solving go on small boards

intro

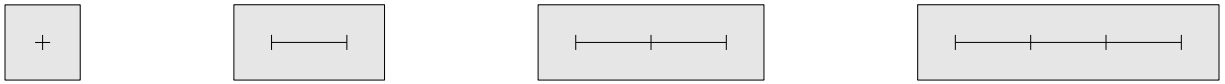
How to solve Go puzzles: given a position (and the history leading to that position) and the player to move, what is the minimax score?

Before discussing today's solvers, which can solve full-board puzzles only on small boards: which rule set should we use? Weiqi/Baduk/Go is popular in China/Korea/Japan, each country has its own particular rule set.

A *situation* is a position together with player to move. Each of C/K/J Go association uses their own form of situational superko. The American Go Association rules as written use situational superko (any move that yields a previous situation is forbidden), but the rule-writers apparently intended to use *natural situational superko*. The *creator* of a position is whoever placed the stone most recently placed in the position. The creator of the initial position is White, the 2nd player.) In natural situational superko, a player is not allowed to move by placing a stone that yields a position they created; they are allowed to move by passing and leave a position that they created.

Unless otherwise stated, when solving Go we will assume Tromp-Taylor rules with no suicide with positional superko. These two features make the search tree a bit smaller. Do you see why?

1xn go, also called linear go



The class github repo has an interactive linear go environment. What is Black's minimax score on these boards? $1 \times n$ scores for n up to 12 are below. See Exploring Positional Linear Go by Weninger and Hayward for details:
`http://webdocs.cs.ualberta.ca/~hayward/papers/lineargo.pdf`

Some answers.

1x1 Go: minimax score 0 (neither player can place a stone).

1x2 Go: if P1 places a stone, P2 can capture, P1 must pass (ko), P2 can pass and win by 2. Minimax score 0 (neither player places a stone).

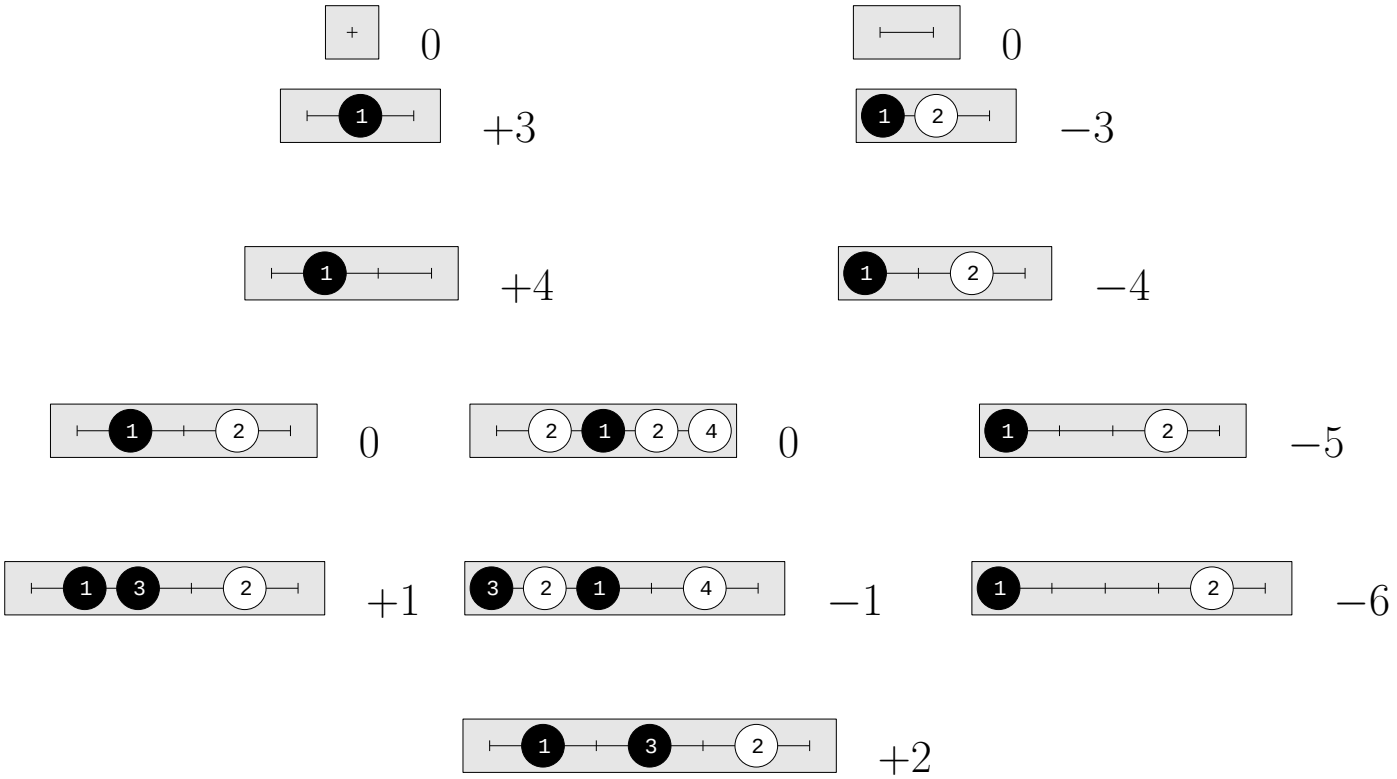
Can you work out the minimax values for n up to 5? Hints on the next page.

n	1	2	3	4	5	6	7	8	9	10	11	12
1×n Go minimax score	0	0	3	4	0	1	2	3	0	1	2	2

linear go **Tromp rules, no suicide, positional superko**

board size	1	2	3	4	5	6	7	8	9	10	11	12	13
1st player minimax value	0	0	3	4	0	1	2	3	0	1	2	2	?

principal variations (initial moves)



2x2 go search space

How hard is it to solve 2x2 Go?

How big is the search space? A go state is defined by its position and move history. (We don't need the complete move history: all we need is the set of board positions so far and whether the last move played was *pass* or a stone move.) An upper bound on the number of states is just the number of legal games (since a game can end at any state if both players then pass).

The number of legal 2x2 Go games is just the number of paths in the go state transition graph, which has as its nodes all legal positions, and has as its arcs all legal inter-node transitions. For 2x2, this has been computed exactly by John Tromp:

`https://tromp.github.io/go/gostate.pdf`

and is discussed here

`https://senseis.xmp.net/?PositionsAndGamesOn2x2:v26`

and here

`https://en.wikipedia.org/wiki/Go_and_mathematics`

The number of legal 2x2 games is 386 356 909 593.

The number of legal 3x3 games is about 10^{1100}.

solving 2x2 go

John Tromp's C implementation
`http://tromp.github.io/java/go/twoxtwo.html`

What is the difference between positional superko and situational superko? Which would take longer to solve? Why?

Tromp implemented three versions of his program: minimax, alpha-beta, and alpha-beta with good move ordering (pass is the first move option). The respective number of tree nodes is

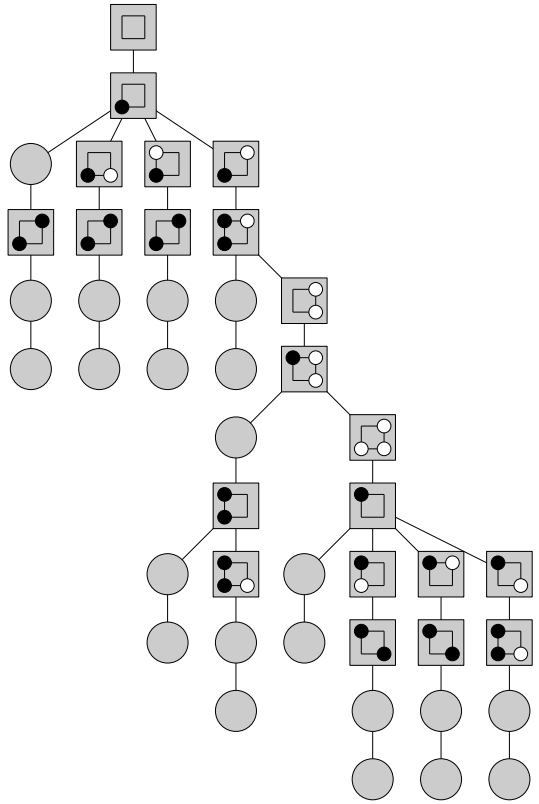
more than 1×10^{12}

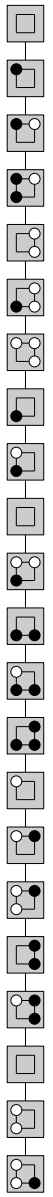
19397529 (max depth 58)

1446 (max depth 22, e.g. as shown at right on the next page).

The minimax value is Black wins by 1.

This strategy tree shows that Black wins by at least 1. (why?) (Each circle is a pass move.) A similar tree shows that Black wins by at most 1. Exercise: create this other tree. These two trees, each with about 50 nodes, prove that the minimax value is Black wins by 1.





Before running Tromp's code, make sure that `CUT` is 1: if `CUT` is 0 you will run the minimax version, which takes over a week to finish. Tromp's code checks the pass move before trying other moves. What happens if you check the pass move after all other moves? To find out, make these changes: in `xab( )` and `oab( )` move the two lines below to be after the `for` loop. What is the difference in the runtime? In the size of the search tree?

```
s = passed ? score(black, white) ...
if (s  ...
```

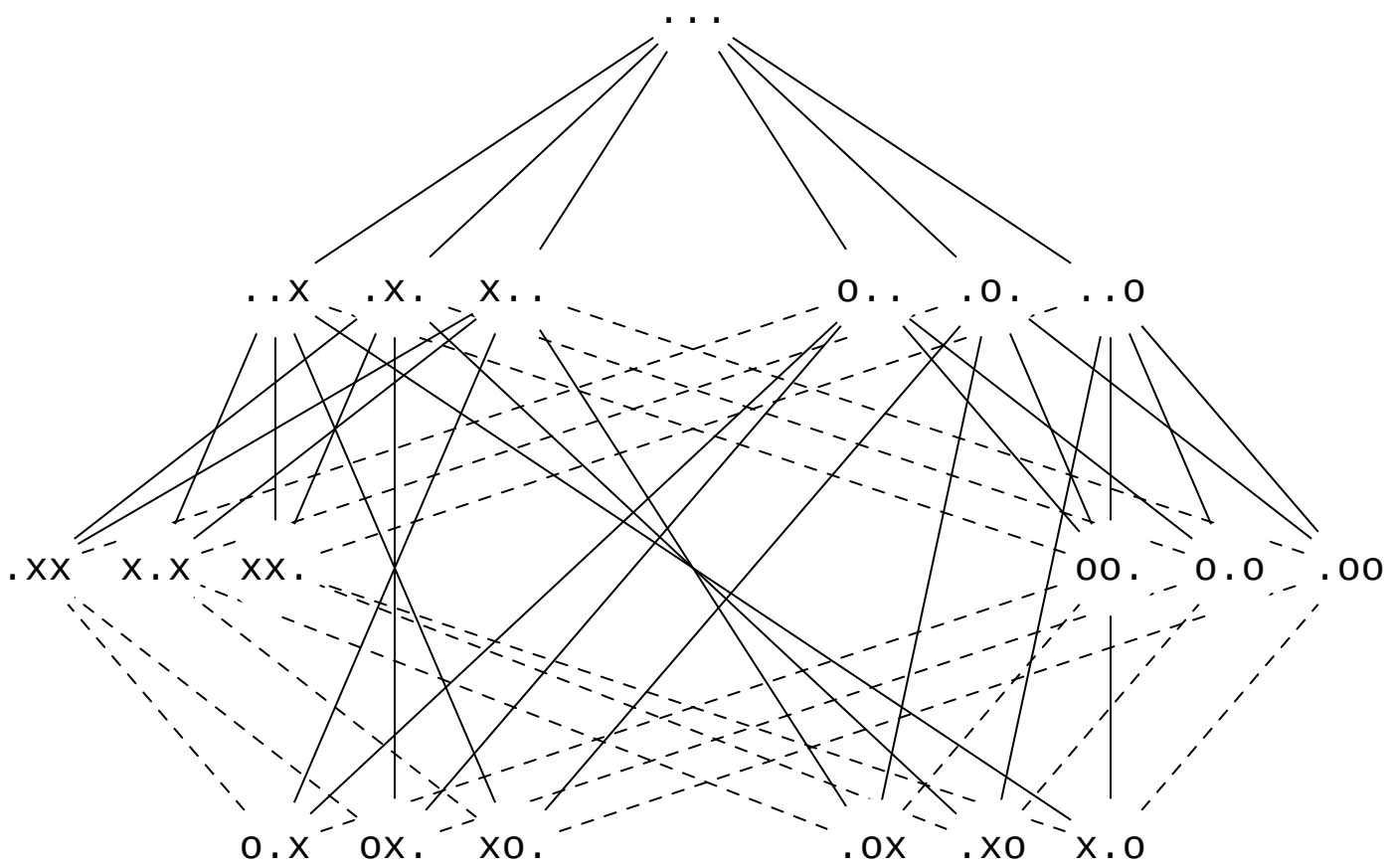
trigo go on a triangle

trigo is the game of go played on a 3-cell triangle. We can write a solver following the observations in Tromp's 2×2 solver

```
...

x..   .x.   ..x   o..   .o.   ..o

.xx   x.x   xx.                     oo.   o.o   .oo

.ox   o.x   .xo   ox.   xo.   x.o
```

trigo position transition digraph

solid arcs: down dashed arcs: up



3439 paths from root

6878 nodes in game tree (final pass node for each path)

3x3 go

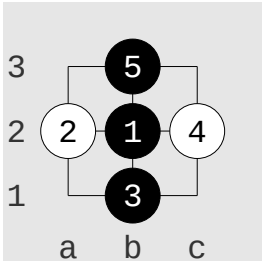
There are about 10^{100} different 3x3 Go games, so there would be this many leaf nodes in the empty-board minimax tree. So, to have any hope of solving 3x3 Go states, we need to use cutoffs and a transposition table and other ideas. One ideas is to recognize regions that can be kept *safe* from the opponent.

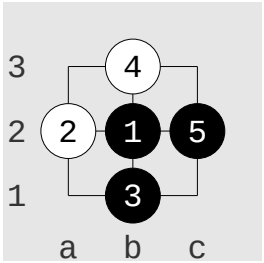
A *group* is a connected set of stones of one color. An *eye* is a connected set of empty cells. We have seen that, on any size board, a group is safe if it has at least two eyes. On the 3x3 board there are two shapes that, together with the edge of the board, create a safe region.

For example, see below left. After move 5 in the board below left, Black can capture both White stones and then the Black group with {1,3,5} splits the board into two regions, White can live in neither, so Black can capture all territory.

Similarly, see below right. The Black group with {1,3,5} creates an empty cell where White cannot play without first occupying the outer five cells. But Black can prevent this from happening, so again can capture all territory.

So, on the 3x3 board, if a player forms either the *middle-3* or *bent-3* shape, they can win 9-0. When solving 3x3 Go, we save time by checking — say before each move — whether the above condition can be reached.





To finish this section, let's solve all 3x3 Go opening-move states: what are the minimax values? Of the 10 possible first moves, four are non-isomorphic.

center

Black can form either a middle-3 or a bent-3 (prove this), so the final minimax score will be Black wins by 9.

pass

White follows any Black-optimal strategy, White wins by 9.

corner

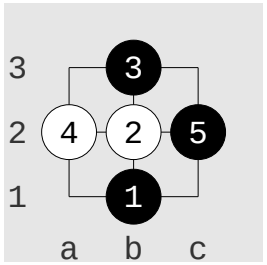
White wins by 9. Proof: exercise. This is not difficult. Hint: first read the rest of this section.

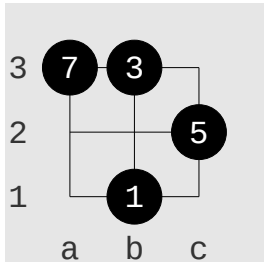
side

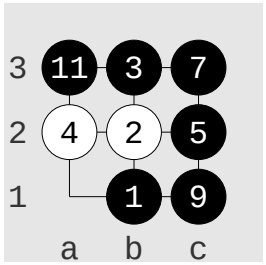
After 1.B[b1] the minimax score is Black wins by 3. Here we outline a proof.

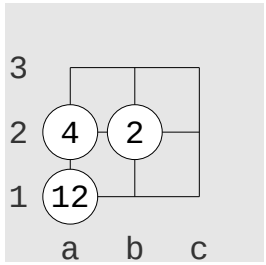
We can assume 2.W[b2] (otherwise 3.B[b2] and on move 5 Black can form a middle-3 or bent-3 and win 9-0). Then Black plays 3.B[b3]. By symmetry, we can assume White's next move is one of pass, a1, a3, a2. In the first three cases 5.B[a2] and Black has a bent-3 and wins 9-0. In the remaining case, 5.B[c2] and the board is as shown below left. If White passes, Black can pass and win by 3.

(White does worse by making a non-pass move. By symmetry, assume 6.W[a1]. Then 7.B[a3] and the board is shown below next. Now if 8.W[a2] then 9.B[b2]; if White makes any other move 8 then 9.B[a2]; in all cases Black can win 9-0.)









So far we have shown that Black can win by at least 3. Can Black do better? No. To prove this, we give a White strategy where White loses by at most 3.

After 2.W[b2], Black has five non-isomorphic moves. If 3.B[pass], 3.B[c3] or 3.B[c1] then 4.W[b3] and White can get a bent-3 and win 9-0. If 3.B[b3] then 4.W[a2] as before; now if Black plays anything except 5.B[c2] then White can win 9-0 (exercise), so again we have the position above left. If 3.B[c2] then White

might be tempted to play 4.W[a3], but 4.W[a2] is good enough. Now Black has to reply 5.B[b3], for otherwise White can get a bent-3, and we have a transposition to the position above left.

From the position above left, if Black passes then White passes and the game ends and Black wins by 3 and we are done. But what if Black tries to do more by attacking? Black must first fill its territory. (If Black attacks before filling then White can capture and will do better than lose by 3. For example, if 6.W[pass] 7.B[a3] then 8.W[c3] 9.B[?] 10.W[b3] and White has a bent-3 and wins 9-0. Or if 6.W[pass] 7.B[c3] 8.W[pass] 9.B[a3] then 10.W[c1] and again White will win 9-0.)

So assume 6.W[pass] 7.B[c3] 8.W[pass] 9.B[c1] 10.W[pass] 11.B[a3] as shown above 2nd-from-right. Now White captures, as shown above right, and from here White can win 9-0.

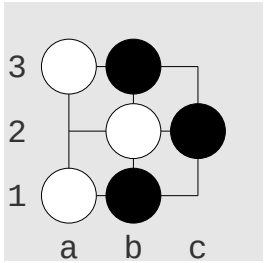
So, from the position above left, the best that Black can do is win by 3. So the minimax value after 1.B[b1] is Black wins by 3.

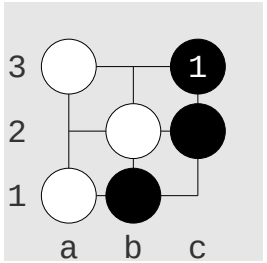
a fun 3x3 puzzle

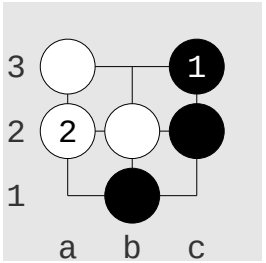
From `https://senseis.xmp.net/?PinwheelKo`. Assume Black plays 1.B[c3] below left, yielding below middle. Now White must play 2.W[a2] (otherwise Black can win 9-0), yielding right. Now Black must play 3.B[c1], and so on.

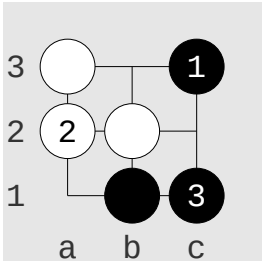
Assuming positional superko, who wins? Draw the principal variation move sequence. Assuming situational superko, who wins? Draw the principal variation

move sequence.









4x4 Go and 5x5 Go

Ted Grange and others solved 4x4 Go by hand around 1999:
http://www.mathpuzzle.com/go.html

In 2002 Erik van der Werf finished solving 5x5 Go with computer, using an enhanced alpha-beta search and a strong move ordering heuristic (good move ordering is necessary to get early cutoffs). Here is an animated webpage showing optimal play for the three strongest 5x5 opening moves, and the paper:
http://erikvanderwerf.tengen.nl/5x5/5x5solved.html
http://erikvanderwerf.tengen.nl/pubdown/solving_go_on_small_boards.pdf

He later solved other rectangular boards with up to 30 cells:
http://erikvanderwerf.tengen.nl/mxngo.html

6x6 Go is open

So far, no one has solved 6x6 Go. Van der Werf's alpha-beta approach will not work: there are too many nodes in the search tree. A directed-acyclic-graph approach with a transposition table should work. There is a technical problem: there will be states with different move histories that have the same position. A solution is to bucket states by position, so one bucket might contain different states. Figuring out what extra information to store for each state in a bucket, and how to proceed algorithmically, is called the graph history interaction problem. This problem was studied by Kishimoto and Mueller, the algorithmic solutions are known, they just need to be implemented.
`https://webdocs.cs.ualberta.ca/~mmueller/ghi.html` . Good move ordering will also be necessary; this could probably be learned by a neural net.