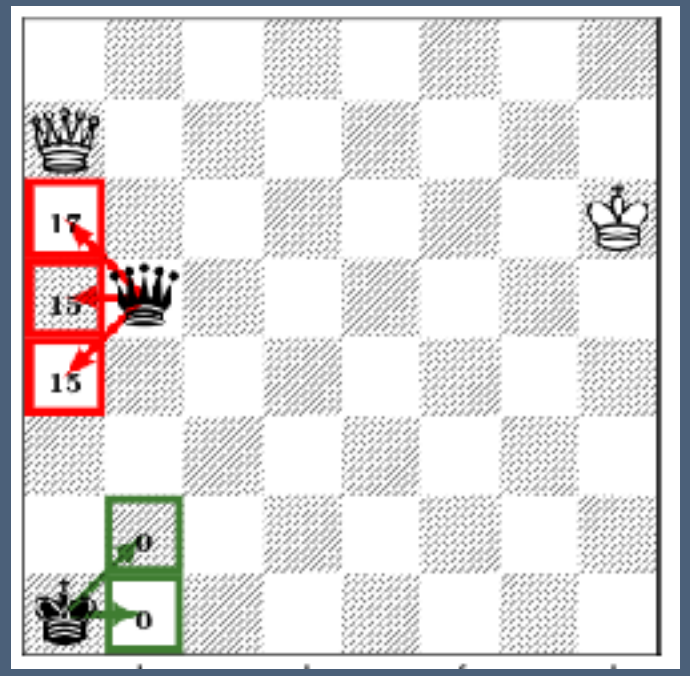
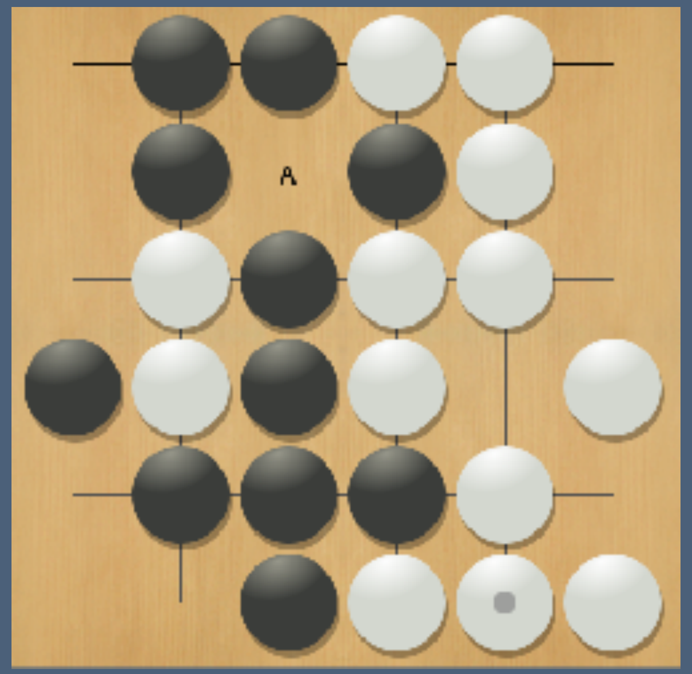
Evaluating Strong Engines Against Perfect Endgame Play

Case Studies in Chess and Go





Work by Rejwana Haque, Asmaul Husna, Ting-han Wei, Quazi Sadmane, Martin Müller

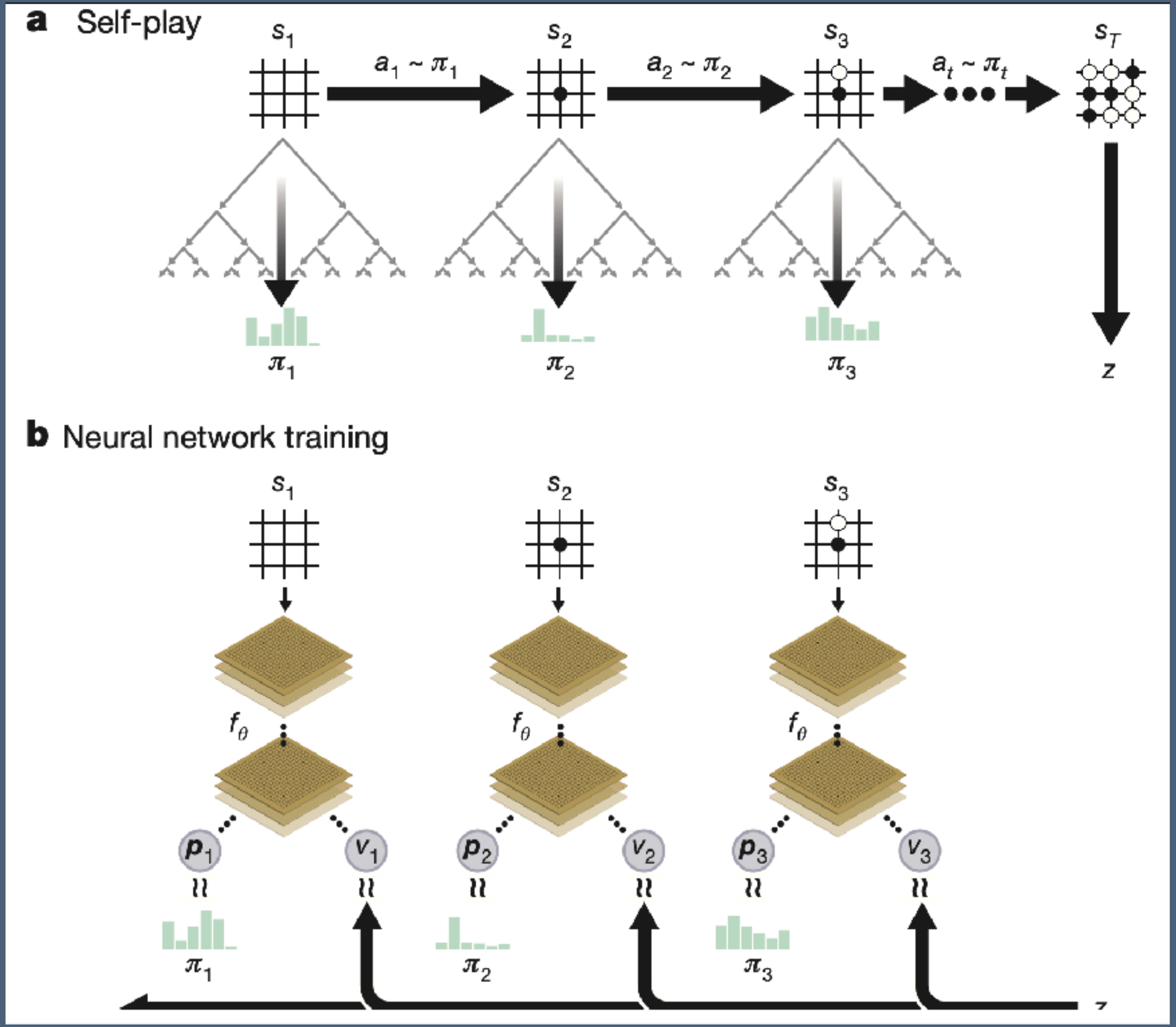
Talk presented by Martin Müller
Funded by NSERC, DeepMind Chair in AI, CIFAR

AlphaGo and Alpha Zero

- Famous series of DeepMind game-playing programs, 2014-18
  - AlphaGo, AlphaGo Zero, AlphaZero
  - Later MuZero, Stochastic MuZero
- Super-human play in Go, chess, shogi
- Inspired many other programs and generalisations
- Strong open source programs: Leela chess zero, KataGo

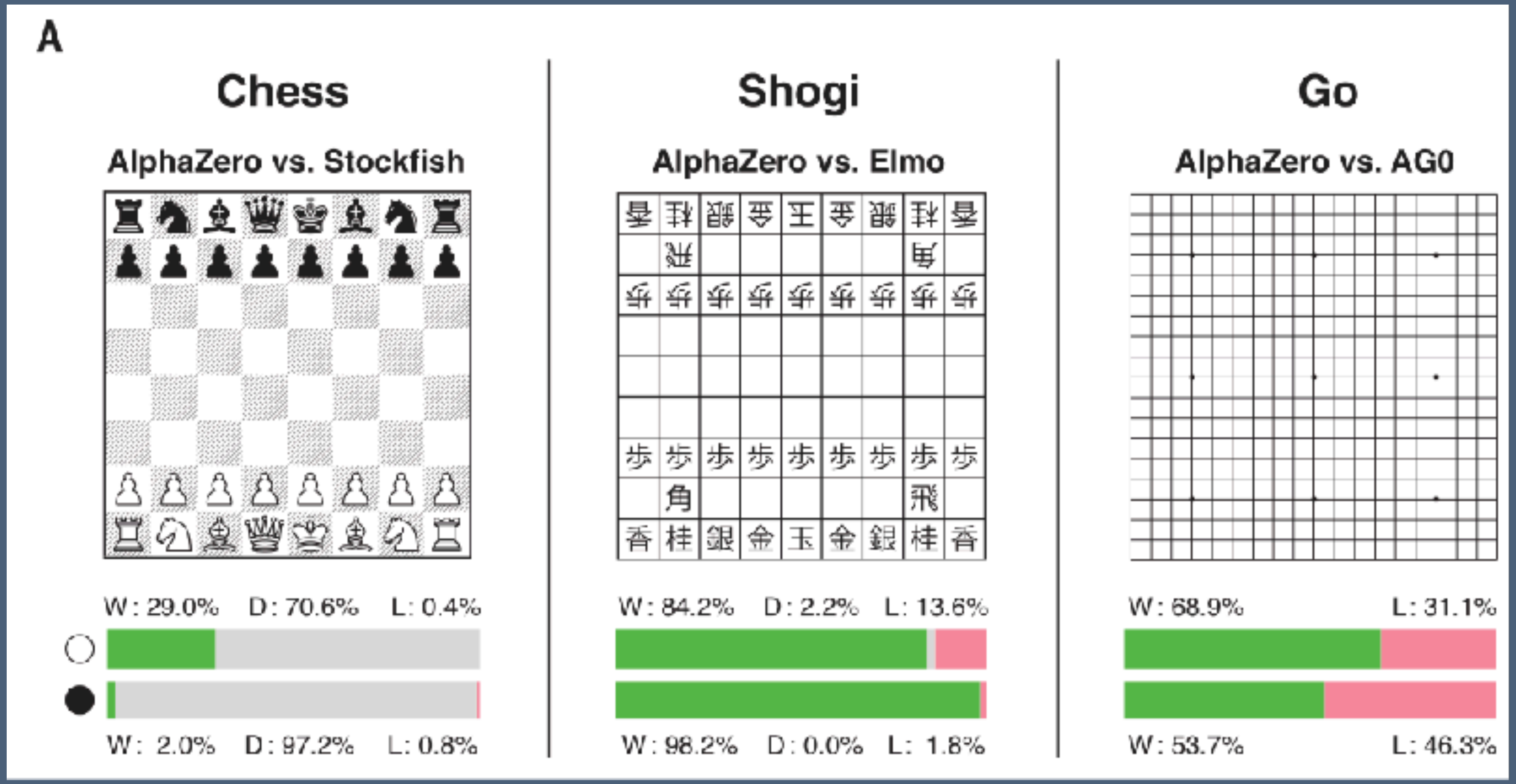
How Do these Programs Work?

- Train deep neural net
  - Policy = move generator
  - Value = state evaluation
- Deep selective search:
  Monte Carlo Tree Search (MCTS) with PUCT
  - Grow a deep search tree shaped by policy and value
- Training: self-play and reinforcement learning
  - Learns network from millions of games
    - from random to super-human



How Good are These Programs?

- Overwhelming success
  - against human players: Lee Sedol, Ke Jie, ...
  - against previous engines such as “plain” MCTS, alphabeta-based engines, previous AlphaGo
- In chess, on par with sophisticated alphabeta search using “simple+fast” NNUE networks
  - Open source Stockfish program with NNUE

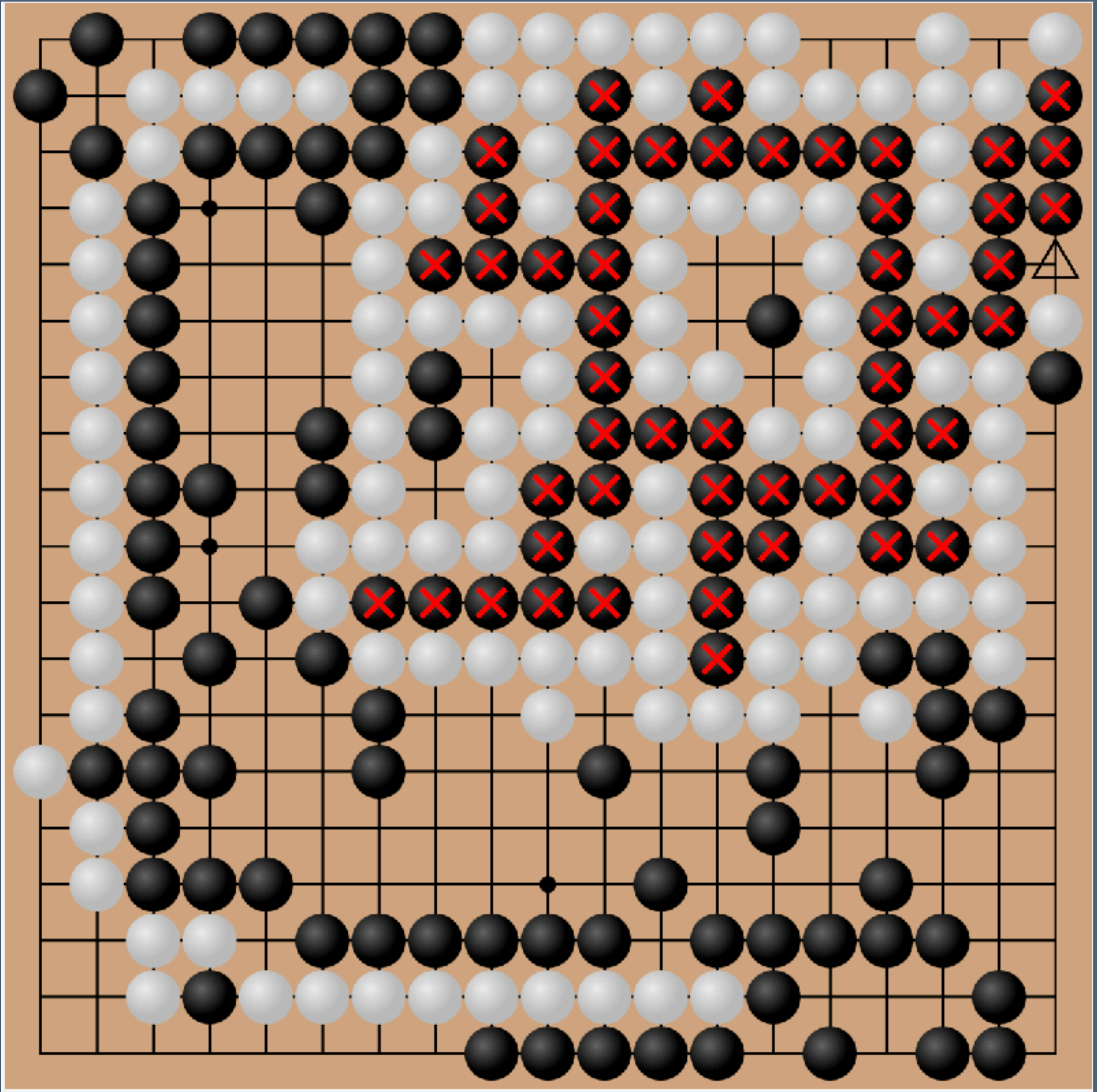


Are the Current Programs Unbeatable?

- Short answer: No
- First evidence: self-play results. Both Black and White player wins games
  - With optimal play, this cannot happen
    - Either one player wins all games, or
    - all games are draws
  - Second evidence: adversarial attacks (next slide)
  - Third evidence: our work on endgames presented here

Adversarial Attacks

- Can "trick" KataGo into losing a game
- First attack: exploit implementation bug
  - Bug: passes when far ahead, even though that loses immediately
    - Oversight of the programmer, easy fix
- Second attack: blindness against surrounding, "cyclic attack"



An example of the original cyclic attack in action.

Adversarial Attacks - Some References

- Li-Cheng Lan, Huan Zhang, Ti-Rong Wu, Meng-Yu Tsai, I-Chen Wu, Cho-Jui Hsieh.
  Are AlphaZero-like Agents Robust to Adversarial Perturbations?
  NeurIPS 2022

- Finbarr Timbers, Nolan Bard, Edward Lockhart, Marc Lanctot, Martin Schmid, Neil Burch, Julian Schrittwieser, Thomas Hubert, Michael Bowling
  Approximate Exploitability: Learning a Best Response
  IJCAI 2022

- Tony T. Wang, Adam Gleave, Tom Tseng, Kellin Pelrine, Nora Belrose, Joseph Miller, Michael D. Dennis, Yawen Duan, Viktor Pogrebniak, Sergey Levine, Stuart Russell
  Adversarial Policies Beat Superhuman Go AIs
  ICLR 2023

Motivation for Our Work

- Can we beat these programs “fairly”, without tricks?
- Can we estimate how close to perfect play they are?
- In general: **no.**
  - Chess, Go, shogi…openings and middle games are much too complicated
  - No human or computer knows what perfect play is
- In specific endgame situations: **yes!**
  - Chess: endgame databases with pre-computed perfect play
  - Go: endgame puzzles with mathematical “sum of games” structure

Related Publications From Our Group

- R. Haque, T.-h. Wei and M. Müller.
  On the Road to Perfection? Evaluating LeelaChess Zero Against Endgame Tablebases.
  Advances in Computer Games (ACG 2021).

- R. Haque.
  On the Road to Perfection? Evaluating LeelaChess Zero Against Endgame Tablebases.
  MSc thesis, University of Alberta, 2021.

- Q. A. Sadmine, A. Husna, and M. Müller.
  Stockfish or Leela Chess Zero? A Comparison Against Endgame Tablebases.
  Advances in Computer Games (ACG) 2023.

- A. Husna.
  Analyzing KataGo: A comparative evaluation against perfect play in the game of Go.
  MSc thesis, University of Alberta, 2024.

From: https://webdocs.cs.ualberta.ca/~mmueller/publications.html

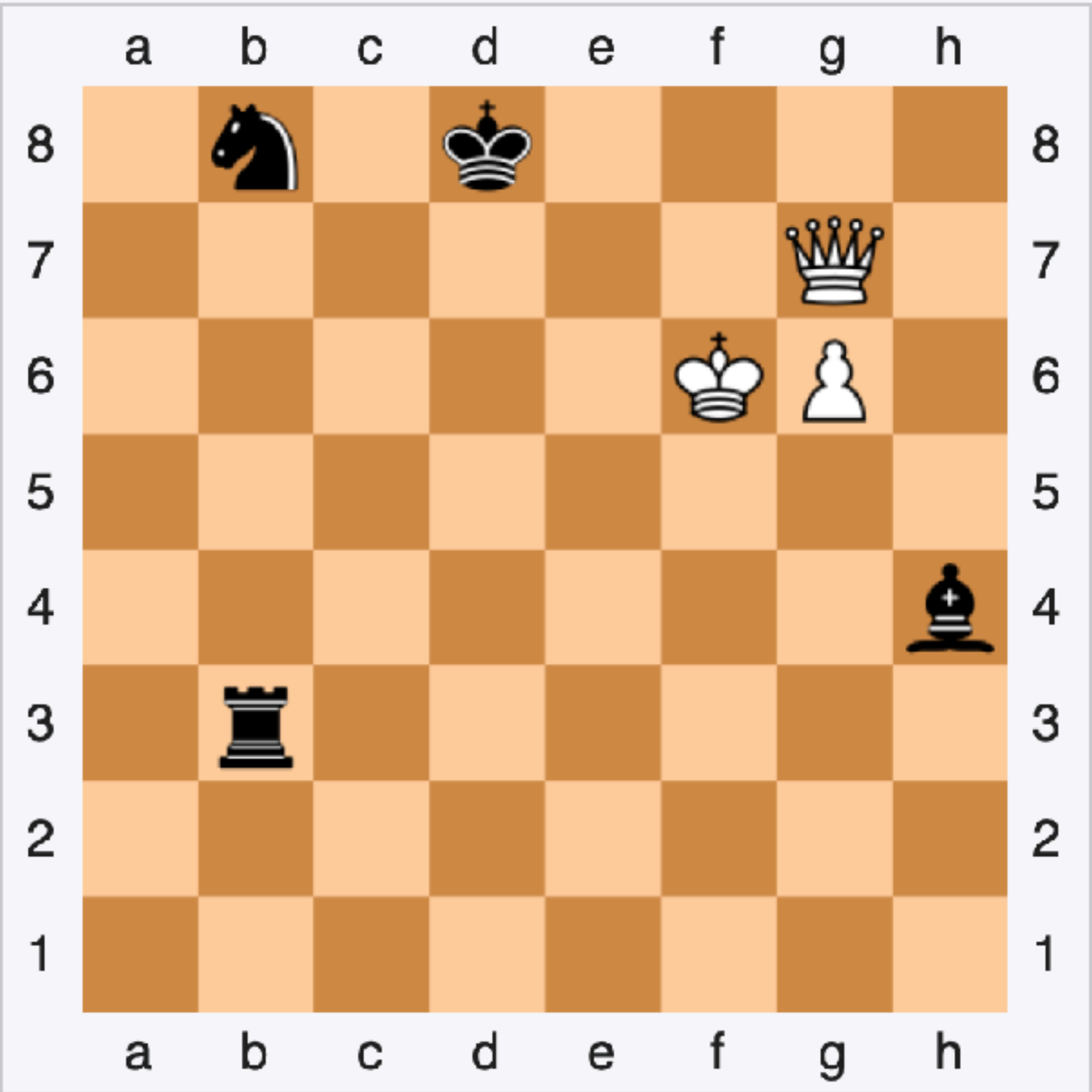
General Research Questions

- How close to perfection is AlphaZero?
- There is evidence that shows AlphaZero still makes mistakes
- Deeper analysis
- Goal: better understanding of AlphaZero limitations

Part 1 - Chess

Chess Endgame Tablebases

- Chess: pieces are captured during the game
- Endgame: only few pieces remain
- Can build complete databases “tablebases” with perfect play (minimax) result
- State of the art:
  - All positions with 7 or fewer pieces completed
  - 8 piece positions under construction (huge...)
  - Results and strategy far beyond human understanding



White to move and **mate in 549**. This is the longest mate with seven or fewer pieces on the board.

Chess Endgame Databases We Used

- Idea: start with simplest databases
- Check how program plays
- Tested all “non-trivial” 3 and 4 piece databases
  - Example of 3 piece: King + Rook vs King
  - Example of 4 piece: King + Queen vs King + Pawn
- One difficult 5 piece database: King, Queen, Rook vs King and Queen
  - Much larger database - used a random sample of 1% of all positions

The Program: Leela Chess Zero

- Leela Chess Zero (Lc0)
- Open source chess program
- Re-implementation of Alpha Zero ideas
  - Adds other improvements such as auxiliary outputs
- Trained by large group of volunteers, who donate computer resources
- One of the strongest open source programs
- We used version Lc0 0.27 on “modest” hardware (1 Nvidia Titan RTX)

Specific Research Questions

- How well does Lc0 play in these endgame positions?
- What is the influence of network training?
  - Strong network vs intermediate (less training)
- What is the influence of search?
  - Raw network vs Monte Carlo Tree Search (PUCT)
- Can we find specific types of mistakes? Can we explain them?

Weak vs Strong Network

- Strong network: best up to May 2021
  - Program strength (with search) 3062 Elo, superhuman
- Weak network: after 60 generations of training
  - Rating 1717 Elo

How do we Define a Mistake?

- Mistake: a bad move that changes the game-theoretic outcome
  - Best move leads to draw, but the program's move loses
  - Best move leads to a win, but program's move loses or draws
  - (If best move leads to loss: ignore the position)
- Other more detailed measures are possible
  - Win in the minimal number of moves
  - Not used here

Overall Results - 3 and 4 Piece Positions

- 3 Piece: easy
  - Weak network without search makes a few mistakes
- 4 Piece
  - Strong network: 20-80x fewer mistakes
  - Some mistakes remain
  - Search solves most, not all

Table 1. Total number of mistakes by the policy net and MCTS with 400 simulations, using strong and weak networks.

EGTB	Total Positions Tested	Weak Network		Strong Network	
		Policy	MCTS-400	Policy	MCTS-400
KPk	8596	390	13	5	**0**
KQk	20743	109	**0**	**0**	**0**
KRk	24692	69	**0**	**0**	**0**
KQkq	2055004	175623	12740	3075	36
KQkr	1579833	141104	3750	4011	46
KRkr	2429734	177263	6097	252	**0**
KPkp	4733080	474896	41763	20884	423
KPkq	4320585	449807	46981	6132	13
KPkr	5514997	643187	60605	13227	196

Overall Results - 5 Piece Positions

- KQRkq - King+Queen+Rook vs King+Queen
- Over 200 million positions
- Randomly sampled 1% for analysis, discard losses
- 683022 wins, 147694 draws
- Raw network, and small MCTS searches
  - Draws are much harder to play
  - More search again gives strong improvement

Search Budget	Winning Error	Drawing Error	Overall Error
MCTS-0	1.137	5.186	1.857
MCTS-400	0.040	0.297	0.086
MCTS-800	0.025	0.17	0.052
MCTS-1600	0.011	0.105	0.028

Table 5.1: Error rate in percent on five piece sample tablebase.

Comparing Search Errors - 3 vs 4 vs 5 Pieces

Search Budget	Three Piece	Four Piece	Five Piece
MCTS-0	0.0092	0.2306	1.8573
MCTS-400	0	0.0034	0.0857
MCTS-800	0	0.0012	0.0516
MCTS-1600	0	0.0004	0.0278

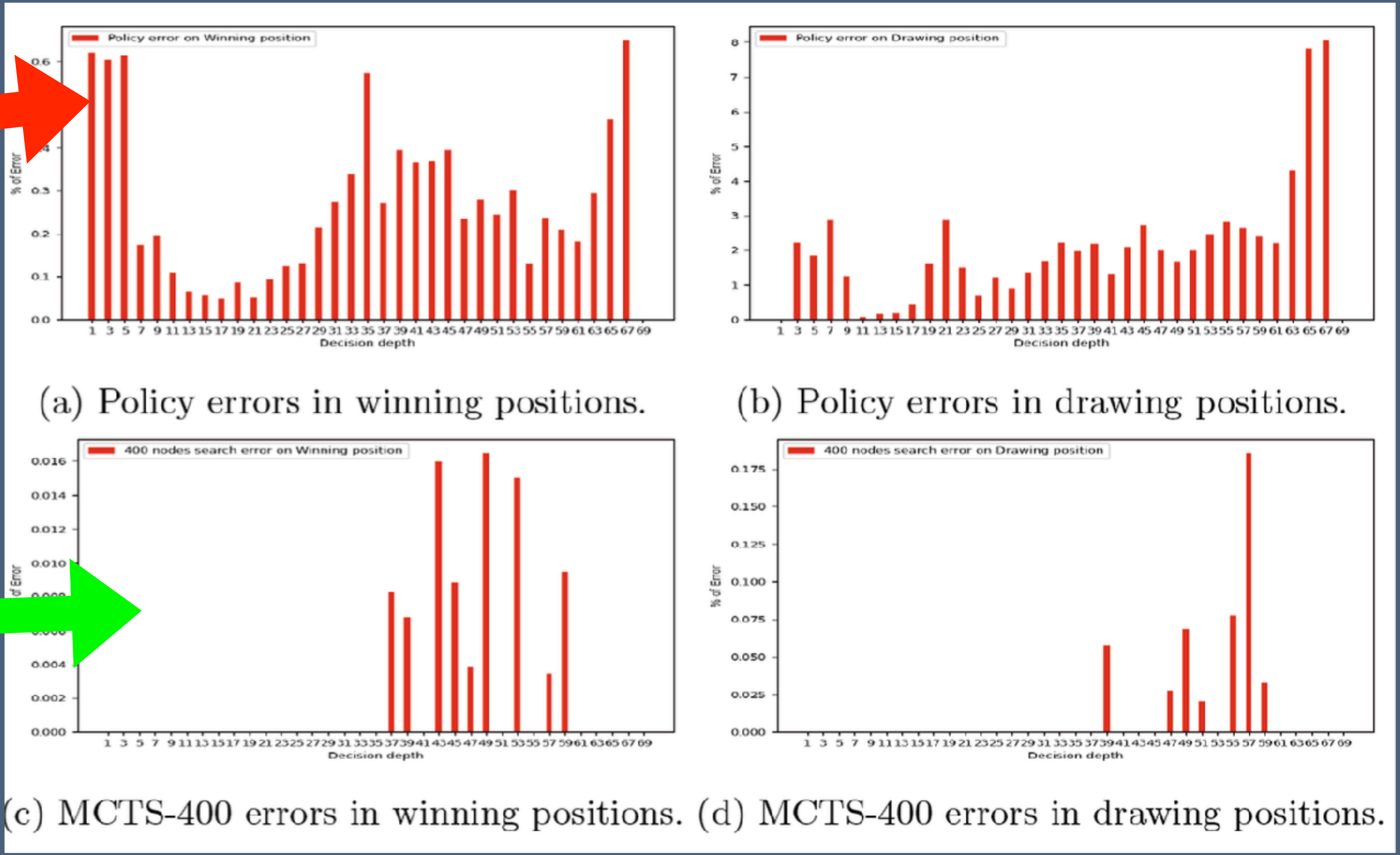
Table 5.2: Average error rate in percent on all tested three, four and five piece tablebases.

Decision Depth

- Decision depth: a rough measure of difficulty of a move decision
  - Winning move: Distance to mate (with best opponent play)
  - Drawing move: Longest distance to mate for other, losing moves

Errors vs Decision Depth

- Policy only (no search) makes some blunders at very low decision depth
- Even a small search is very powerful
- All errors at decision depths below 35 disappear

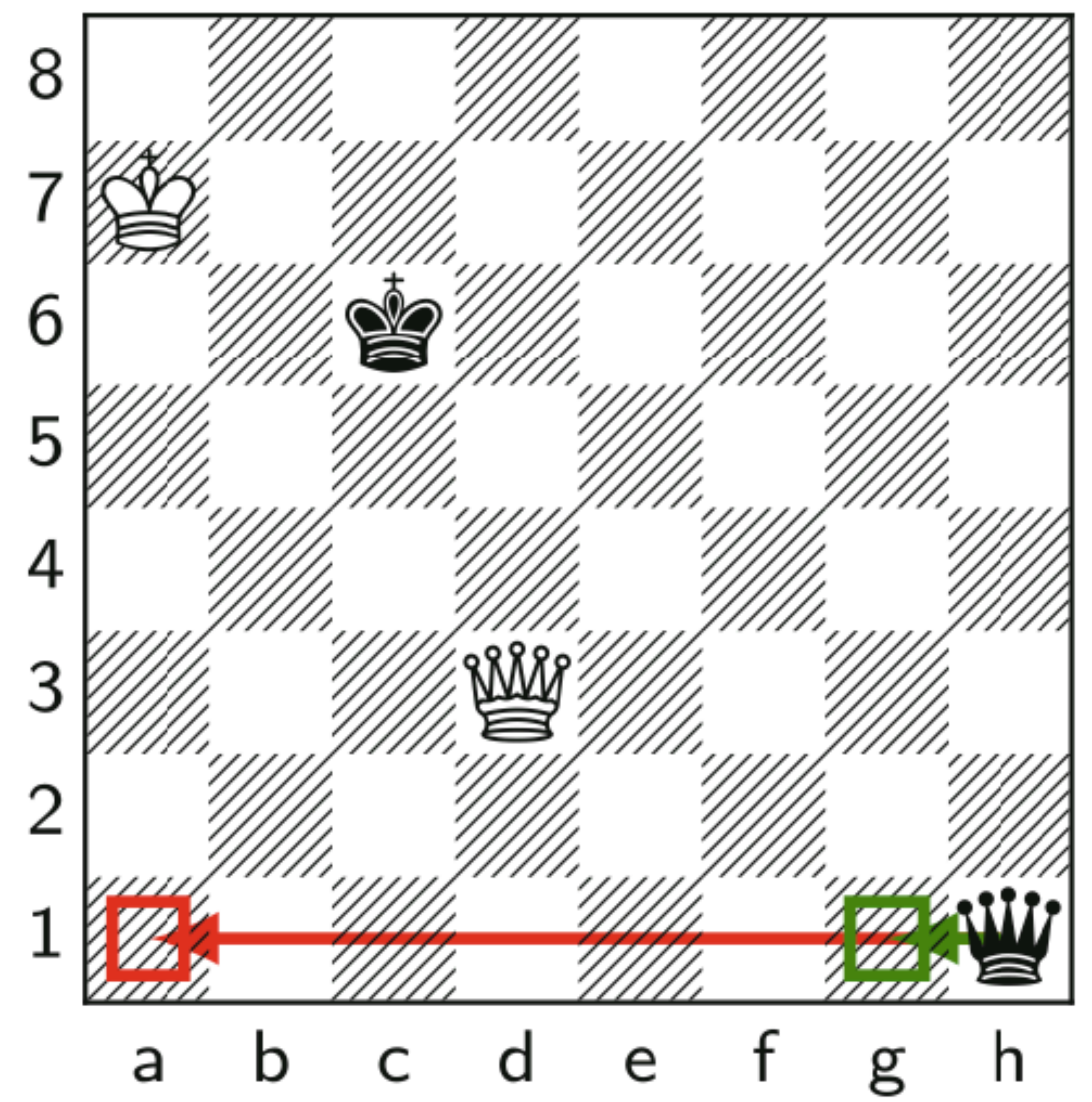


Some Interesting Mistakes

- Policy errors
- Search errors
- Search making things worse
- Why do these mistakes happen?

Example - Bad Policy, Easy Search

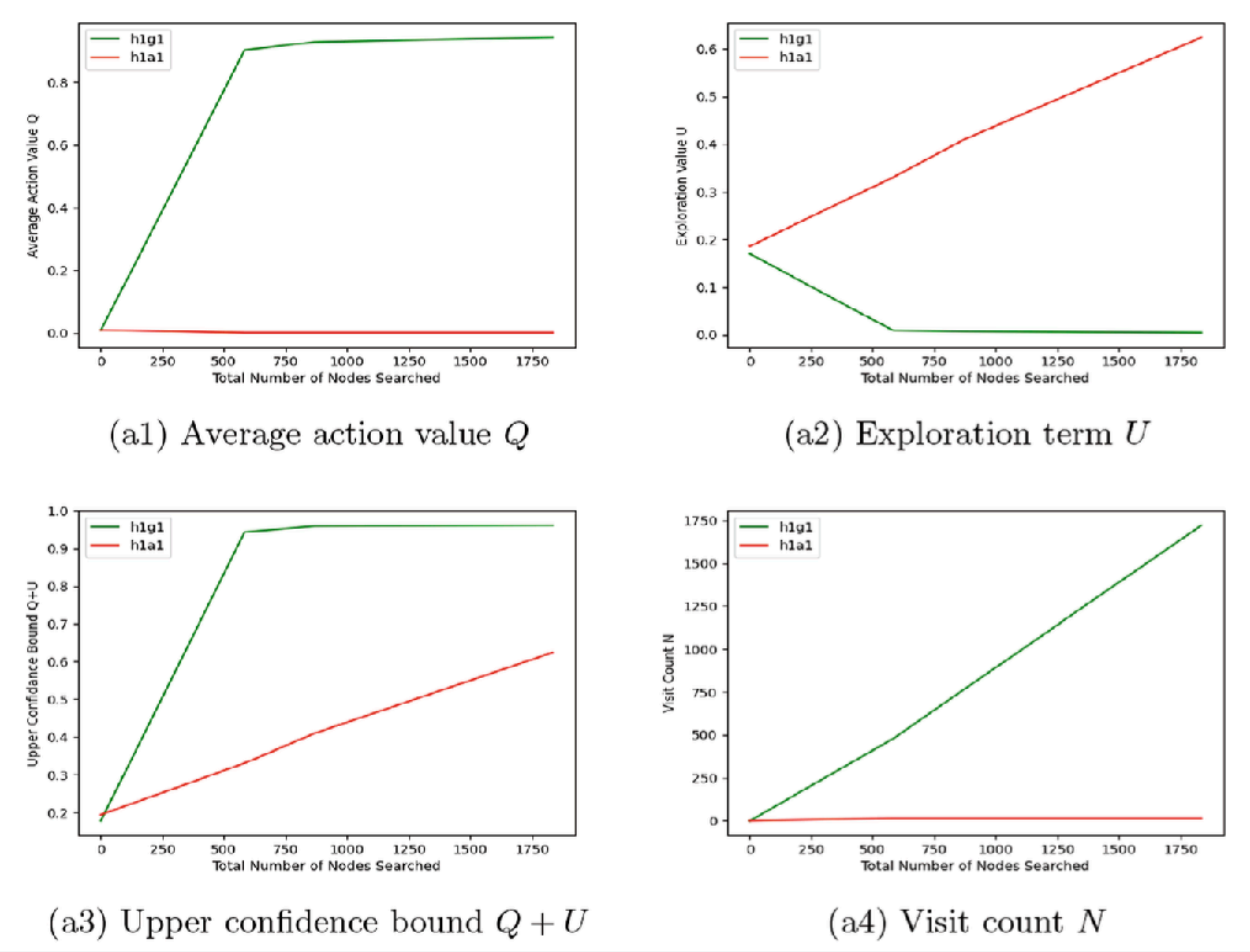
- Qg1 wins
- Qa1 only draws
- Policy: Qa1 has higher probability
- Search: very easy to see that Qg1 is correct
  - Value after 1 move is already much higher



(a) Policy wrong, search correct

Easy Search - Progress

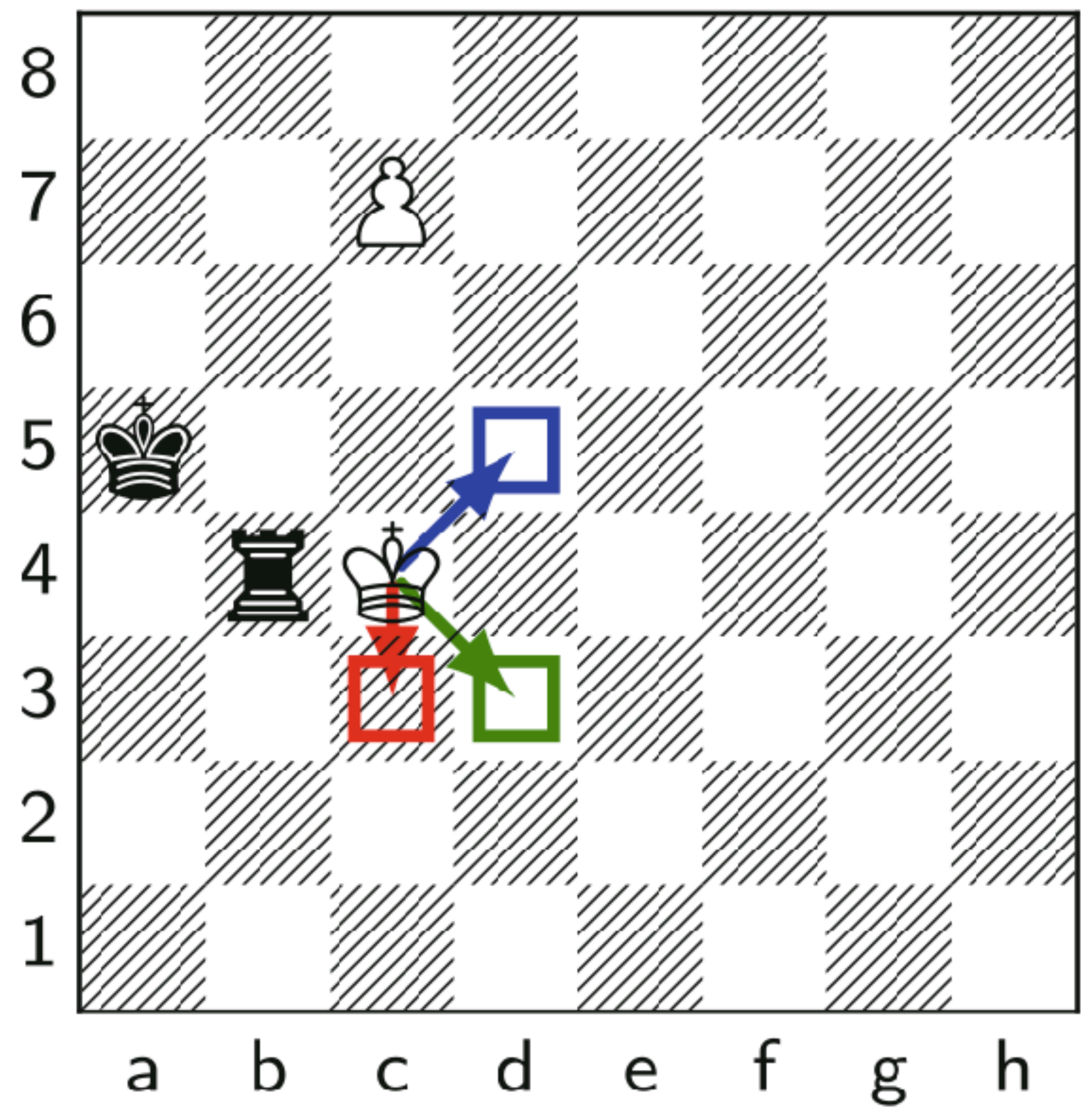
- Qg1 quickly becomes best move
- Dominates in both Q value and UCB value Q+U
- Almost all simulations explorer Qg1



(a1) Average action value Q
(a2) Exploration term U
(a3) Upper confidence bound $Q + U$
(a4) Visit count N

Bad Policy, Difficult for Search

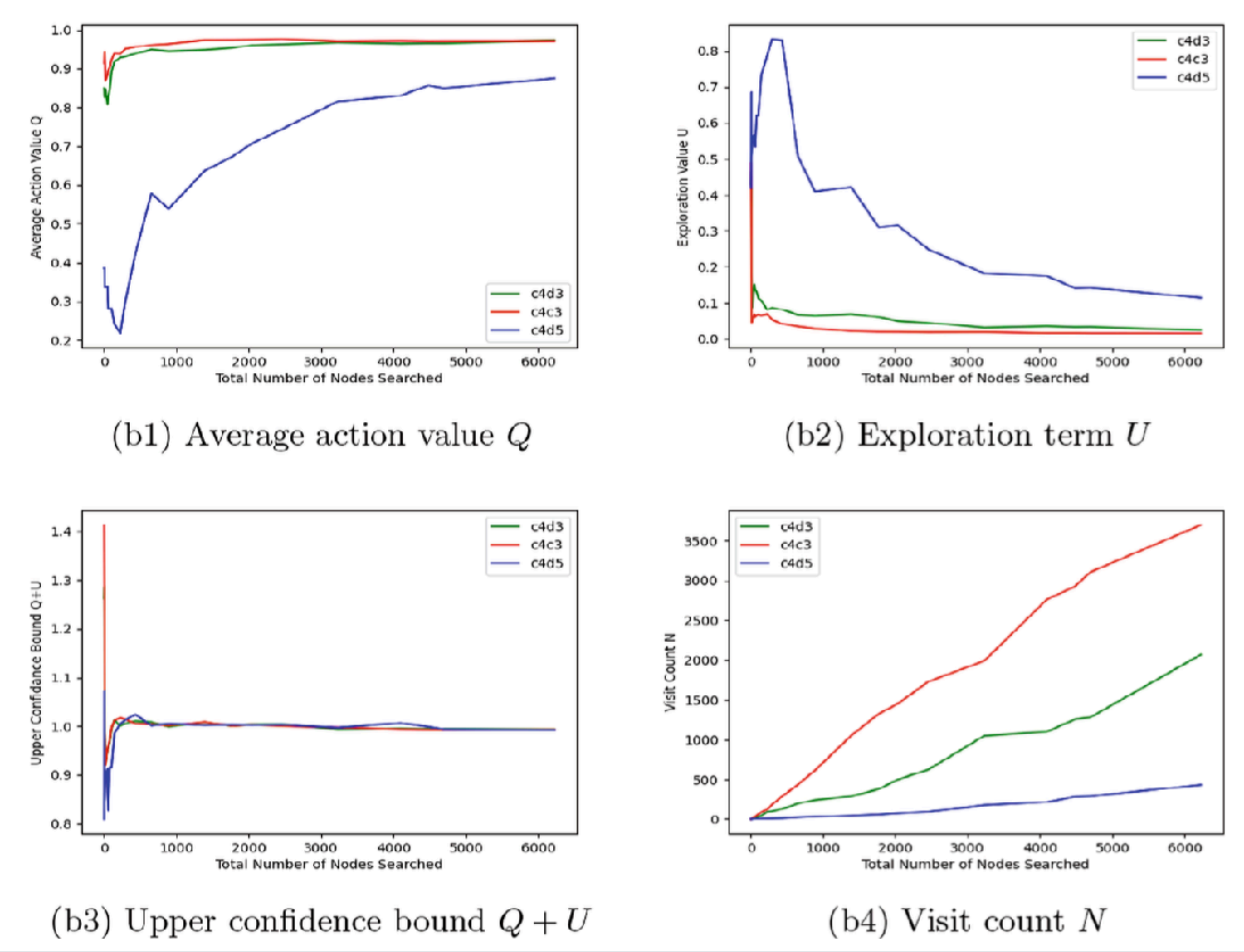
- Kd3 and Kd5 win
- Kc3 is only a draw
- Both policy and (small) search prefer Kc3
- Search needs 12000 simulations to switch to a correct move



(b) Policy wrong, search also wrong

Difficult for Search - Early Progress

- Red = bad move Kc3
- Blue and green = good moves
- Within 6000 simulations, search cannot see that red move is bad



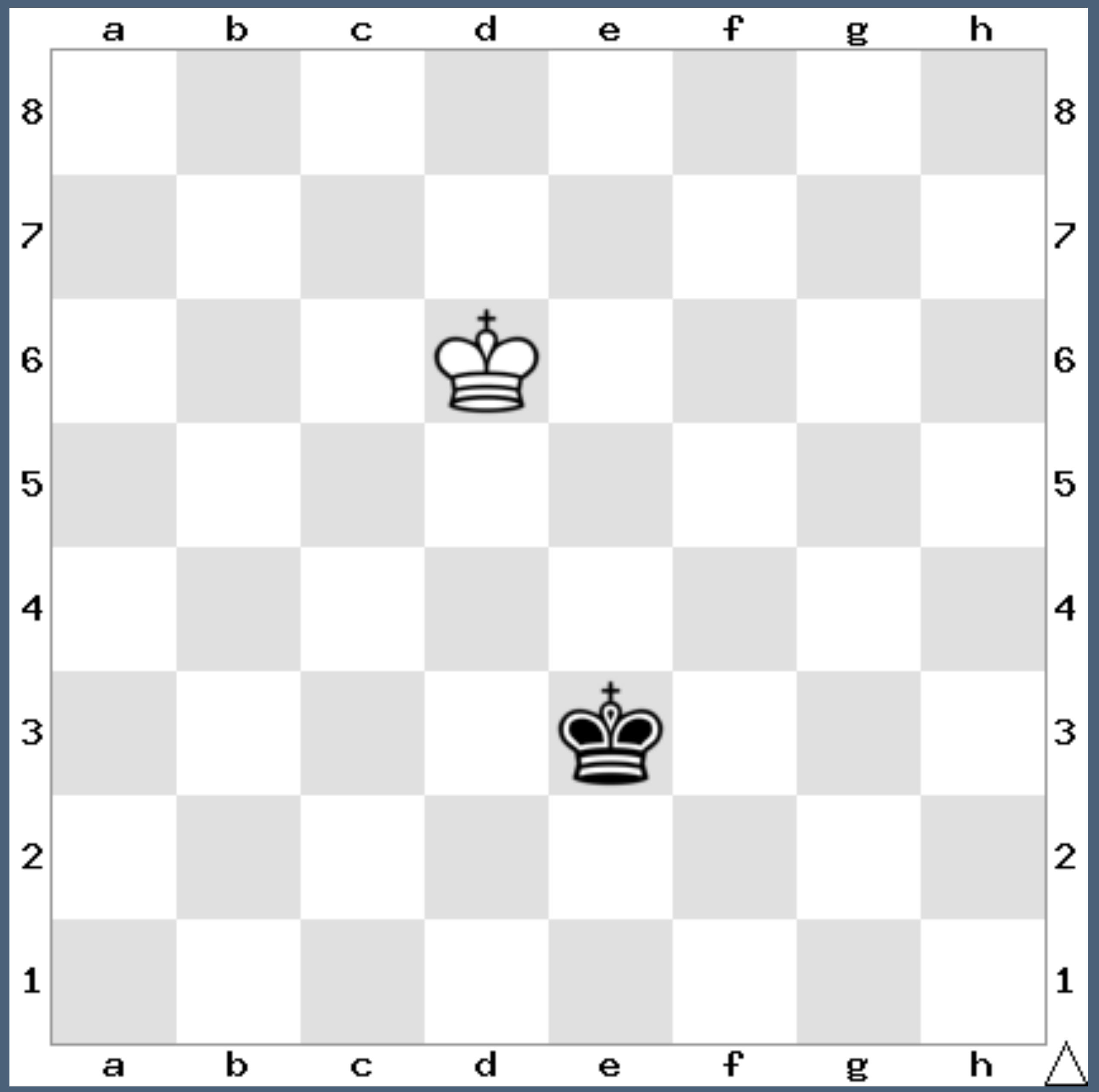
(b1) Average action value Q
(b2) Exploration term U
(b3) Upper confidence bound $Q + U$
(b4) Visit count N

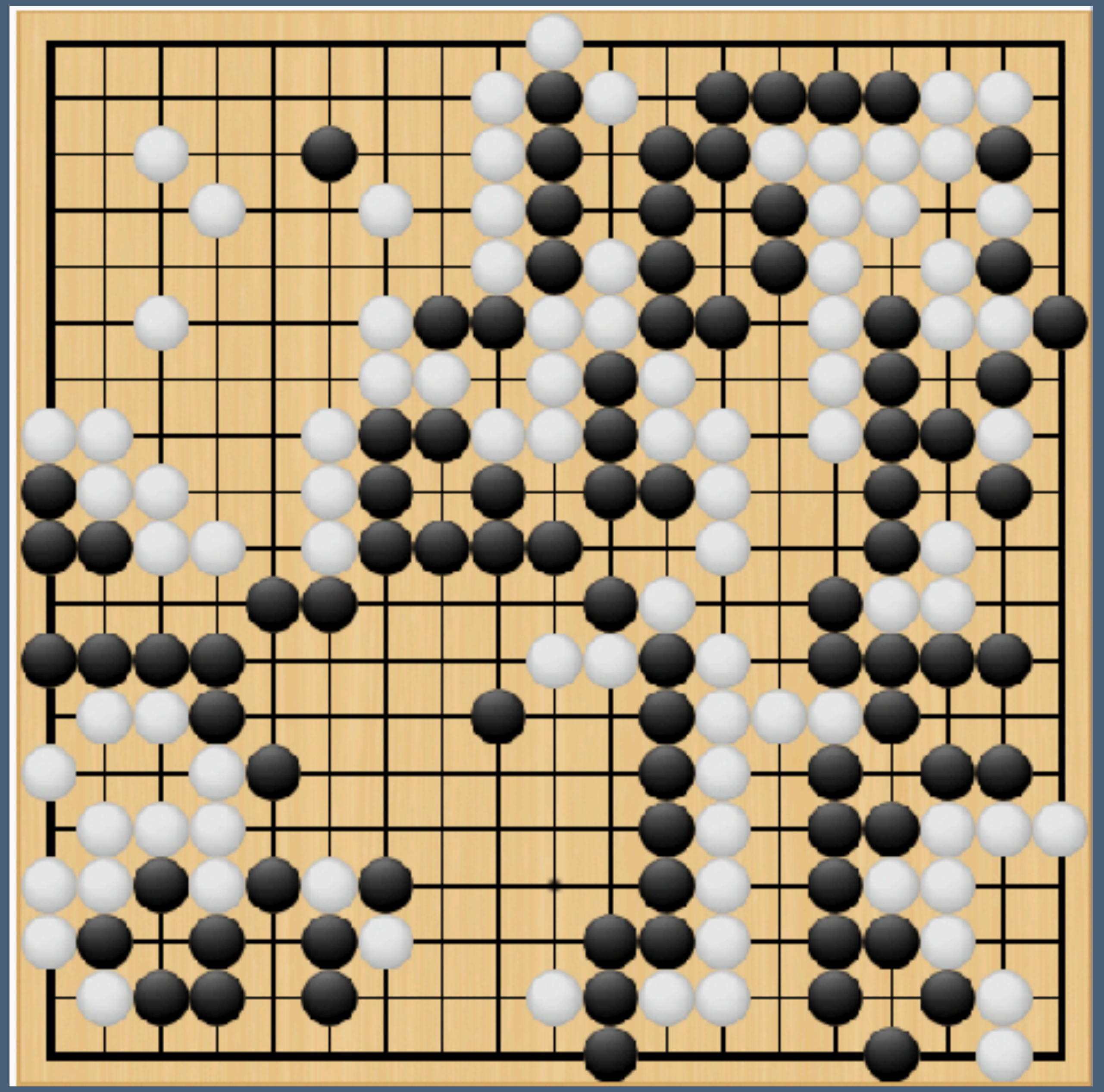
Part 2 - Go

- Differences from chess to Go
- The KataGo program
- Endgame puzzles and Decomposition Search
- Experiments and Results
- Discussion

From Chess to Go

- Chess: game becomes simplified in endgame, “converging game”
    - Fewer pieces, fewer positions
    - Can build complete tablebases
- Go: game does not become simpler, “diverging game”
    - Cannot build databases
    - How to evaluate perfectly?



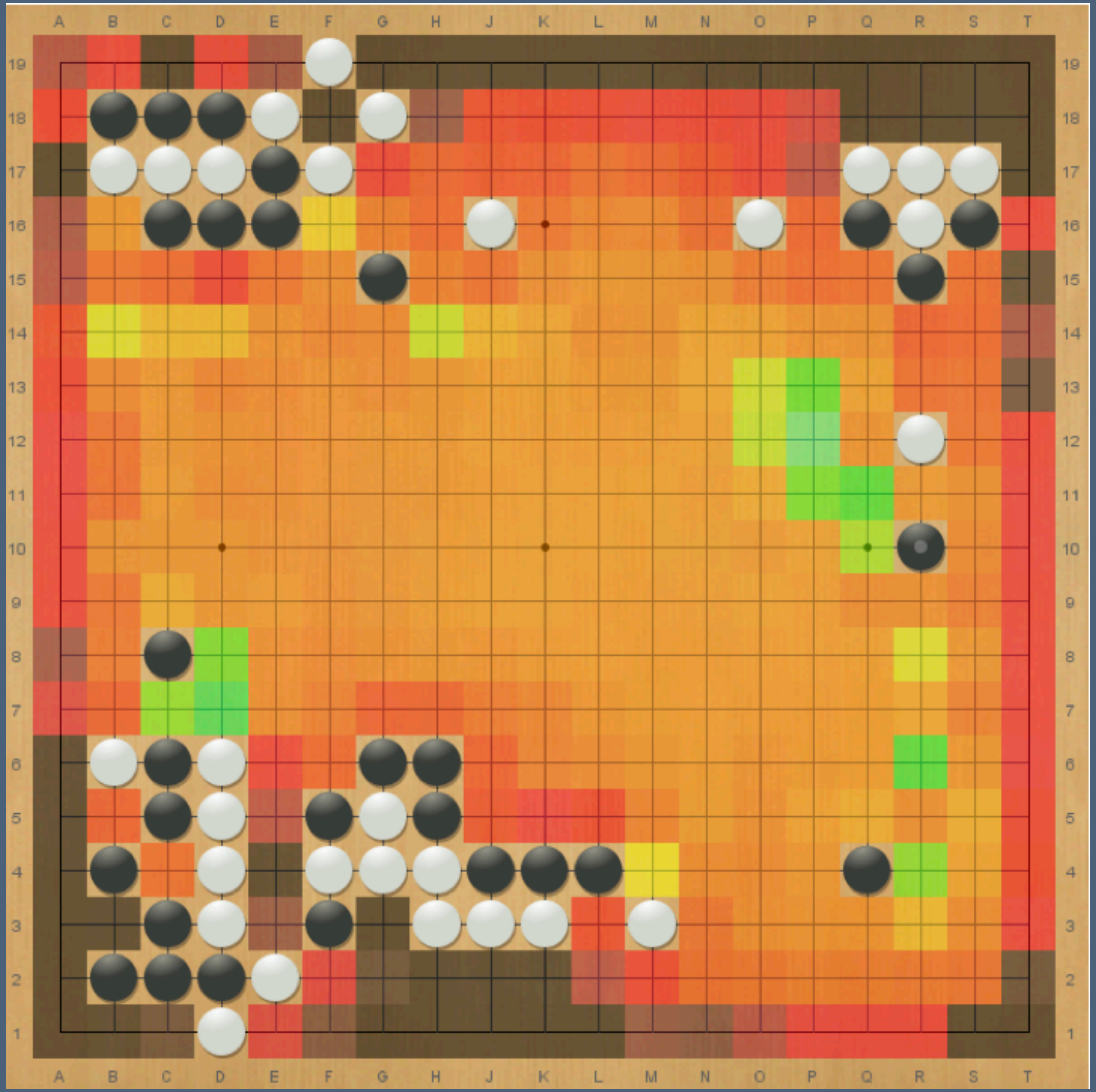


Solving Go Positions

- How can we solve Go?
- Of course, regular Go is much too hard
- We can solve only in special cases:
- Small boards, 5x5, 6x6, 7x7 killall Go
- Endgames with special mathematical structure
- AlphaZero type programs play very strongly, but not perfect
- In this work, we test AlphaZero type program KataGo against a perfect player in Go endgames

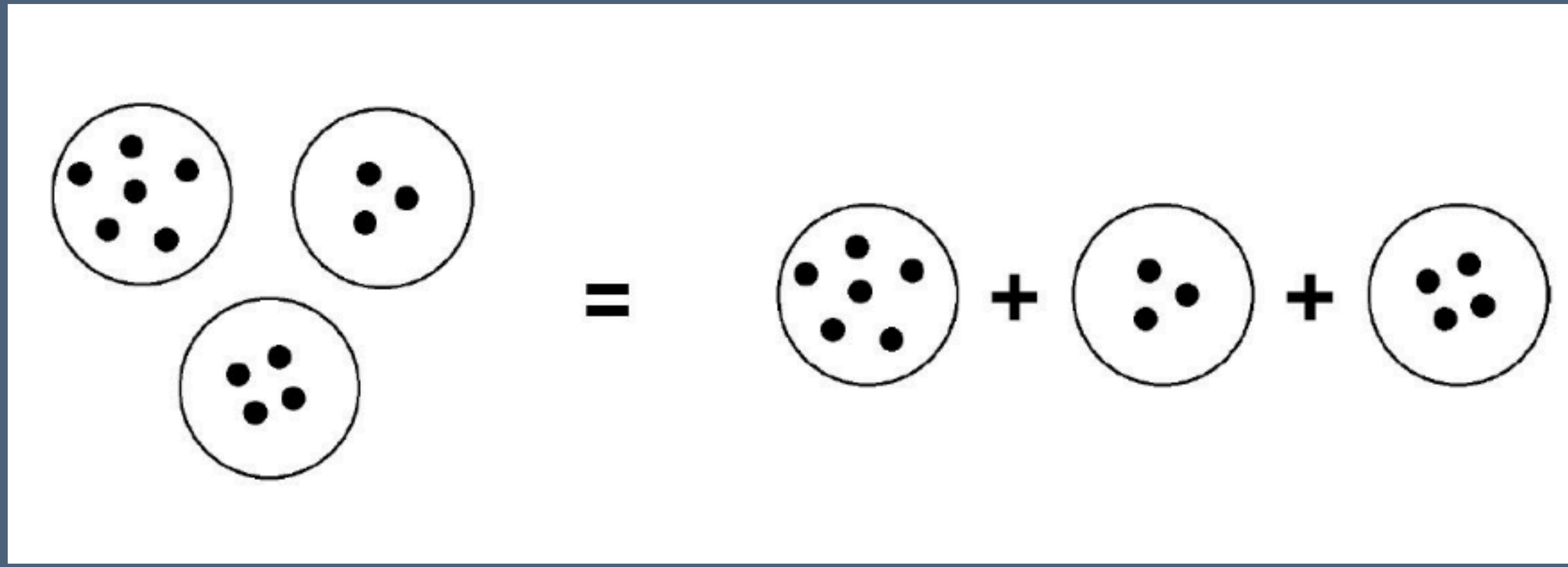
KataGo

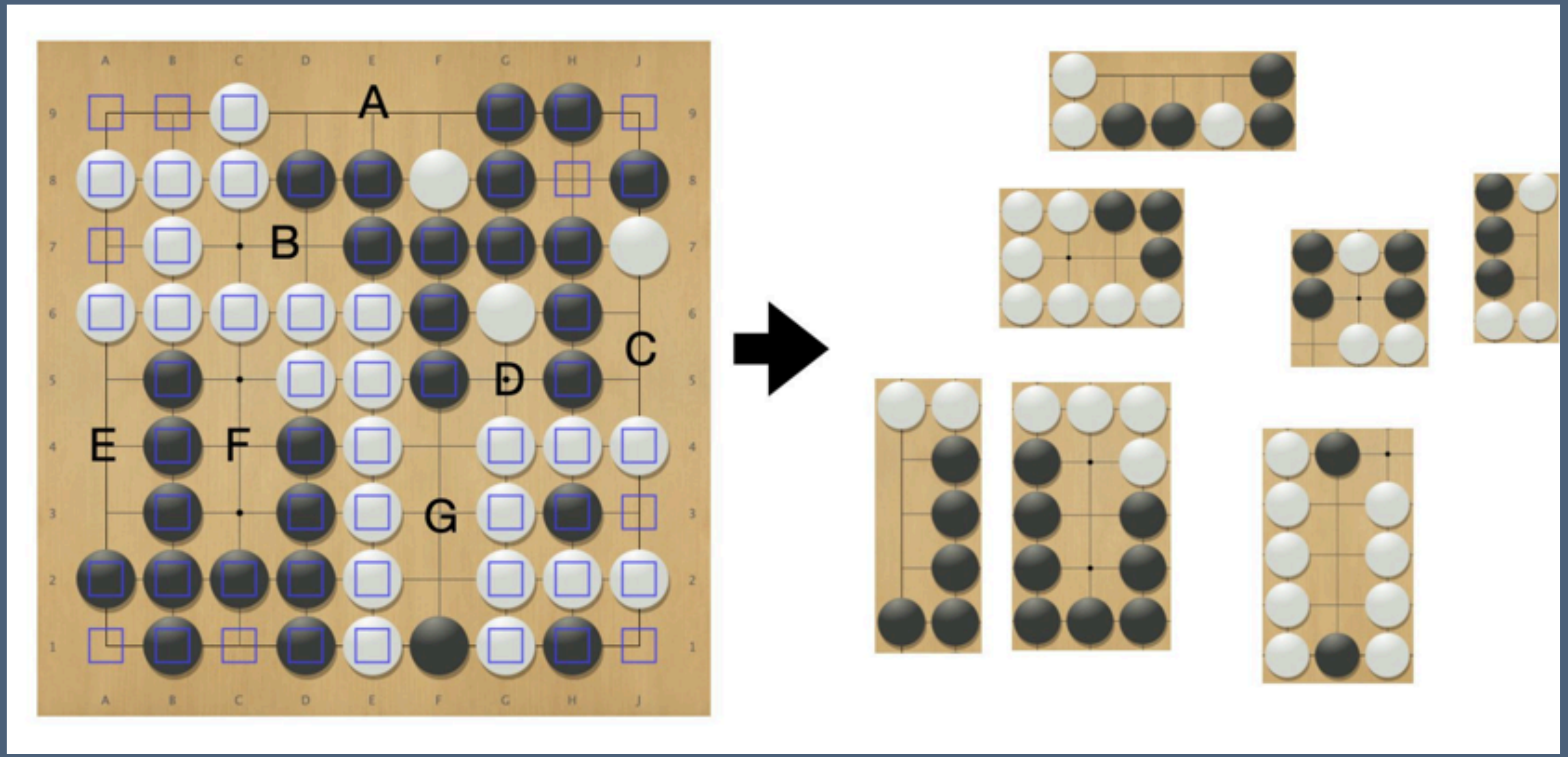
- Strongest open source Go program
- Based on AlphaZero
- Improvements
  - Better training efficiency
  - Can play different board sizes, komi, rules
  - Auxiliary targets: territory ownership and score



Combinatorial Game Theory (CGT)

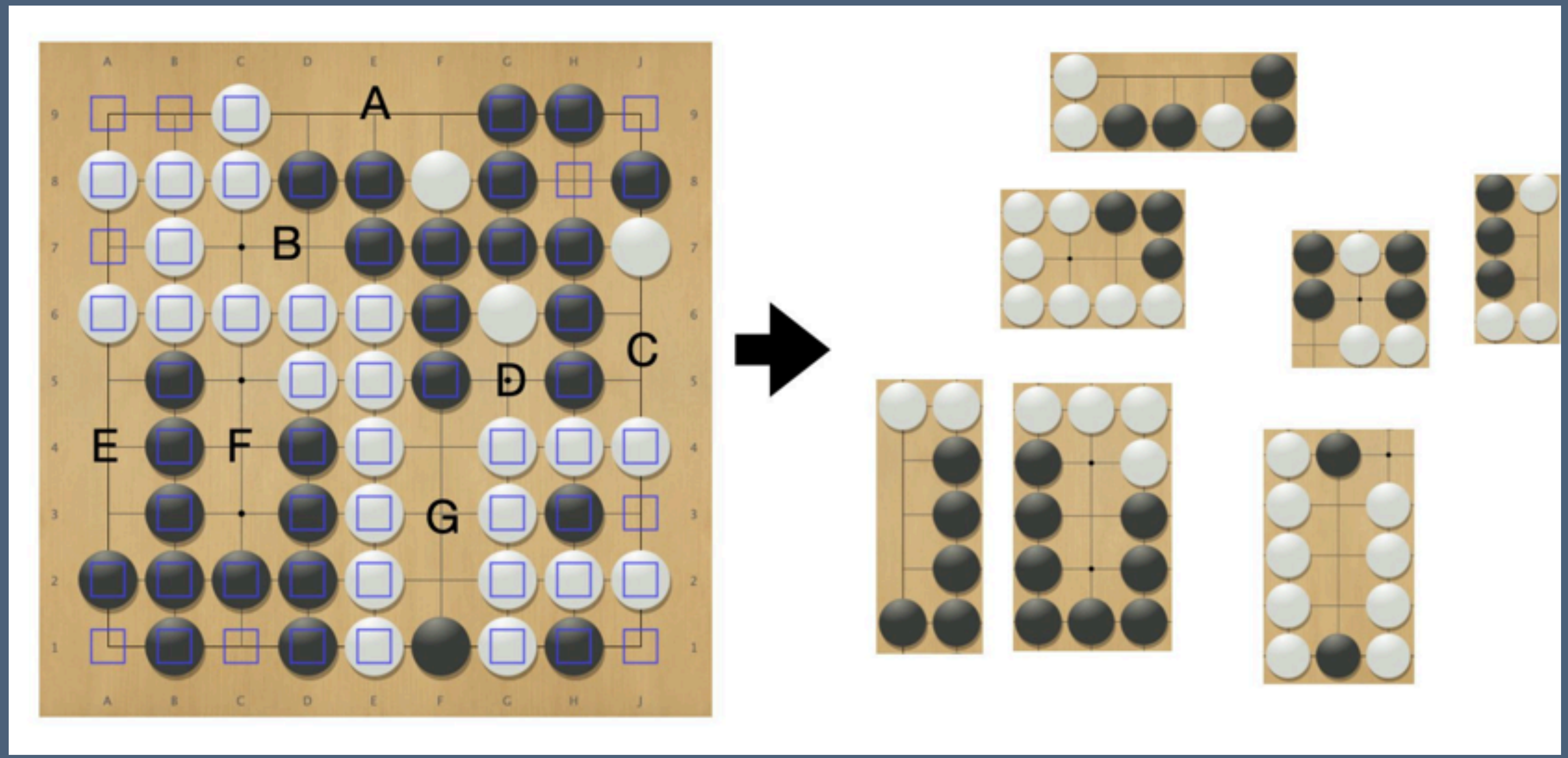
- Mathematical theory for games
- Applies to games that consist of independent subgames
- Can solve some of these games very efficiently
- Much faster than “full-board” minimax search
- Find optimal play in a sum game
- A game that consists of independent subgames
- Examples: Nim, Go endgames





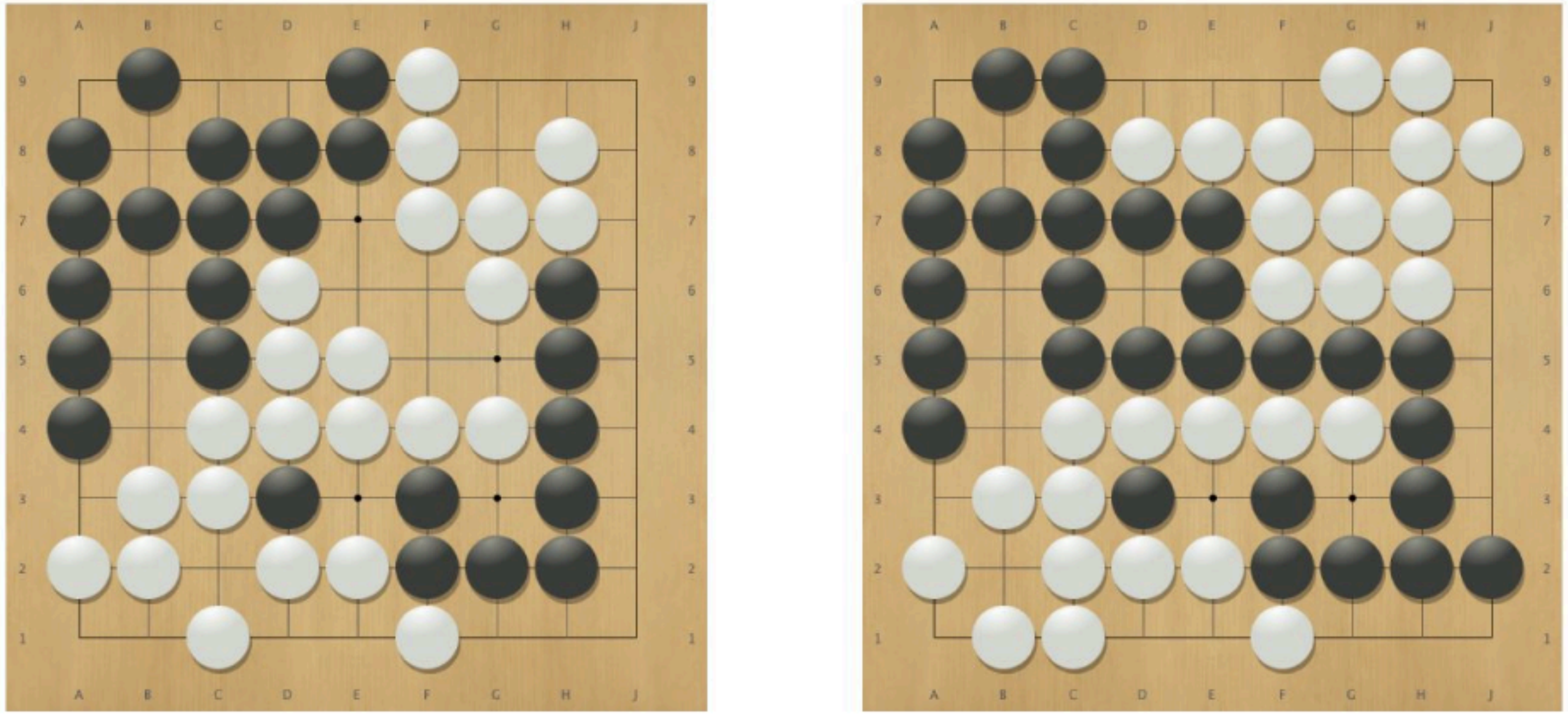
Decomposition Search

- A game tree search method based on combinatorial game theory (Müller 1995)
- Application: Go endgames
- Identify subgames
- Local combinatorial games search
- Find the combinatorial game evaluations
- Select optimal moves



Go Endgame Problems

- E. Berlekamp and D. Wolfe, Mathematical Go: Chilling gets the last point (1994)
- Original set of 22 problems, White wins by 1/2 point in all cases
- Human analysis of Go endgames
- Independence of subgames verified by hand
- Modified problems:
  - Same endgame values
  - Territories modified to allow computer analysis
  - Can be solved by Decomposition Search



Original Modified

Extend Dataset - Perfect Games

- Start with modified problems C.1, C.2, C.3, C.6, C.7, C.8, C.9, C.10, and C.21
- Use exact solver self-play to generate one full game
- Adjust the komi when a stone is captured.
- A total of 126 more test positions
- Exact solver used to create set of all optimal moves in each position

a) Modified problem C.7, white plays A b) After white's move, black to play c) After black's move, white to play

Start of a "perfect game"

Experiment - Testing KataGo on Endgames

- Similar to chess experiment
- Two versions of neural network, strong and weak
  - Strong: 18 blocks and 384 channels
  - Weak: 6 blocks and 96 channels, less training
  - Run KataGo with and without search
- Generate a move, check if it is in set of optimal moves
- (Not in this talk: two different definitions of optimal moves)

KataGo with KaTrain Interface

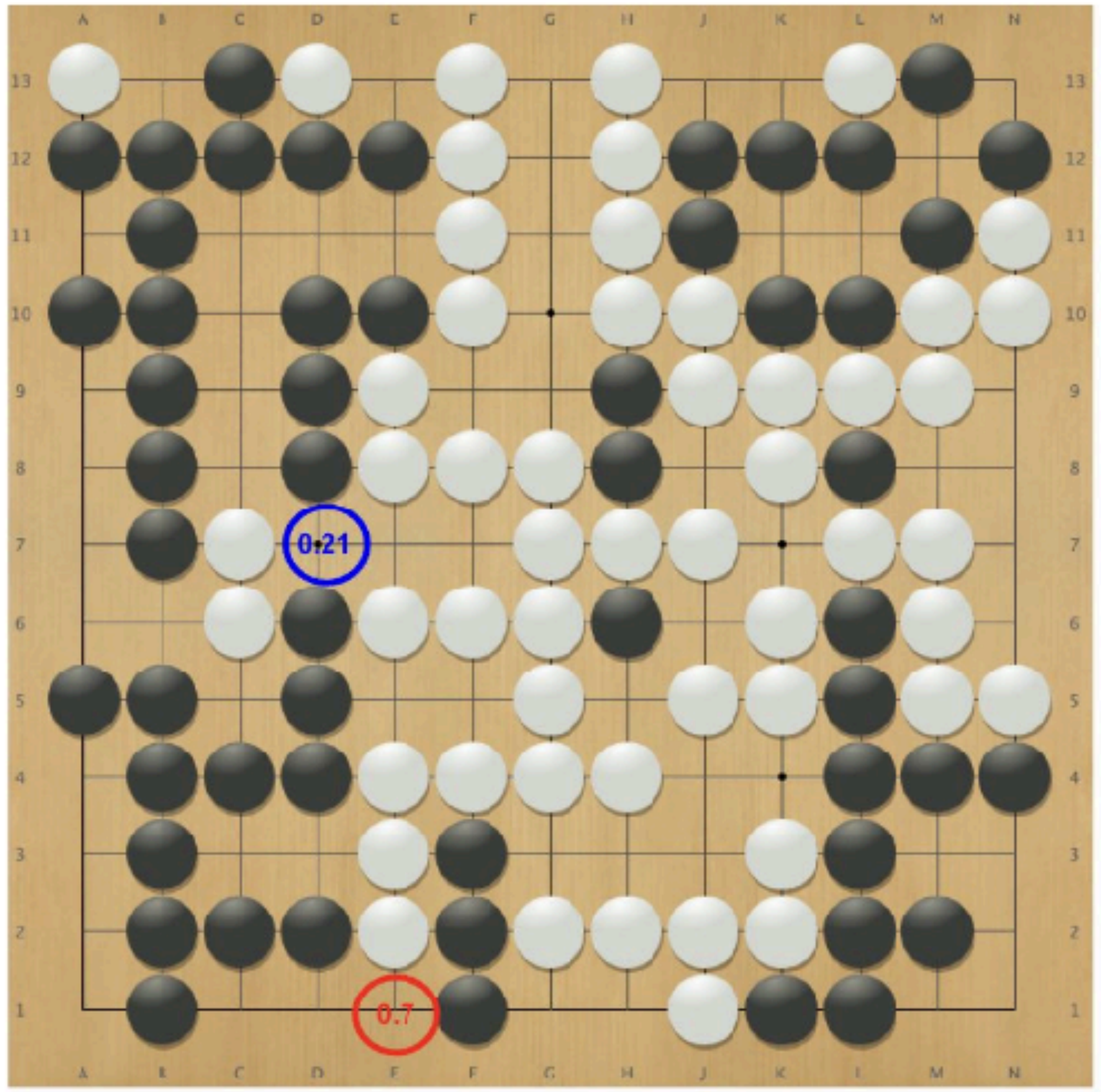
Summary of Results - No search

- Many problems are difficult for KataGo policy

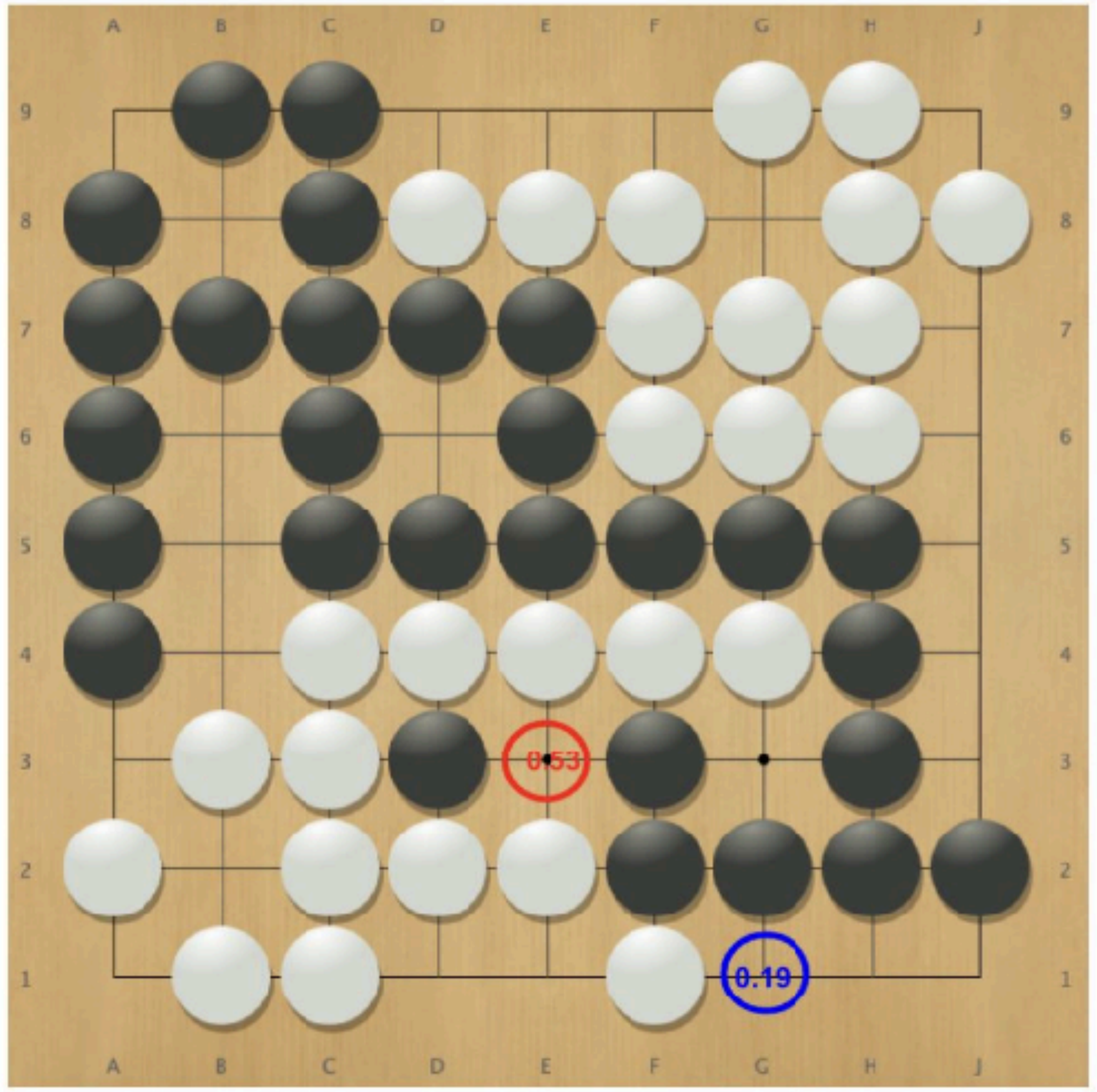
Name of Dataset	Number of Endgames	Total Number of Correct Moves with Success Rate (%)	
		Weak policy (min<avg<max)	Strong policy (min<avg<max)
Original	22	7 < 8.8(40%) < 11	10 < 12.8(58.2%) < 14
Modified	22	7 < 7.8(35.5%) < 10	13 < 13.6 (61.8%) < 14
Perfect games	126	97 < 98.8**(78.4%)** < 102	118 < 118.8**(94.3%)** < 120

Examples of Policy Errors

- Left side
  - Two valuable moves
  - Value are similar but different
  - Only D7 wins
  - Kata Go policy prefers E1
- Right side
  - Much simpler case, KataGo policy is wrong
  - KataGo likes capturing moves too much



(a) White to play. KataGo's move is $E1$ and the only winning move is $D7$.



(b) White to play. KataGo's move is $E3$ and the only winning move is $G1$.

Difficult case

Simpler case

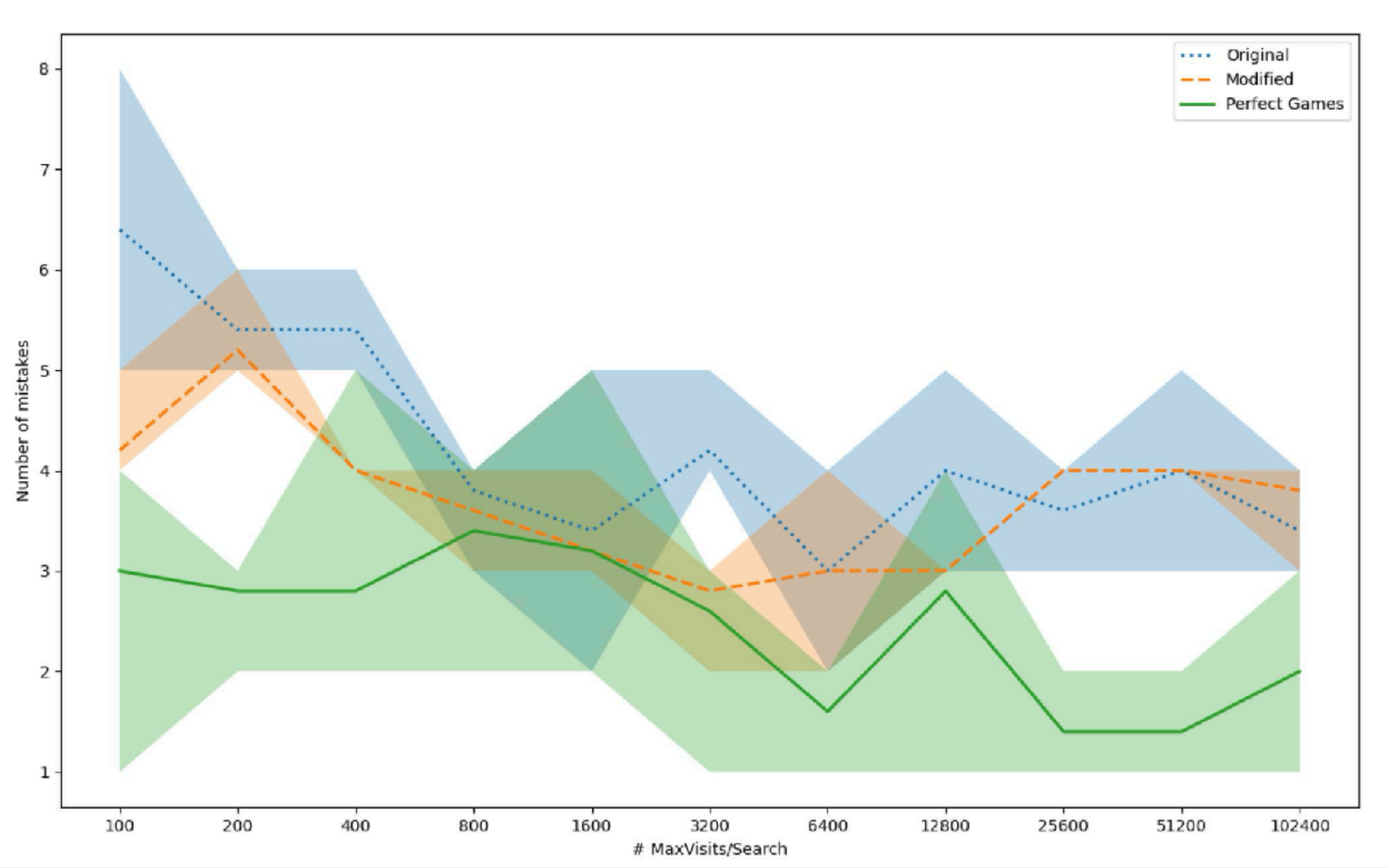
KataGo with Small Search

- 100 nodes search
- All results improve a bit
- Still a number of errors remain

Name of Dataset	Number of Endgames	Total Number of Correct Moves with Success Rate (%)	
		Weak policy + MaxVisits = 100 (min<avg<max)	Strong policy + MaxVisits = 100 (min<avg<max)
Original	22	7 < 9.6(43.6%) ↑3.6% < 12	15 < 16.2(73.6%) ↑**15.4%** < 17
Modified	22	7 < 9(40.9%) ↑5.4% < 11	16 < 16.8(76.4%) ↑**14.6%** < 17
Perfect games	126	104 < 105.8(84%) ↑5.6% < 108	123 < 123.8(98.3%) ↑4% < 124

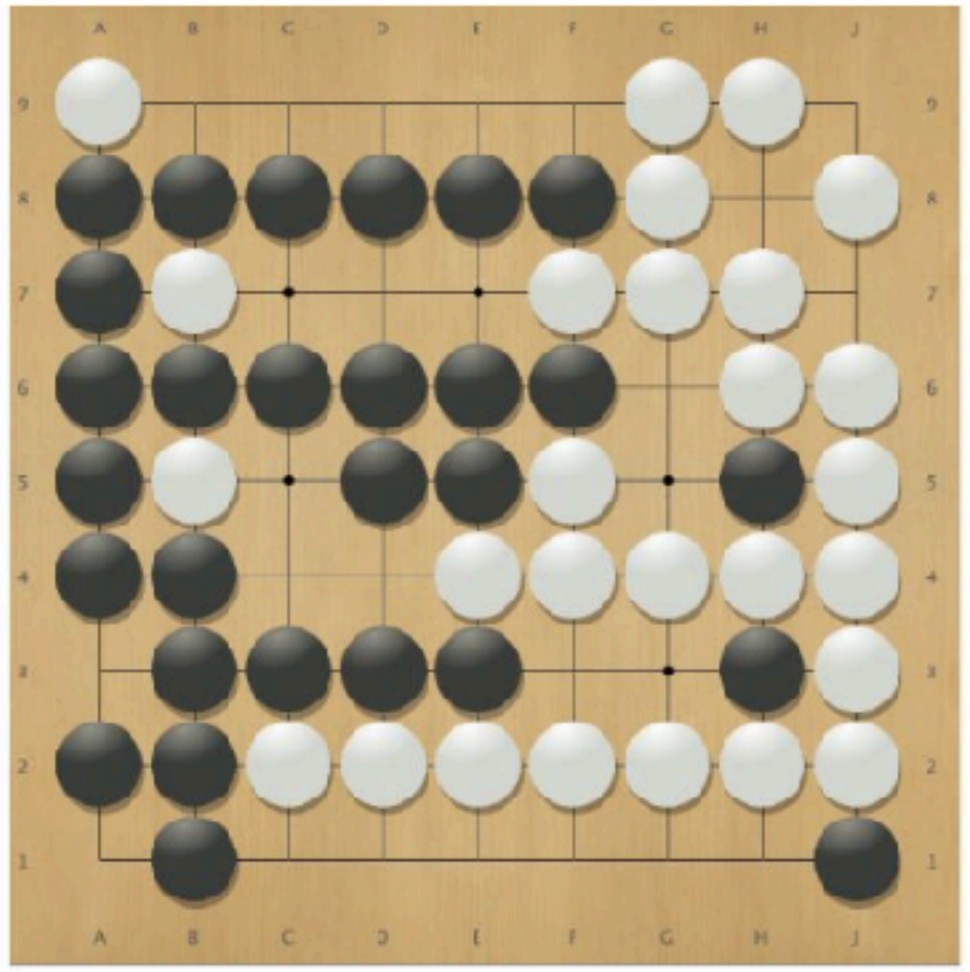
Scaling Up the Search

- Each data point = double the search
- From 100 to >100k nodes
- Small improvements, not consistent
- Many problems are still unsolved

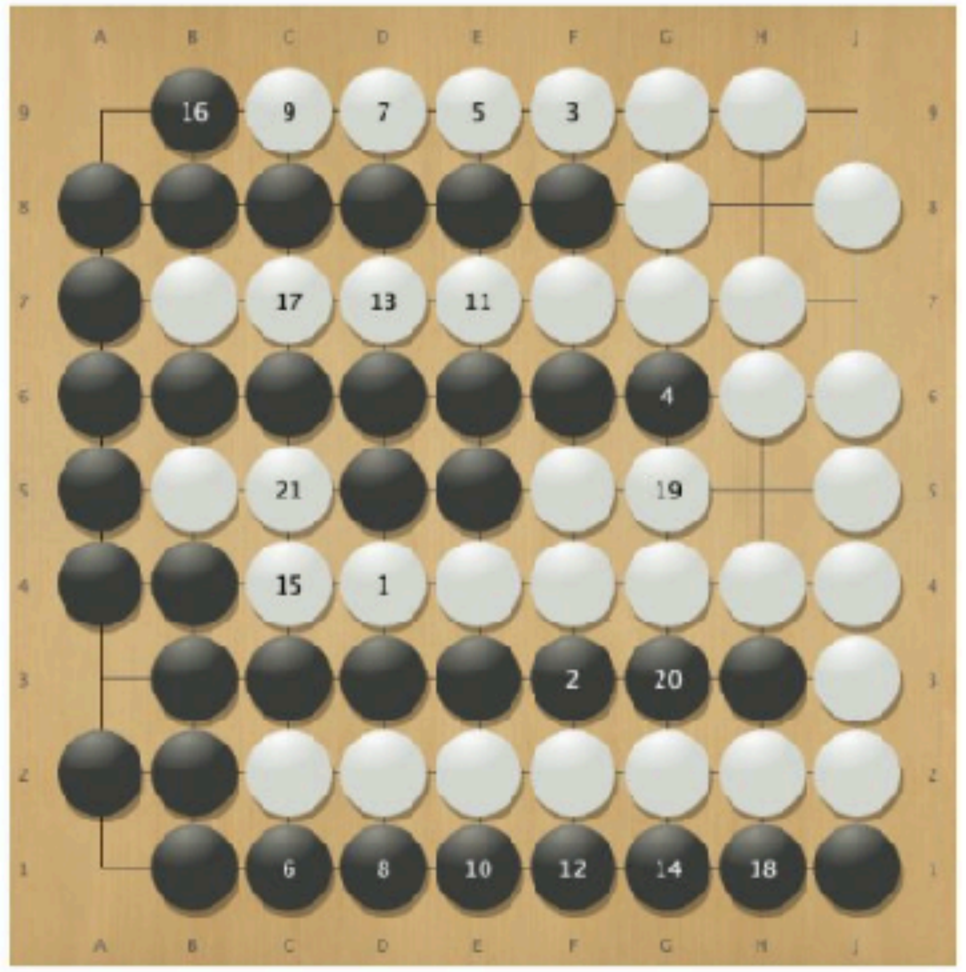


Playing Matches between Exact Solver and KataGo

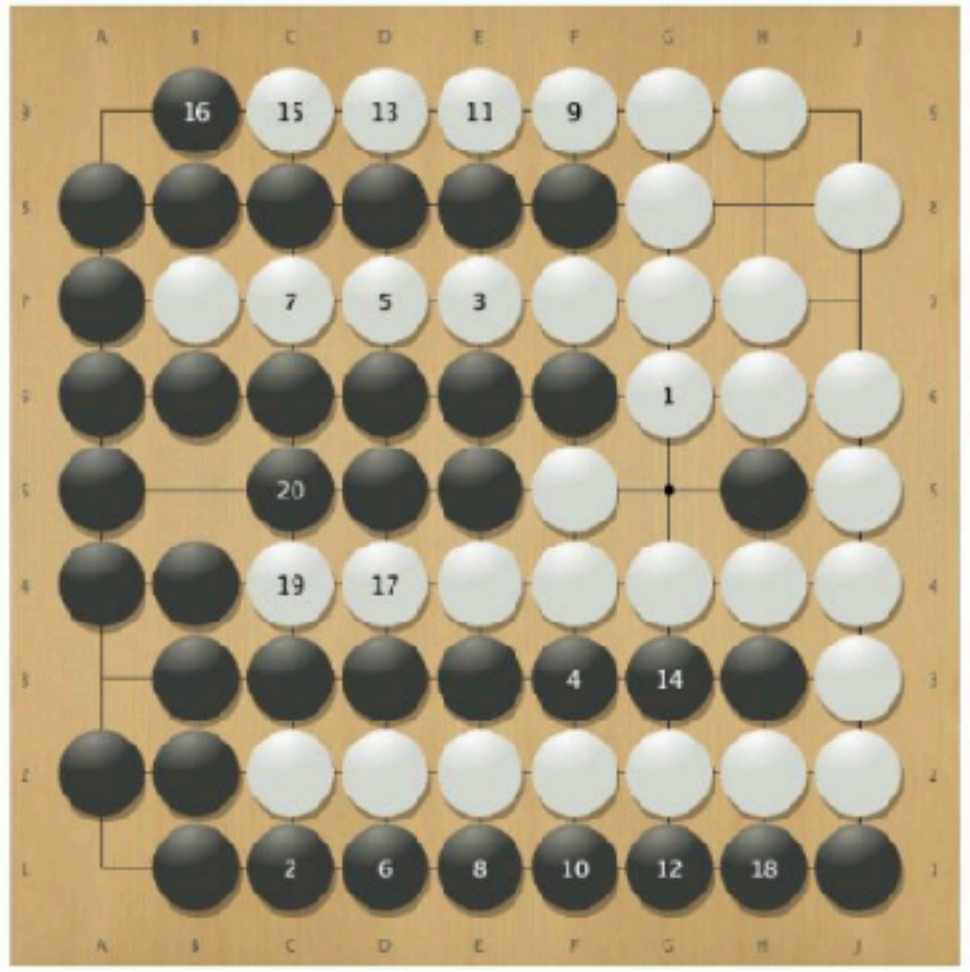
- 120 positions from "perfect games"
- White wins by 0.5 points in each case
- Exact Solver vs KataGo, 100 nodes
- Exact Solver is white: 120 wins
- KataGo is white: 109 wins, 11 losses
- Example: KataGo loses



(a) A simple 9×9 perfect game from modified problem $C.2$.



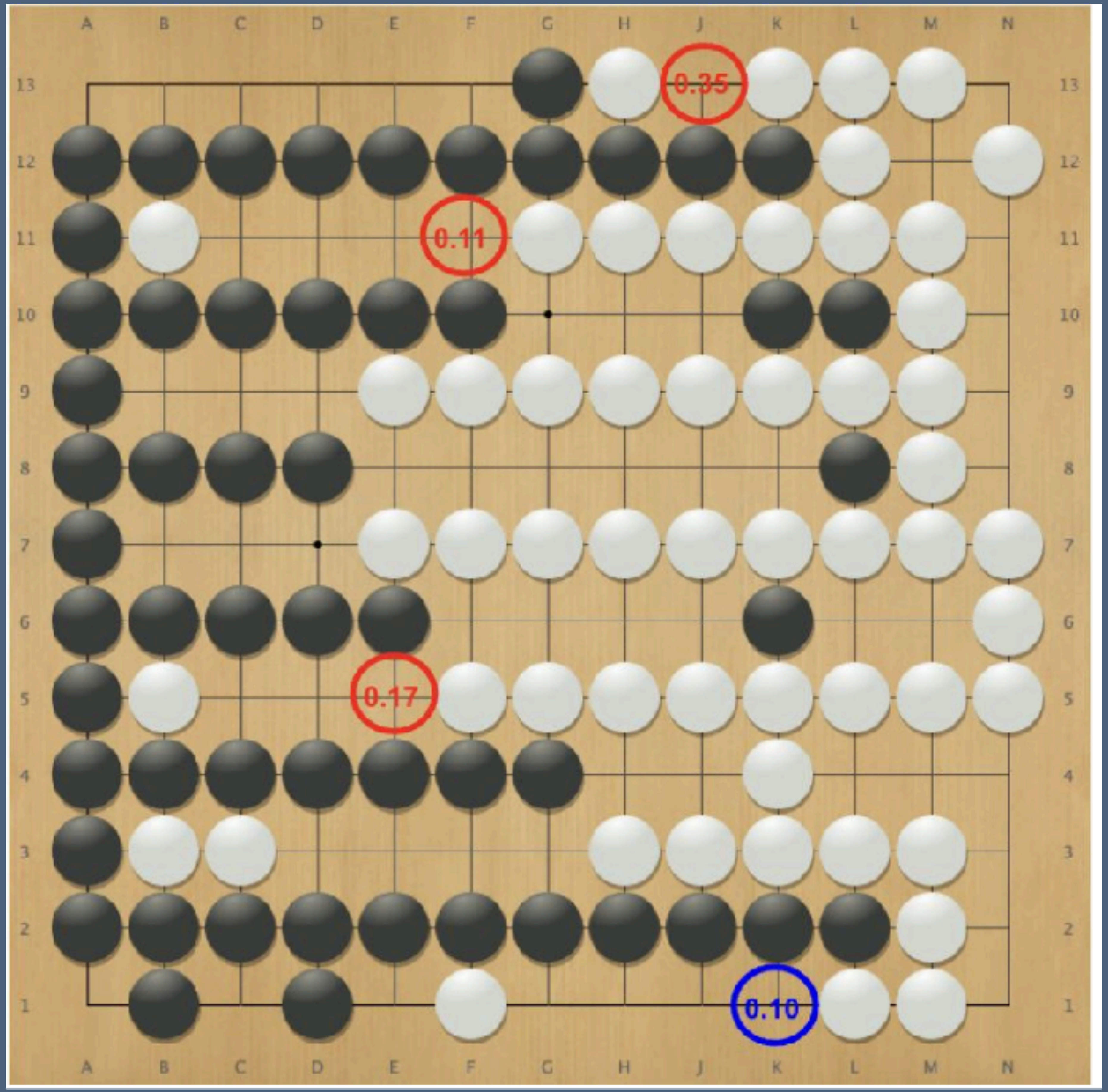
(b) Exact solver wins as white against KataGo.



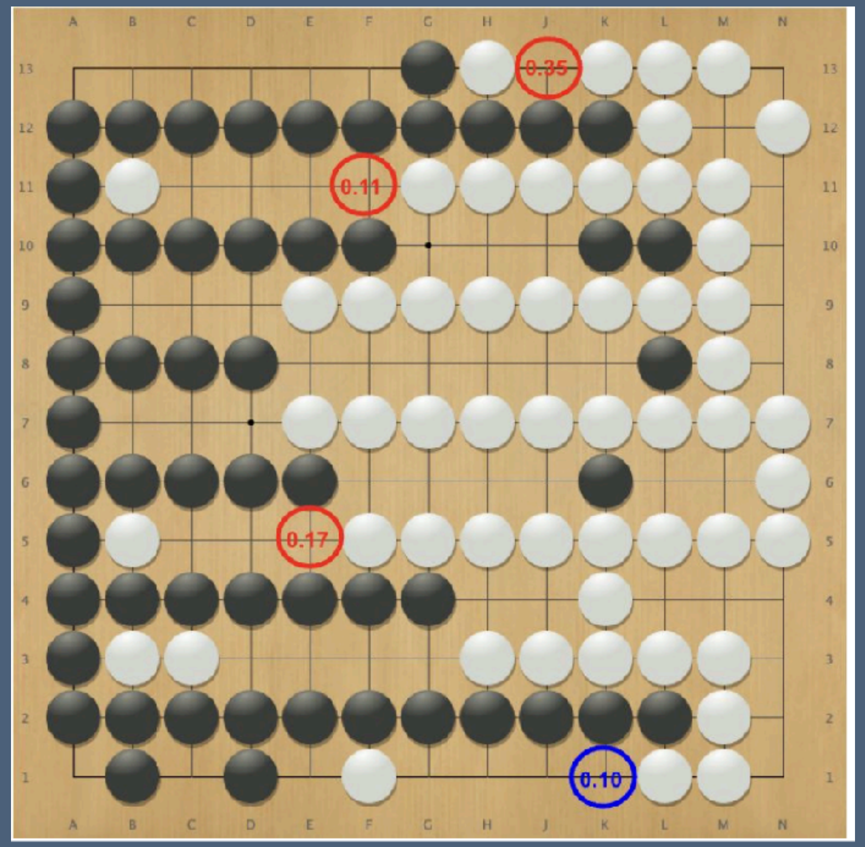
(c) KataGo loses as white against exact solver.

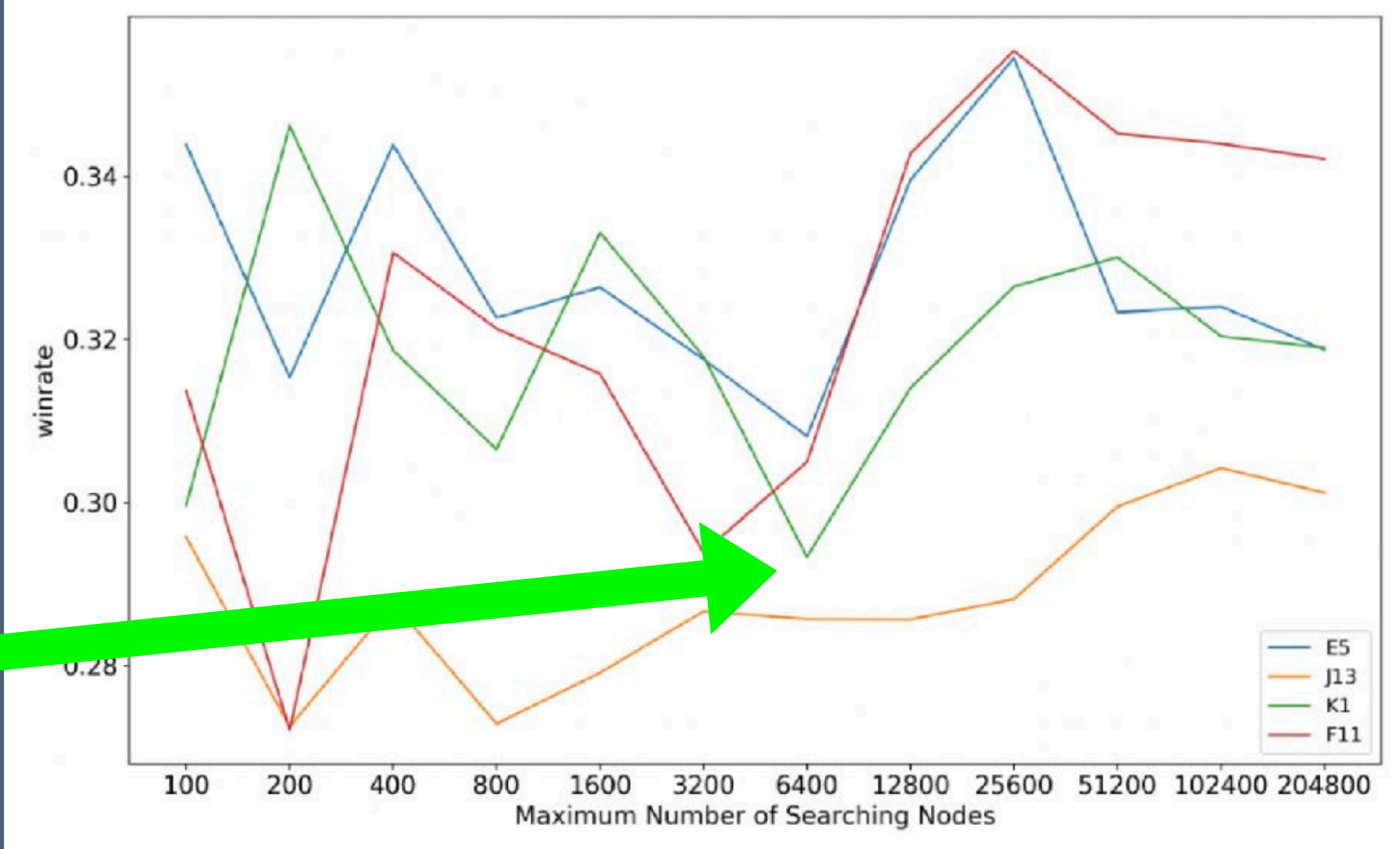
Example: Scaling to Larger Search

- Blue move on K1: only winning move
- Red moves: losing moves
- Policy network:
  - 10% on blue move
  - 35%, 17%, 11% on the three red (losing) moves
- Extend search up to 200k nodes
- Never finds blue move,
- Search switches between three red moves



Same Example - Far from Solved...





- Winrate around 30-34%
- True winrate should be 100%
- Cannot separate winning move K1 from losing moves

Example: a Very Simple Error in Early Training

- Example with Weak policy
- Only 3 points remain
- Very simple case, D7 is better
- Policy net likes G9 better...

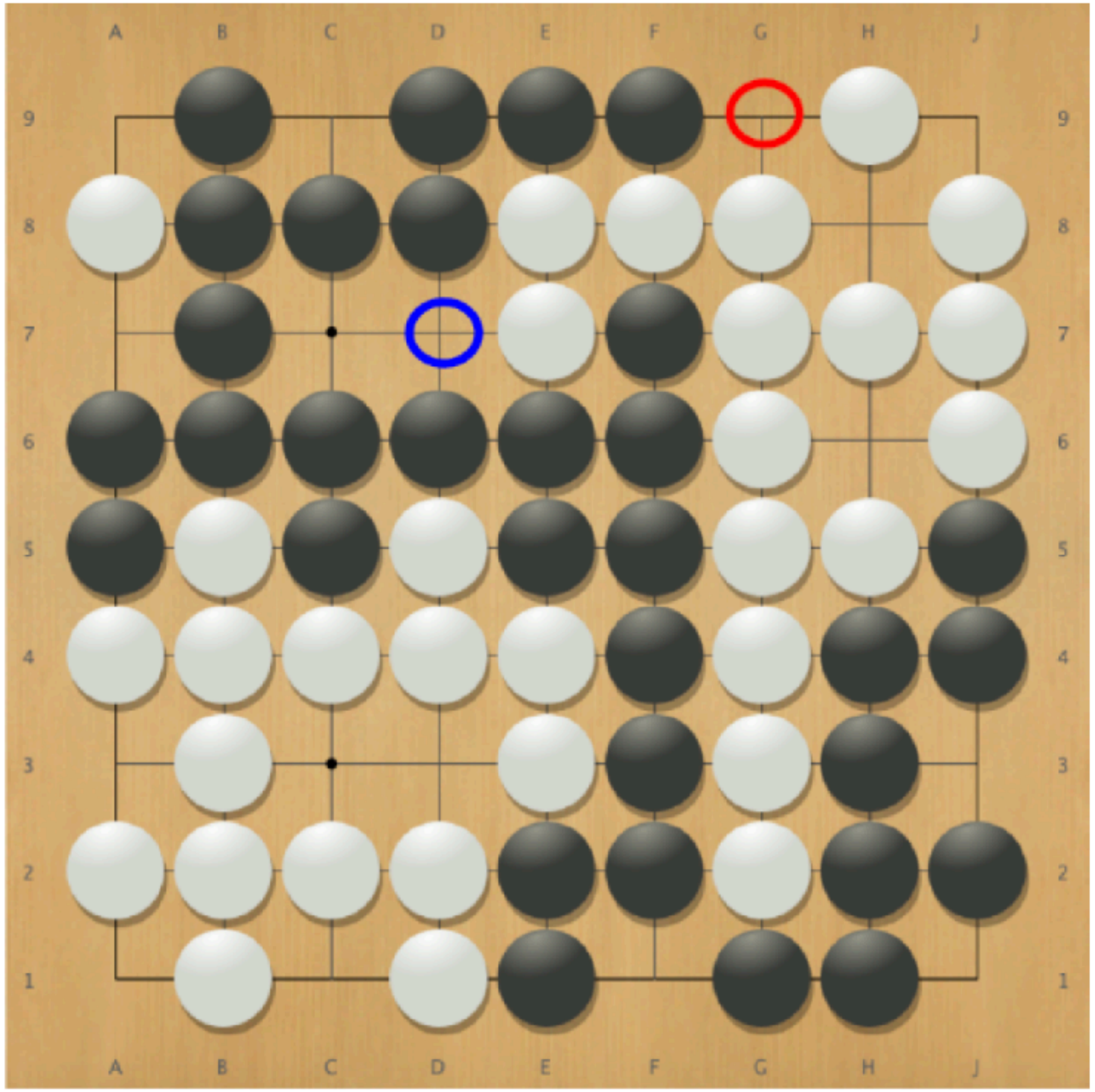


Figure 4.6: A small size (size 3) endgame where KataGo's weak policy makes mistake. Here, *D7* is the only winning move where KataGo's move is *G9*.

Summary

- Are Alpha Zero type programs close to perfect play?
  - Two case studies, chess and Go
  - Clear answer: No
- Very strong playing performance, far beyond human
- Limitations seen in difficult game positions for which we have exact, perfect information
  - Chess endgame tablebases
  - Go endgame puzzles
- Search using a strong network cannot overcome mathematical analysis
- A lesson for other applications where strong guarantees of correctness are needed?
  - Medicine, engineering of safety-critical systems, ...