the order of $3^{|V|}$. This gives a tree with around $(\frac{1}{2}|V|)^{3^{|V|}}$ nodes. In the standard board $|V|$ is 19^2, so we're talking about an unmanageably large tree, of size $(\frac{1}{2} \times 19 \times 19)^{3^{19 \times 19}}$. I tried to find a scientific notation representation for this, to get an idea of its size, but this number is so tremendously huge that I was not able to calculate it on a computer.

[4]That sort of double occurrence would violate the coe rule.