# Expected Work Search: Combining Win Rate and Proof Size Estimation

Owen Randall, Martin Müller, Ting-Han Wei, and Ryan Hayward

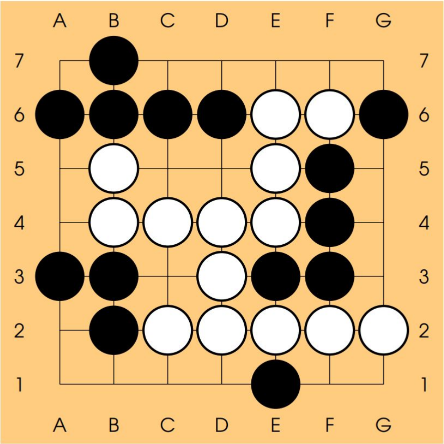
University of Alberta

# Introduction

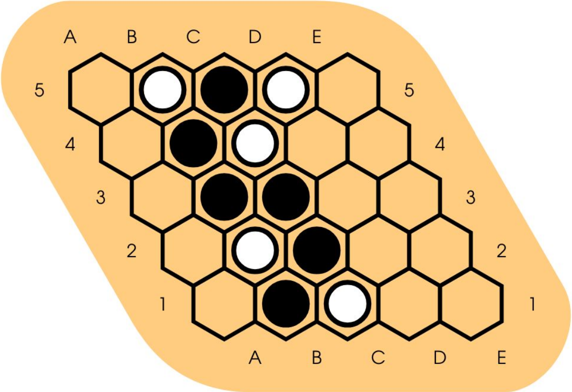
- Expected Work Search
    - 2-player perfect information solving algorithm
- Combines existing heuristics
    - Estimates **win rate** (Monte Carlo Tree Search)
    - Estimates **proof size** (Proof Number Search)
    - Minimizes **Expected Work** to efficiently find proofs
- Results
    - **Orders of magnitude** faster than tested programs solving Go and Hex
    - First program to **solve 5x5 Go** with positional superko rules

# Motivation

- Search has many modern applications
  - Optimization
  - Logistics
  - Artificial Intelligence
- Solving games is a convenient test bed
  - Well behaved
  - Easily scalable
- Evaluated EWS by solving Go and Hex
  - Simple rules, complex state space



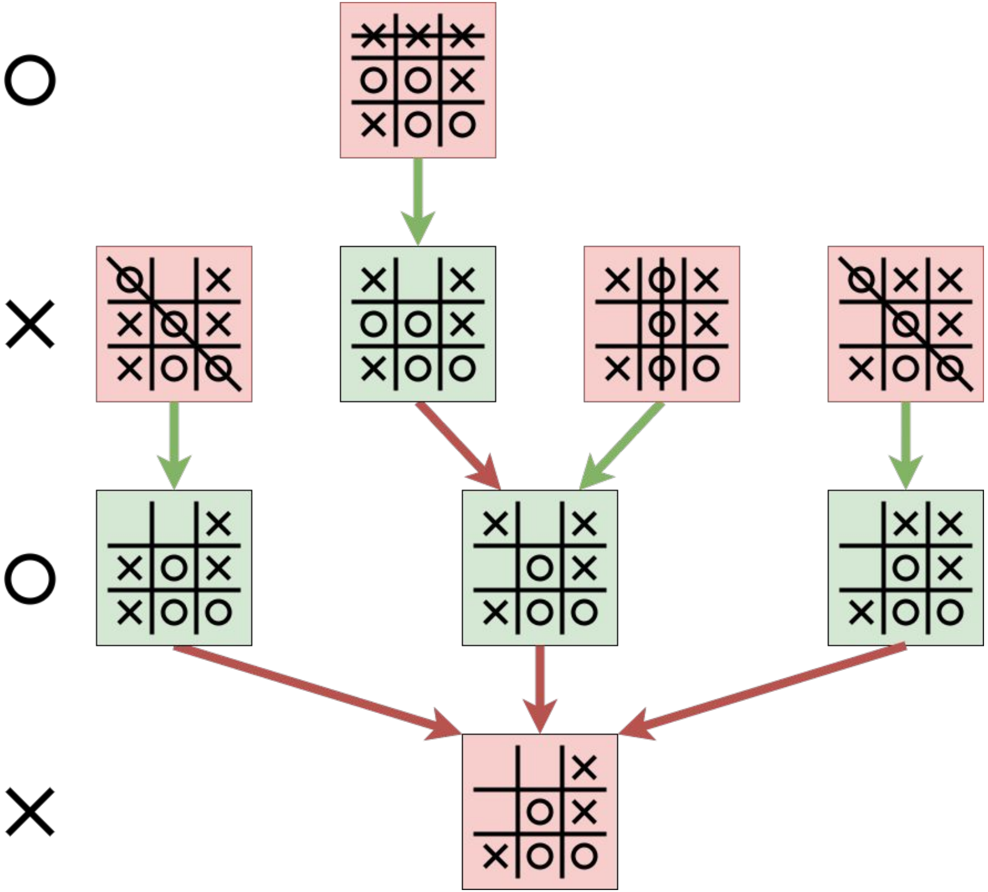

Example 7x7 Go position




Example 5x5 Hex position

# Solving Games

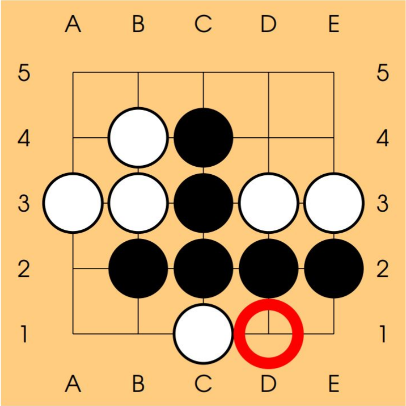
- Determine who wins under **optimal play**
- Requires a **proof tree**
- Positions with **all** losing moves are **losing positions** (AND)
- Positions with **any** winning move are **winning positions** (OR)



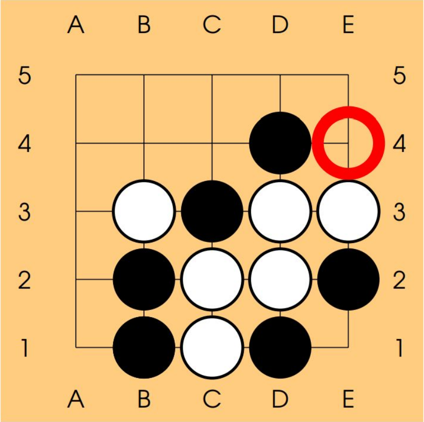
Tic Tac Toe solving example

# Heuristics

- Win rate estimation
    - Prioritizes strong moves
    - No incentive to find small solutions
- Proof size estimation
    - Prioritizes small solution size
    - Often suggests bad moves
- Both have strengths and weaknesses
- EWS combines both to eliminate weaknesses



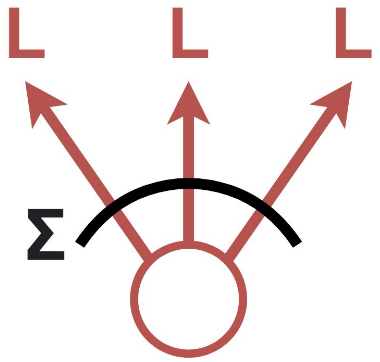

Example where win rate fails



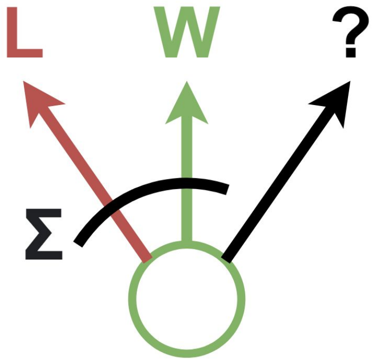

Example where proof size fails

# Expected Work

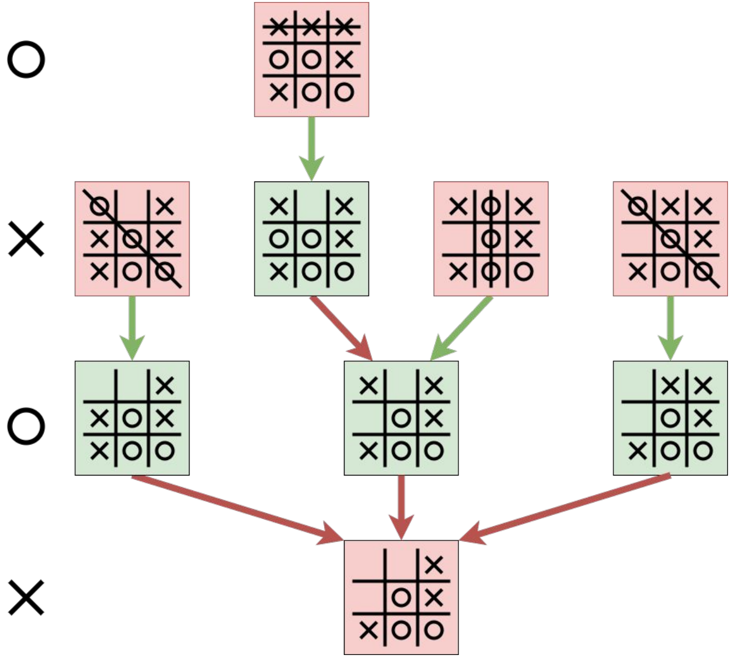
- A position is either **winning** or **losing**
- Estimate the expected amount of work to solve
- Search positions with low Expected Work




Losing position (AND)

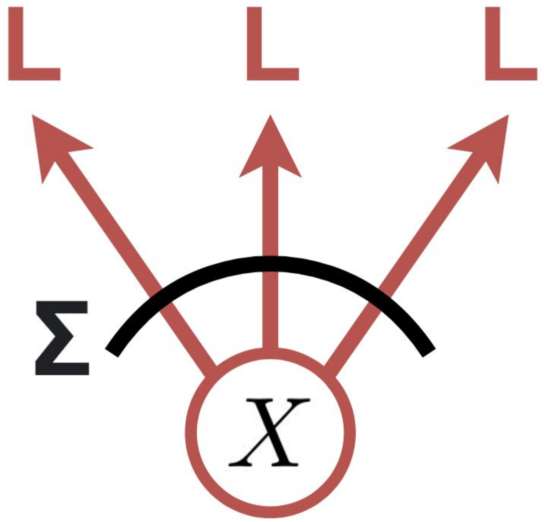



Winning position (OR)



# Expected Work Calculation

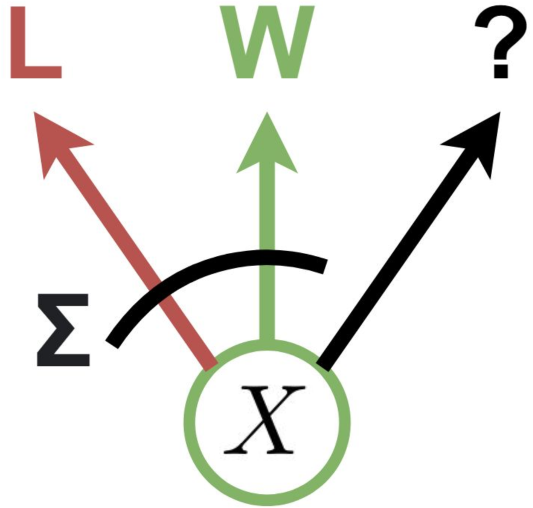
- **Losing case:**
- All children must be searched and proven
- Sum of winning children's EW



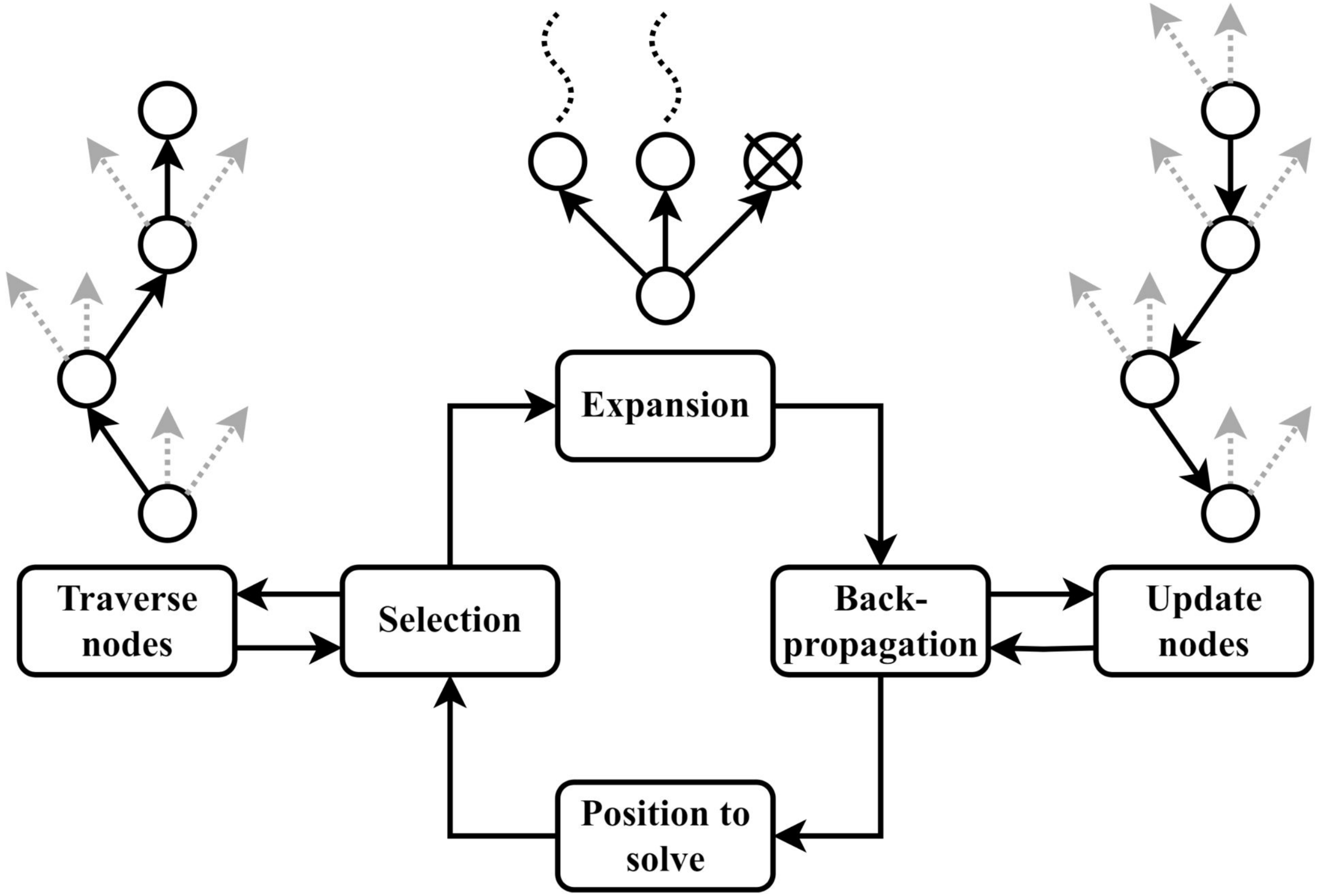

$$EW_{loss}(X) := \sum_{i=1}^{n} EW_{win}(C_i)$$

# Expected Work Calculation

- **Winning case:**
- **Weighted** sum of losing children's EW
- Multiply by the probability that a given child is searched




$$EW_{win}(X) := \sum_{i=1}^{n} \left( EW_{loss}(C_i) \cdot \prod_{j=1}^{i-1} WR(C_j) \right)$$




Expected Work Search

# Selection

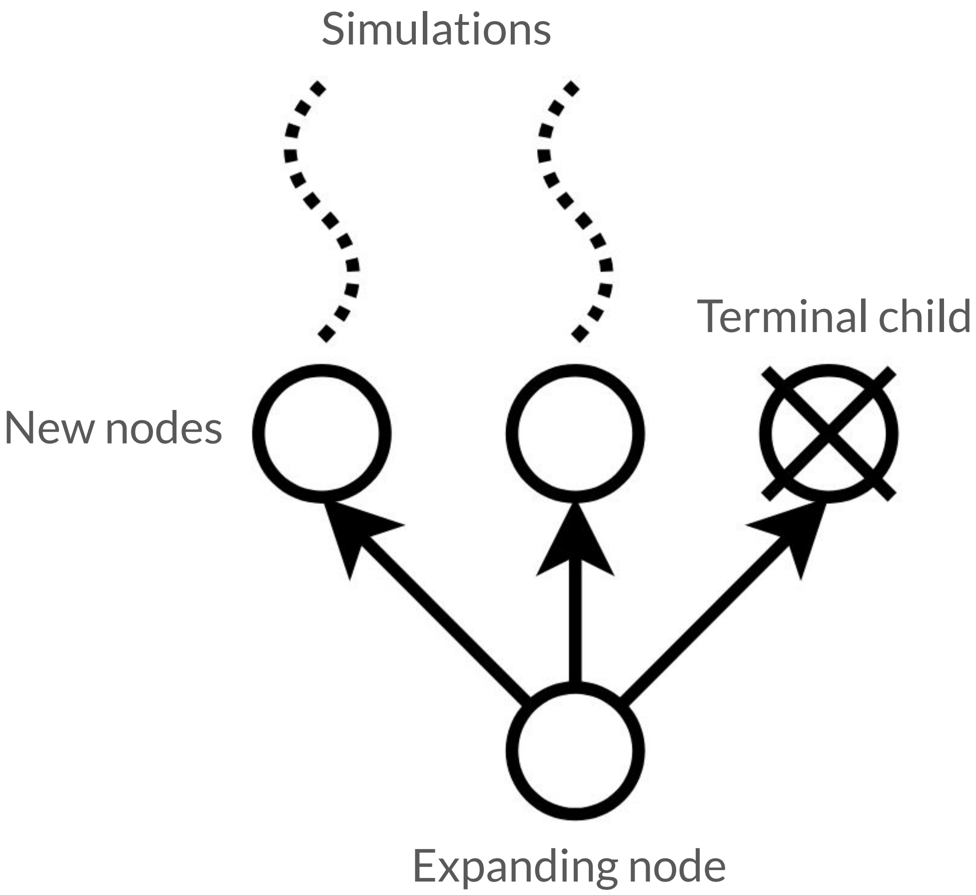
- Traverse the search tree until a leaf node is found
- Sort children in ascending order of the following formula:

$$EW_{loss}(X)/(1-WR(X))$$

- Where X is a child, WR is win rate, and EW is expected work
- Always select the first ordered child
- Move ordering of children determines $EW_{win}$
    - **This ordering minimizes $EW_{win}$**

# Expansion

- Create new nodes
- Check for terminal children
- Initialize WR and EW with random simulation results
- Return if the expanded node is proven



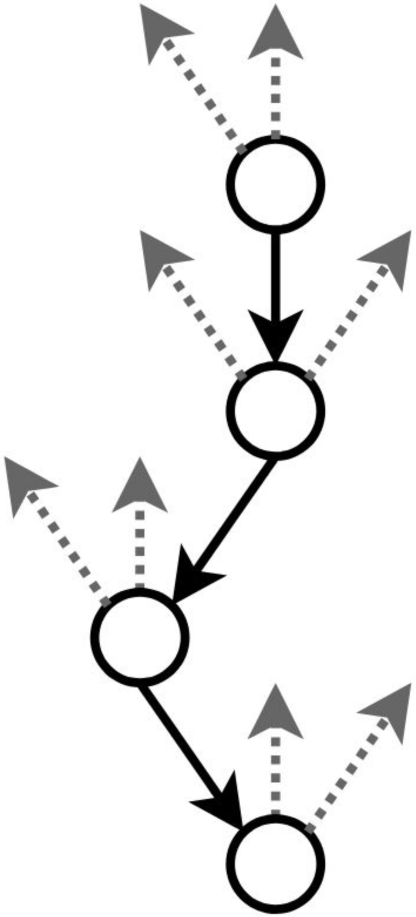
# Backpropagation

- Back up new information along the path chosen by selection:

- Win rate

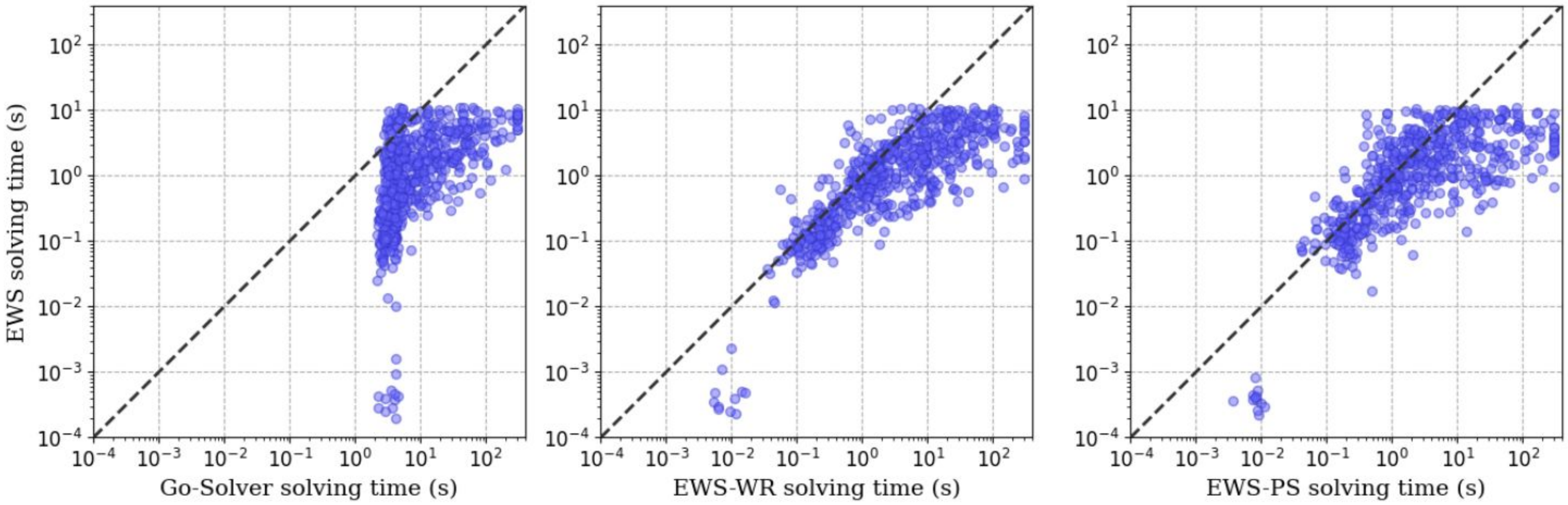
- Expected Work

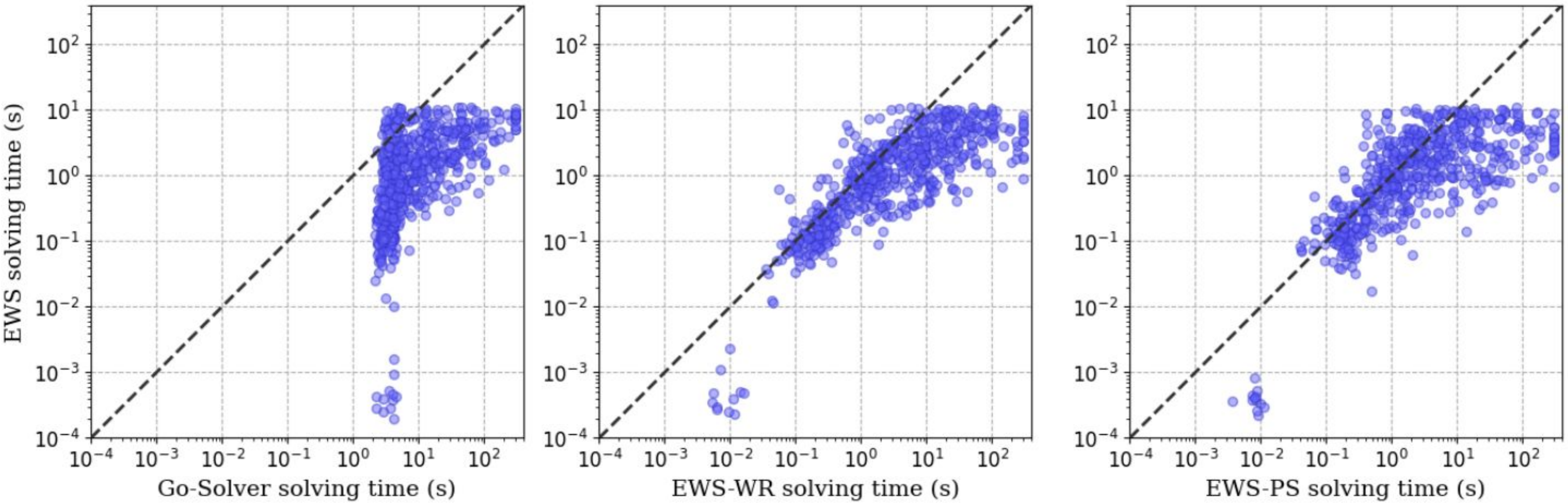
- Proofs



# Results

- Evaluated on 600 6x6 Go positions
- Compared against Go-Solver, and ablation removing win rate or proof size estimation






| Program | Av. solve time (s) | # Faster than EWS | # Timeouts |
|---|---|---|---|
| EWS | 2.32 | - | - |
| Go-Solver | 22.63 | 31 (5.2%) | 11 (1.8%) |
| EWS-WR | 19.70 | 96 (16.0%) | 11 (1.8%) |
| EWS-PS | 19.58 | 171 (28.5%) | 14 (2.3%) |

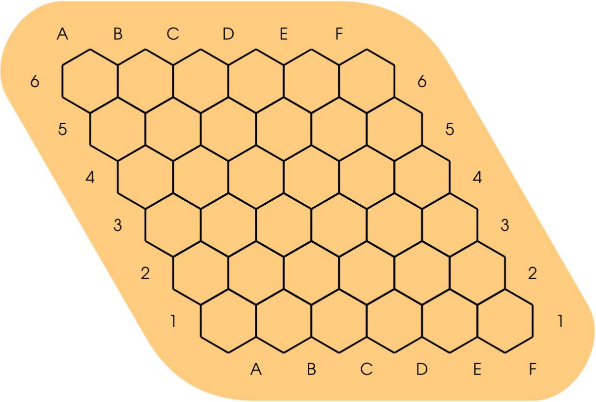
# Solving Square Go Boards

- Evaluated EWS on solving empty Go boards
- Compared against Go-Solver, MIGOS (S.O.T.A with Japanese rules), and ablation

| | $3 \times 3$ Go | | $4 \times 4$ Go | | $5 \times 5$ Go | |
|---|---|---|---|---|---|---|
| Program | Time (s) | Nodes | Time (s) | Nodes | Time (h) | Nodes |
| **EWS** | **0.053** | **161** | **13.626** | **495,494** | **5.252** | **2,605 M** |
| EWS-WR | 0.074 | 796 | 40.396 | 1,562,718 | $> 24$ | - |
| EWS-PS | 0.070 | 232 | 18.320 | 744,169 | $> 24$ | - |
| Go-Solver | 1.299 | 1,628 | 51.522 | 799,607 | $> 24$ | - |
| MIGOS SSK | $< 3.3$ | $\sim$ 25,118 | $\sim$3,960 | $\sim$3,162 M | - | - |

# Solving NxN Hex Boards

- Comparing against:
  - An enhanced alpha beta solver we developed
  - Morat Monte Carlo Tree Search
  - Morat Proof Number Search




| | $4 \times 4$ Hex | | $5 \times 5$ Hex | | $6 \times 6$ Hex | |
|---|---|---|---|---|---|---|
| Program | Time (s) | Nodes | Time (s) | Nodes | Time (h) | Nodes |
| **EWS** | **0.002** | **283** | **0.253** | **37,034** | **0.422** | **93,963,192** |
| Enhanced AB | 0.004 | 1,673 | 3.935 | 698,402 | $> 24$ | - |
| Morat MCTS | 0.035 | 4,644 | 29.669 | 3,554,546 | $> 24$ | - |
| Morat PNS | 0.054 | 8,871 | 758.102 | 154,539,591 | $> 24$ | - |

# Thank you for your time!

# Questions?

Contact me at: davidowe@ualberta.ca

# References


[1] Allis, L., Herik, H., and Huntjens, M. Go-Moku and Threat-Space Search. *Computational Intelligence 12* (1994), pp. 7–23.

[2] Allis, L., van der Meulen, M., and van den Herik, H. Proof-Number Search. *Artificial Intelligence 66*, 1 (1994), pp. 91–124.

[3] Anshelevich, V. V. The Game of Hex: An Automatic Theorem Proving Approach to Game Programming. In *AAAI/IAAI 17* (2000), pp. 189–294.

[4] Anshelevich, V. V. A Hierarchical Approach to Computer Hex. *Artificial Intelligence 134*, 1 (2002), pp. 101–120.

[5] Arneson, B., Hayward, R., and Henderson, P. Solving Hex: Beyond Humans. In *Computers and Games* (2010), pp. 1–10.

[6] Benson, D. Life in the Game of Go. *Information Sciences 10* (1976), pp. 17–29.

[7] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games 4*, 1 (2012), pp. 1–43.

[8] Campbell, M. The Graph-History Interaction: On Ignoring Position History. In *ACM Annual Conference on the Range of Computing* (1985), pp. 278–280.

[9] Doe, E., Winands, M. H. M., Kowalski, J., Soemers, D. J. N. J., Górski, D., and Browne, C. Proof Number Based Monte-Carlo Tree Search. In *IEEE Conference on Games* (2022), pp. 206–212.

[10] Du, H., Wei, T. H., and Müller, M. Solving NoGo on Small Rectangular Boards. In *Advances in Computer Games* (2024), Springer Nature Switzerland, pp. 39–49.

[11] Enzenberger, M., Müller, M., Arneson, B., and Segal, R. Fuego—An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search. *IEEE Transactions on Computational Intelligence and AI in Games 2* (2011), pp. 259–270.

[12] Ewalds, T. V. Playing and Solving Havannah, 2012. *University of Alberta Master's Thesis* (2012), https://github.com/tewalds/morat.

[13] Hayward, R., Björnsson, Y., Johanson, M., Kan, M., Po, N., and van Rijswijck, J. Solving 7×7 Hex: Virtual Connections and Game-State Reduction. *Advances in Computer Games: Many Games, Many Challenges* (2004), pp. 261–278.

[14] Henderson, P., Arneson, B., and Hayward, R. Solving 8x8 Hex. *IJCAI International Joint Conference on Artificial Intelligence* (2009), pp. 505–510.

[15] Kishimoto, A., and Müller, M. A General Solution to the Graph History Interaction Problem. In *Proceedings of the 19th National Conference on Artifical Intelligence* (2004), pp. 644–649.

[16] Kishimoto, A., Winands, M., Müller, M., and Saito, J. T. Game-Tree Search Using Proof Numbers: The First Twenty Years. *ICGA Journal 35* (2012), pp. 131–156.

[17] Knuth, D. E., and Moore, R. W. An Analysis of Alpha-Beta Pruning. *Artificial Intelligence 6*, 4 (1975), pp. 293–326.

[18] Kocsis, L., and Szepesvári, C. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML* (2006), pp. 282–293.

[19] Müller, M. Playing it Safe: Recognizing Secure Territories in Computer Go by Using Static Rules and Search. In *Game Programming Workshop* (1997), pp. 80–86.

[20] Nagai, A. DF-PN Algorithm for Searching AND/OR Trees and its Applications. *University of Tokyo Ph.D. Thesis* (2002).

[21] Neto, J. P., and Taylor, W. Game Mutators for Restricting Play. *Game & Puzzle Design* (2015), pp. 64–65.

[22] Niu, X., Kishimoto, A., and Müller, M. Recognizing Seki in Computer Go. In *Advances in Computer Games. 11th International Conference* (2006), vol. 4250 of *Lecture Notes in Computer Science*, Springer, pp. 88–103.

[23] Niu, X., and Müller, M. An Improved Safety Solver for Computer Go. In *Computers and Games: 4th International Conference* (2006), vol. 3846 of *Lecture Notes in Computer Science*, Springer, pp. 97–112.

[24] Niu, X., and Müller, M. An Open Boundary Safety-of-Territory Solver for the game of Go. In *Computers and Games. 5th International Conference* (2007), vol. 4630 of *Lecture Notes in Computer Science*, Springer, pp. 37–49.

[25] Niu, X., and Müller, M. An Improved Safety Solver in Go using Partial Regions. In *Computers and Games. 6th International Conference* (2008), vol. 5131 of *Lecture Notes in Computer Science*, Springer, pp. 102–112.

[26] Pawlewicz, J., and Hayward, R. B. Scalable Parallel DFPN Search. In *Computers and Games* (2013), pp. 138–150.

[27] Pawlewicz, J., and Lew, L. Improving Depth-First PN-Search: 1 + Epsilon Trick. In *Proceedings of the 5th International Conference on Computers and Games* (2006), pp. 160–171.

[28] Randall, O., Wei, T.-H., Hayward, R., and Müller, M. Improving Search in Go Using Bounded Static Safety. In *Computers and Games* (2022), pp. 14–23.

[29] Rosin, C. Multi-armed Bandits with Episode Context. *Annals of Mathematics and Artificial Intelligence 61* (2010), pp. 203–230.

[30] Saffidine, A., and Cazenave, T. Developments on Product Propagation. In *Computers and Games* (2013), pp. 100–109.

[31] Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S. Checkers is Solved. *Science 317*, 5844 (2007), pp. 1518–1522.

[32] Schaeffer, J., and Plaat, A. New Advances in Alpha-Beta Searching. In *Proceedings of the 1996 ACM 24th annual Conference on Computer Science* (1996), pp. 124–130.

[33] Shih, C.-C., Wu, T.-R., Wei, T. H., and Wu, I.-C. A Novel Approach to Solving Goal-Achieving Problems for Board Games. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence* (2021), pp. 10362–10369.

[34] Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature 529* (2016), pp. 484–489.

[35] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-Play. *Science 362*, 6419 (2018), pp. 1140–1144.

[36] Slate, D., and Atkin, L. Chess Skill in Man and Machine. In *Springer-Verlag* (1977), pp. 82–118.

[37] Song, J., and Müller, M. An Enhanced Solver for the Game of Amazons. *IEEE Transactions on Computational Intelligence and AI in Games 7* (2015), pp. 16–27.

[38] Sutton, R. S., and Barto, A. G. Reinforcement Learning: An Introduction. *MIT Press* (1998), pp. 39-40.

[39] van der Werf, E., Herik, H., and Uiterwijk, J. Solving Go on Small Boards. *ICGA Journal 26* (2003), pp. 92–107.

[40] van der Werf, E., and Winands, M. Solving Go for Rectangular Boards. *ICGA Journal 32*, 2 (2009), pp. 77–88.

[41] Wagner, J., and Virág, I. Solving Renju. *Joint International Computer Games Association 24* (2001), pp. 30–35.

[42] Winands, M. H. M., Björnsson, Y., and Saito, J.-T. Monte-Carlo Tree Search Solver. In *Computers and Games* (2008), pp. 25–36.

[43] Wu, T.-R., Shih, C.-C., Wei, T.-H., Tsai, M.-Y., Hsu, W.-Y., and Wu, I.-C. AlphaZero-based Proof Cost Network to Aid Game Solving. In *International Conference on Learning Representations* (2022).

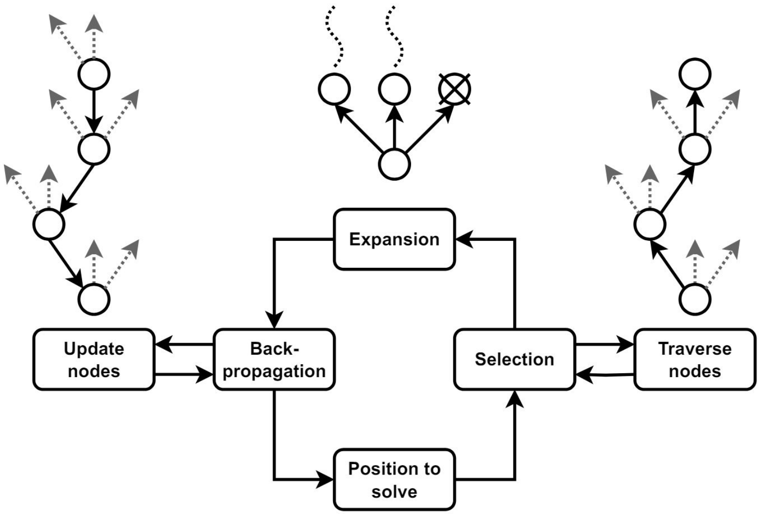
[44] Zobrist, A. A New Hashing Method with Application for Game Playing. *ICGA Journal 13* (1990), pp. 69–73.

# Conclusion

- Expected Work Search combines proof size and win rate estimation
    - Balances these heuristics
    - Covers previous weaknesses

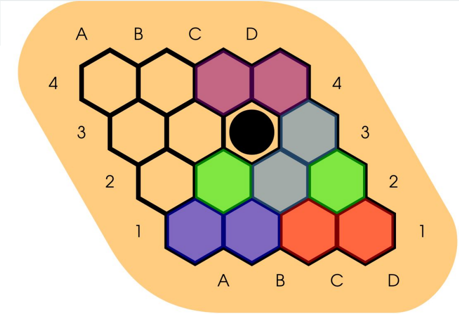
- Strong results on Go and Hex
    - Orders of magnitude faster than tested programs
    - New results solving 5x5 Go

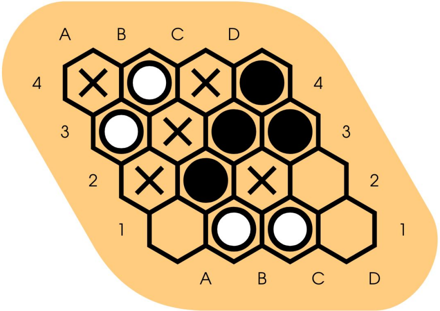
- Future work
    - Parallelize
    - Use EW as a problem complexity estimator
    - Evaluate on more problems



# Solving with Hex Knowledge

- Significantly reduces search space
- 8x8 can be solved
- Search is much slower, but requires less nodes
- More Hex knowledge is required for S.O.T.A.

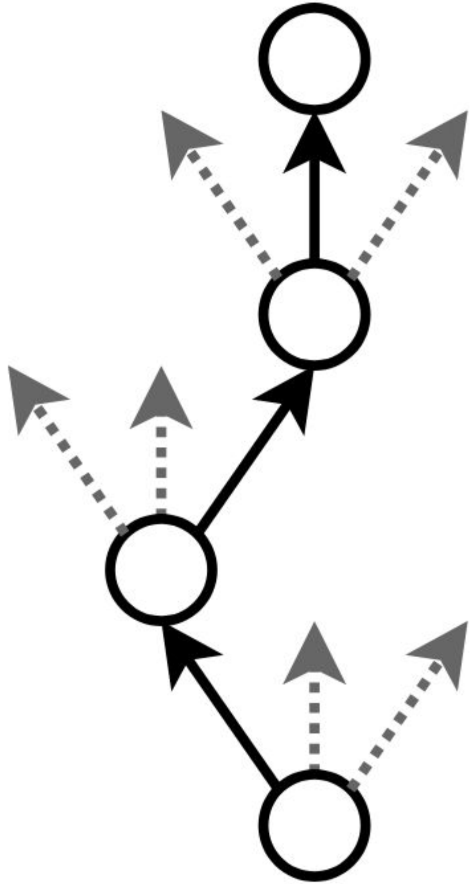



| | $6 \times 6$ | | $7 \times 7$ | | $8 \times 8$ | |
|---|---|---|---|---|---|---|
| Program | 6×6 Time (s) | Nodes | Time (s) | Nodes | Time (s) | Nodes |
| EWS | 1,519 | 93,963,192 | - | - | - | - |
| **EWS with knowledge** | **0.005** | **26** | **0.588** | **2,318** | **234.449** | **555,158** |

# Selection

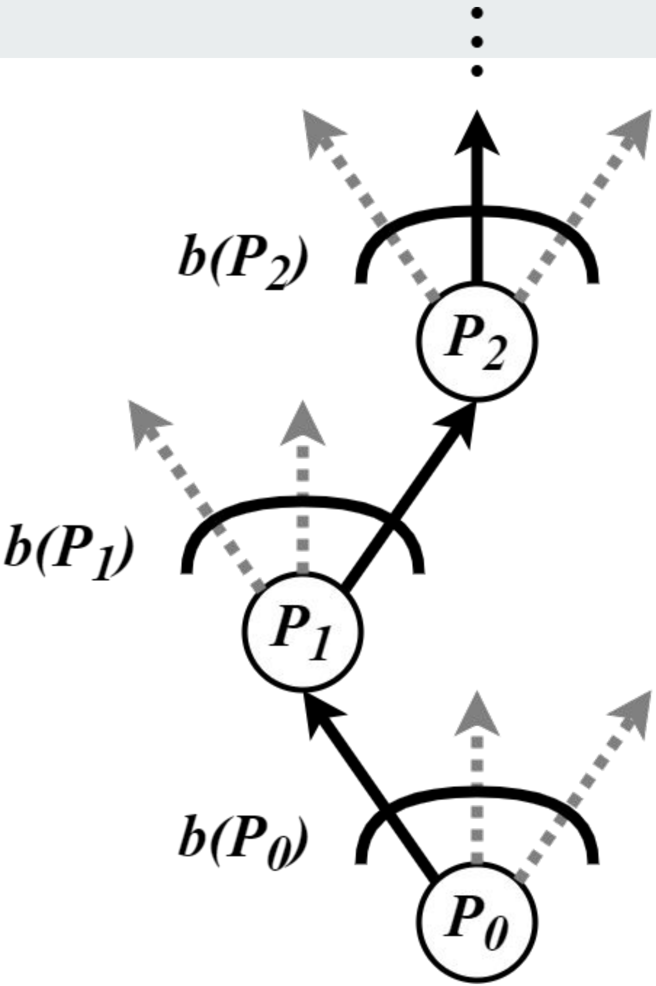
- Traverse the search tree until a leaf node is found
- Select the best child according to the move ordering
- If the selected child has been expanded, continue selection
- Otherwise, expand the leaf node



# Initial Evaluation

- Random simulations
- Win rates = wins / visits
- Initialize EW as the sum of the branching factors of positions in random simulations

$$EW_0(X) := \sum_{i=0}^{m} b(P_i)$$



# Results

- Compared against different independent solving implementations
  - GoSolver, MIGOS, Enhanced AB, Morat PNS, Morat MCTS
- Ablation study
  - Removed proof-size / win-rate information
  - Performance drops
- Evaluated a variety of Go positions
  - Strong general performance
  - New results on empty boards
- Evaluated empty nxn Hex
  - Strong results with and without knowledge

# Future Work

- Improved implementation
- Additional games
- Parallelize
- EWS Estimation
- Further solving
- Further evaluation

| Objective | Timeline |
|---|---|
| Refactor to improve code<br>Rectangular board solving<br>$>$64 move game solving | May - June 2024 |
| Solving NoGo<br>Solving Amazons<br>Solving Gomoku<br>Paper on new results | June - September 2024 |
| EWS in parallel workers<br>EWS in manager program | September - November 2024 |
| Estimating game complexity<br>Paper on EWS estimation | November - February 2025 |
| Solving 9x9, 10x10 Hex<br>Solving 6x6 Go<br>Solving larger NoGo/Amazons/Gomoku<br>Paper on new results | February - May 2025 |
| Thorough EWS settings evaluation | May - June 2025 |
| Finish writing thesis | June - September 2025 |

# Improved Implementation

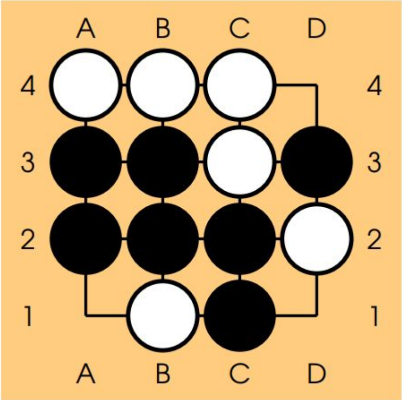
- Refactor
    - Improve software quality
- Remove artificial limitations
    - 64-bit data types limit games with >64 moves
    - Go and Hex implementations do not support rectangular boards
- One month

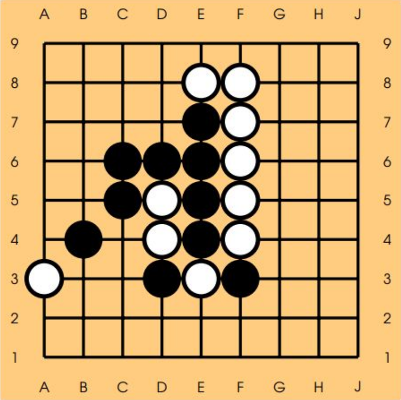
| Objective | Timeline |
|---|---|
| Refactor to improve code<br>Rectangular board solving<br>$>$64 move game solving | May - June 2024 |

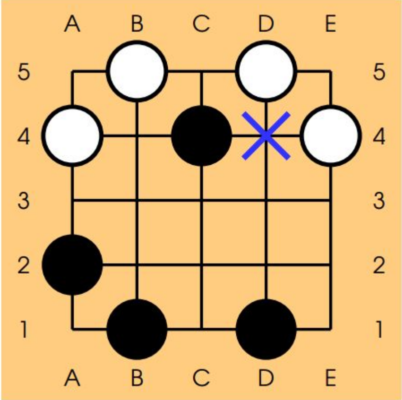
# Additional Games

- NoGo
  - Compare against new SBH results
- Amazons
  - Integrate existing endgame databases
- Ninuki
  - Gomoku variant
  - Capturing
  - More complicated
- Three months

| | |
|---|---|
| Solving NoGo | June - September 2024 |
| Solving Amazons | |
| Solving Gomoku | |
| Paper on new results | |



# Parallelization

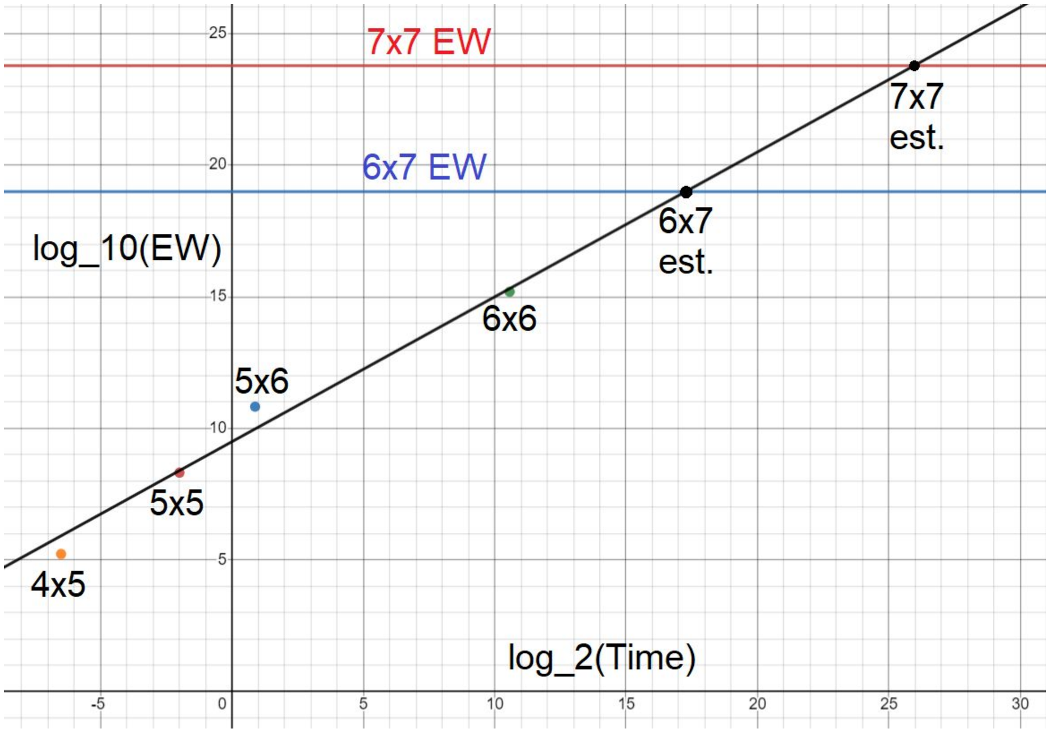
| | |
|---|---|
| EWS in parallel workers<br>EWS in manager program | September - November 2024 |

- Currently running single threaded
- Integrate EWS in existing manager-worker program
- Use EWS implementation as the worker program
- Change manager to use EW and WR info for job selection
- Two months

# EWS Estimation

| | |
|---|---|
| Estimating game complexity<br>Paper on EWS estimation | November - February 2025 |

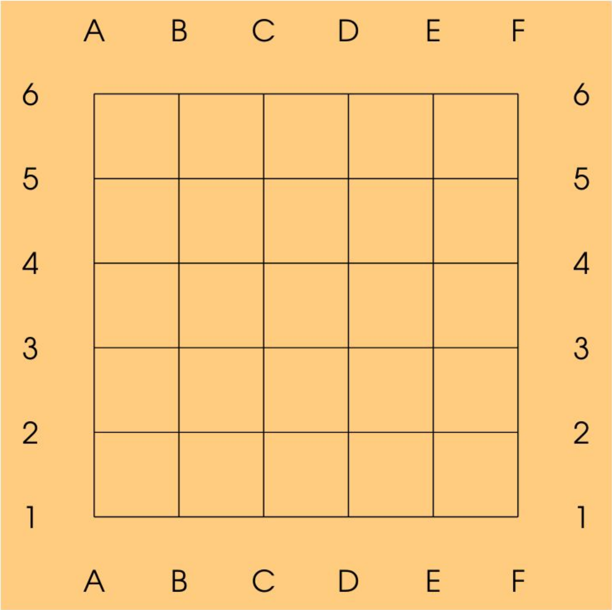
- Use EW for solving complexity estimation
- Collect a dataset of solved positions
- Use regression
- May need to tweak the EWS implementation / transform the data
- Three months



# Further Solving

| | |
|---|---|
| Solving 9x9, 10x10 Hex<br>Solving 6x6 Go<br>Solving larger NoGo/Amazons/Gomoku<br>Paper on new results | February - May 2025 |

- Utilize new parallelization and estimation
- Solve larger problems
  - 6x6 Go
  - 9x9, 10x10 Hex
  - Other large rectangular boards
  - 6x5 Amazons
  - 6x6, 7x7 NoGo
- Three months



| | |
|---|---|
| Thorough EWS settings evaluation | May - June 2025 |
| Finish writing thesis | June - September 2025 |

# Empirical Evaluation

- Hyperparameter sweep
- Test alternate EWS settings
    - EW initialization
    - Move ordering
    - Asymmetric algorithm
    - Proof Cost Network
- One month

$$EW_{loss}(A)/(1-WR(A)) < EW_{loss}(B)/(1-WR(B))$$

$$\Longleftrightarrow$$

$$A < B$$

$$\downarrow$$

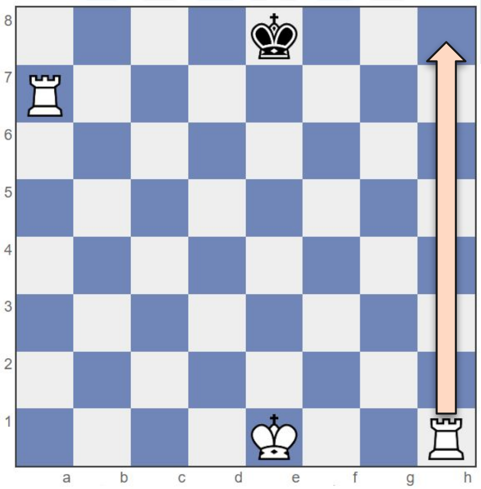
$$EW_{loss}(A) \cdot WR(A) < EW_{loss}(B) \cdot WR(B)$$

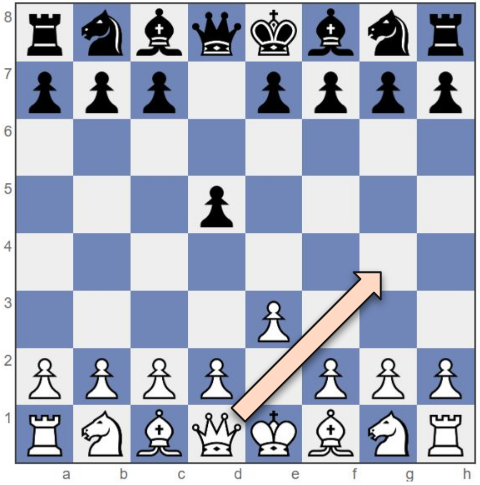
$$\Longleftrightarrow$$

$$A < B$$

# Heuristics

- Win rate estimation
    - Monte Carlo Tree Search
    - Prioritize strong moves
- Proof size estimation
    - Proof number search
    - Prioritize reducing the solution size
- Both have strengths and weaknesses
- EWS combines both using Expected Work

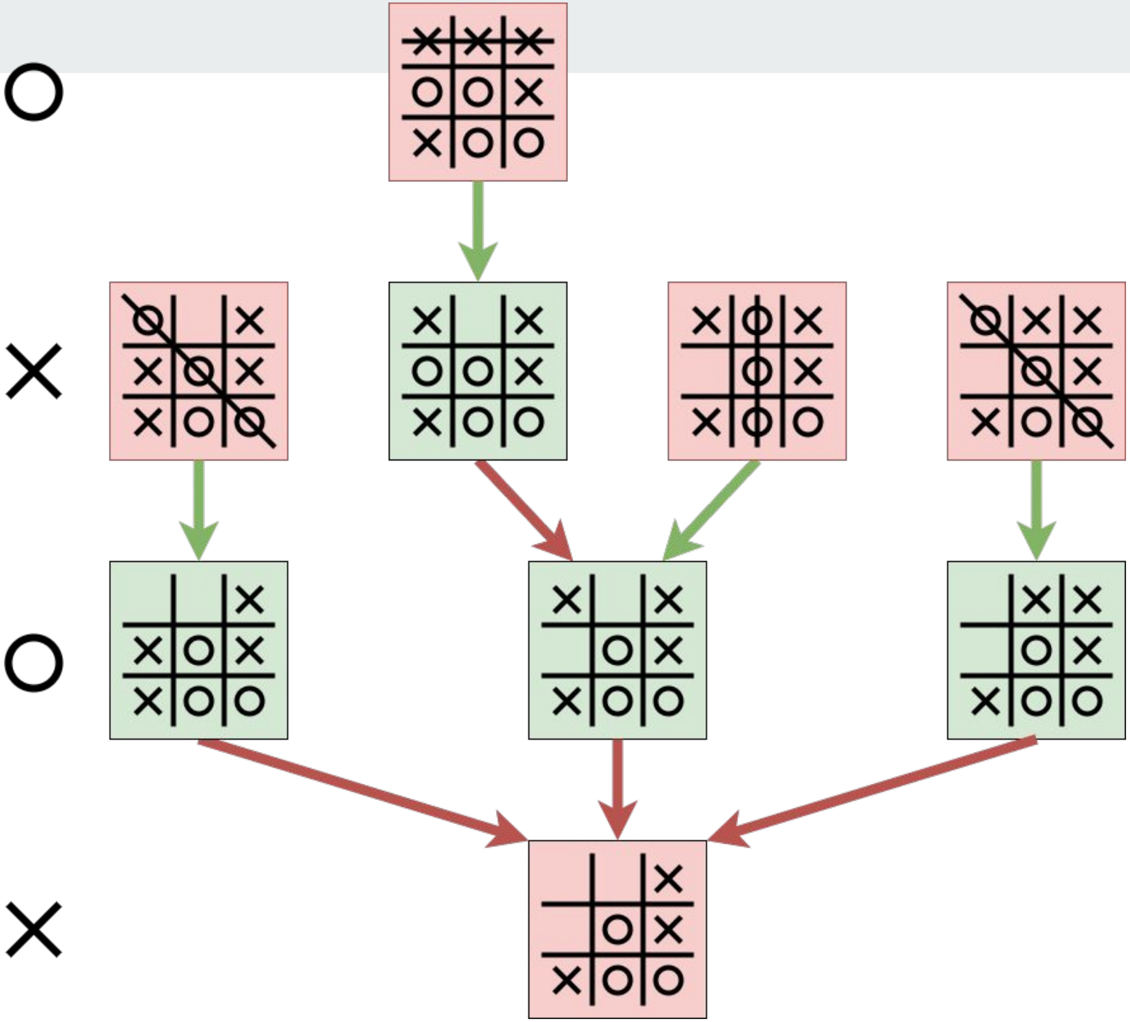



https://www.365chess.com/board_editor.php

# Solving Example

- Red = Losses
- Green = Wins
- Losing positions have all losing move
- Winning positions have at least one winning move

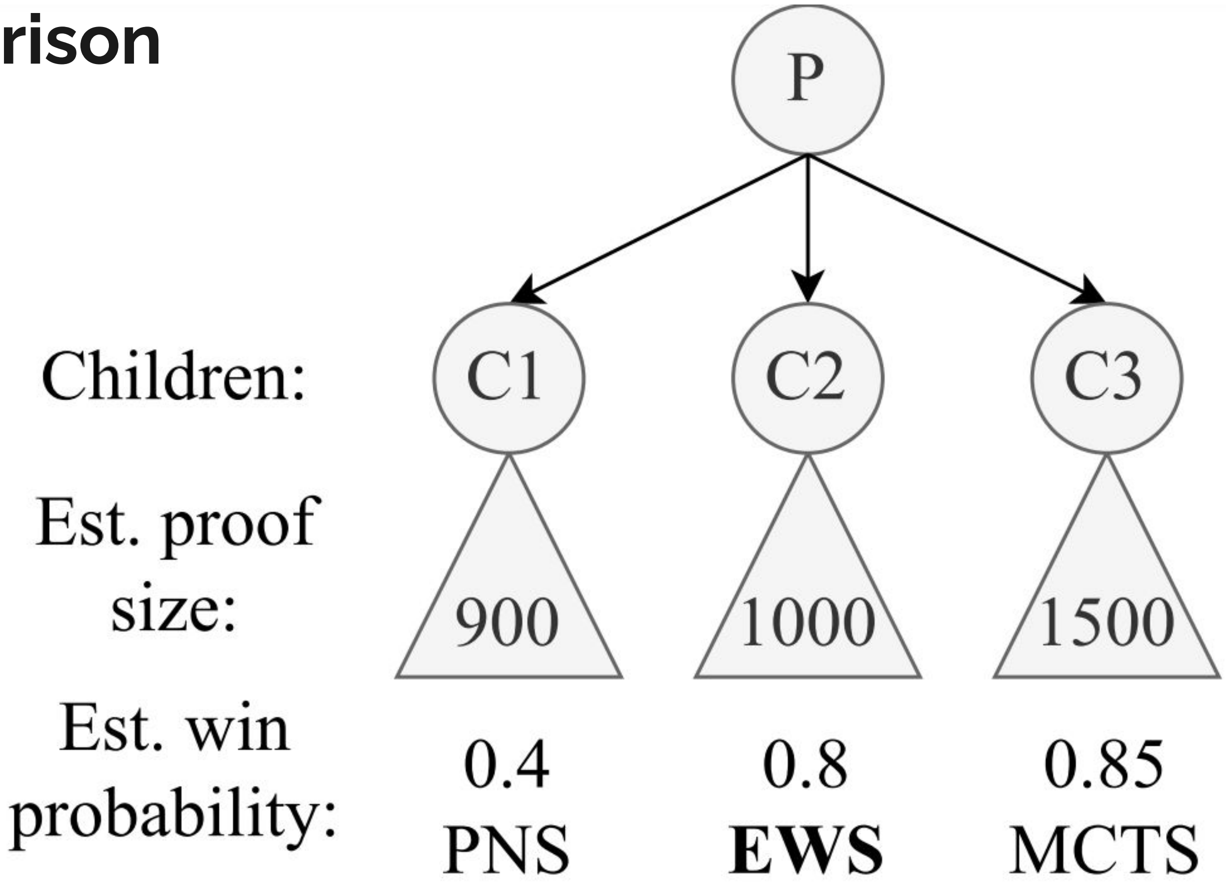


# Search comparison

- PNS will choose the smallest proof size

- MCTS will choose the highest win rate

- EWS will minimize EW
- Accounts for proof size and win rate

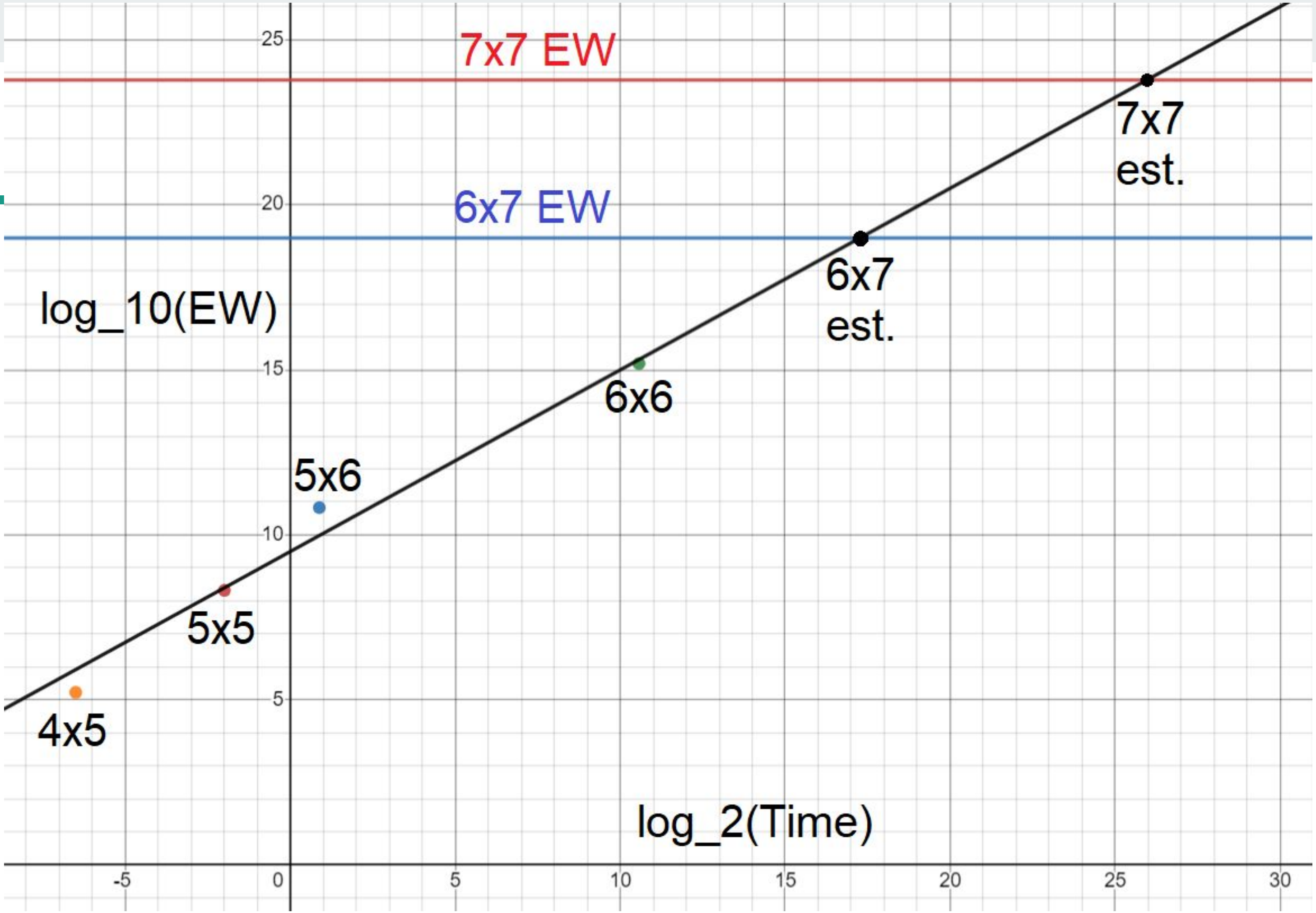


# Implementation

- All nodes are stored in an unordered table
- Table is indexed by hashes of game positions
- Zobrist hashing is used
- Table size is 2^22 nodes
- Linear probing is used to resolve hash conflicts
- Nodes are pruned from memory when solved, or if the table is too full
- Current principle variation is never pruned

# Baseline Comparison

- Weak baseline: basic negamax algorithm with transposition table
- Strong baseline: Enhanced AB algorithm with:
  - Iterative deepening
  - Alpha Beta pruning
  - A custom heuristic value function for hex
  - Transposition table
  - The history heuristic
  - Killer move heuristic
  - Relevancy zones
  - Relevancy zone pattern matching

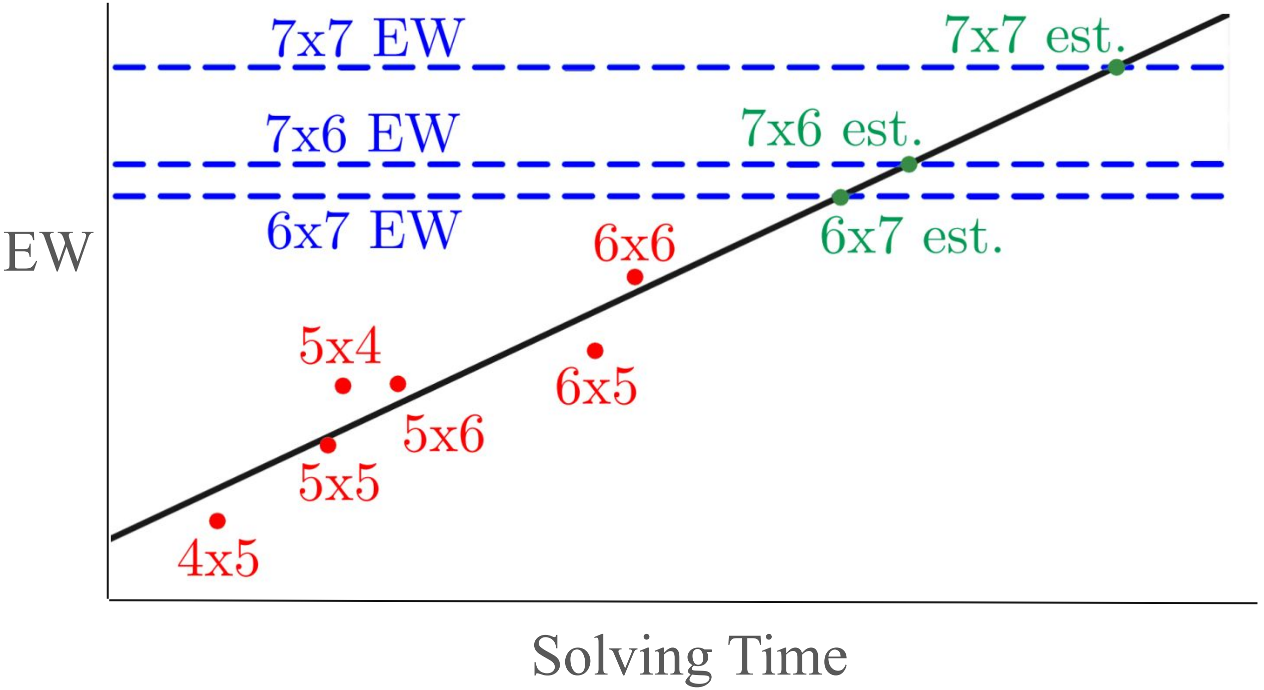


7x7 EW
6x7 EW
log_10(EW)
7x7
est.
6x7
est.
6x6
5x6
5x5
4x5
log_2(Time)
25
20
15
10
5
-5
0
5
10
15
20
25
30

# EWS Estimation

| Estimating game complexity<br>Paper on EWS estimation | November - February 2025 |
|---|---|

- Use EW for solving complexity estimation
- Collect a dataset of solved positions
- Use regression
- May need to tweak the EWS implementation / transform the data
- Three months



# Expansion

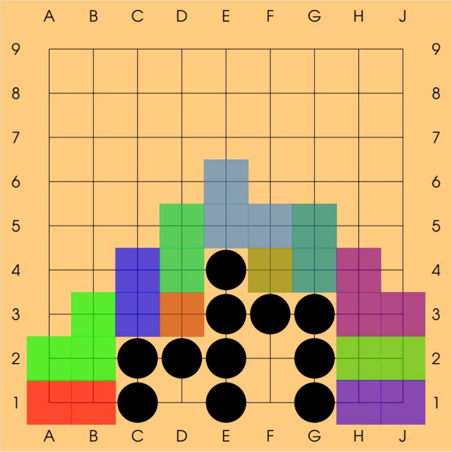
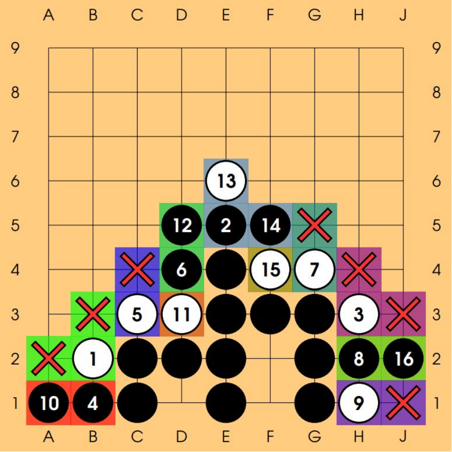
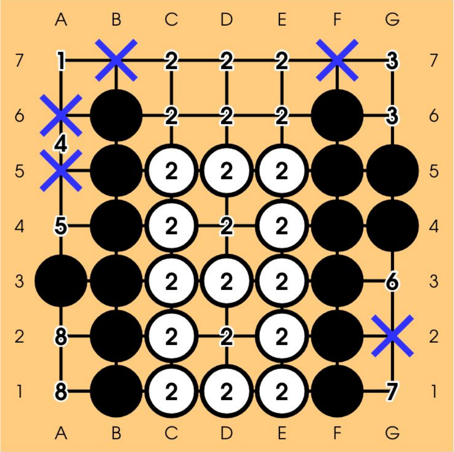
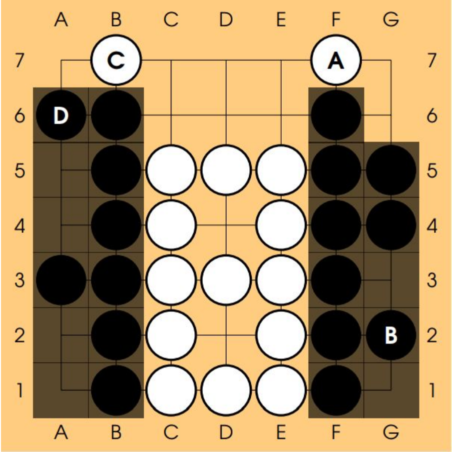
- For all legal moves:
- If the move is a terminal winning move, the node is solved as a win
- If the move is not terminal, create a new child node
- New nodes evaluate initial win rate and EW
- If the expanded node has no unsolved children left, it is solved as a loss

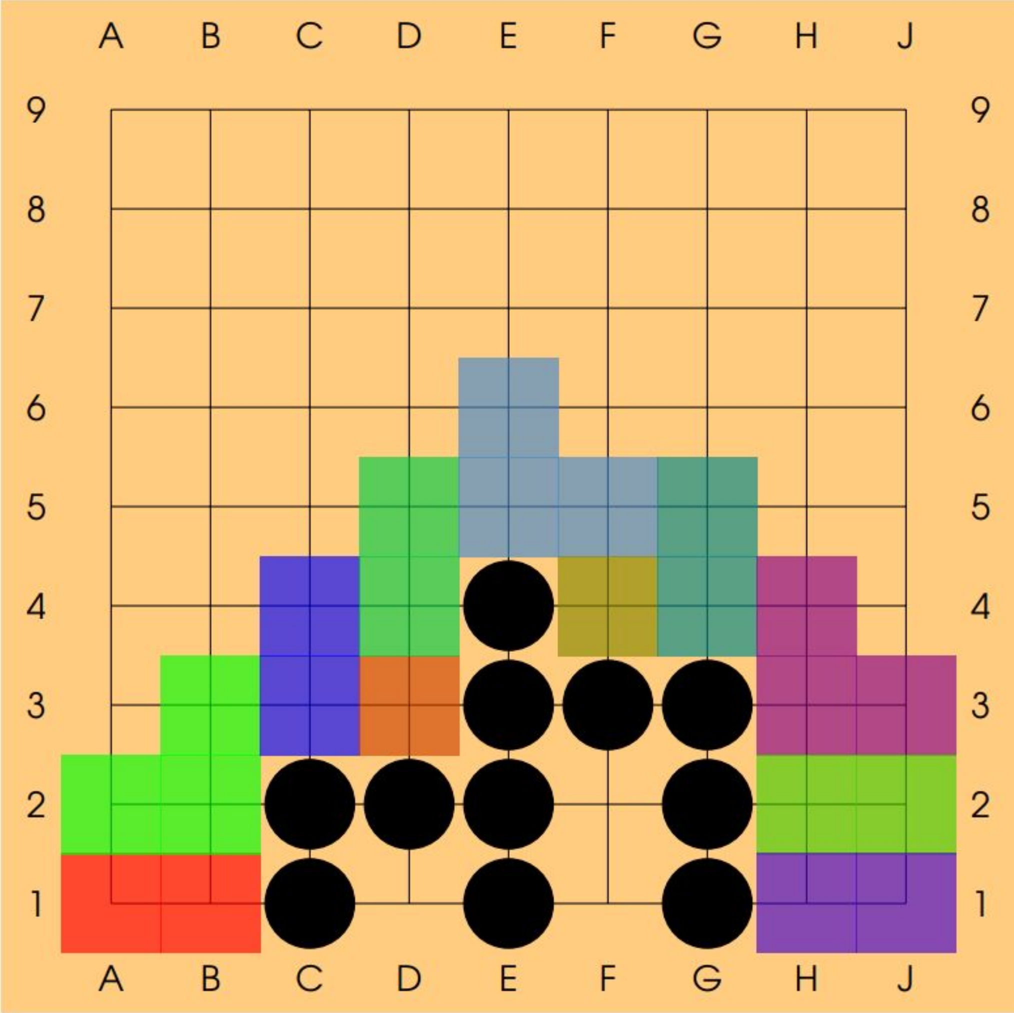
**Algorithm 2** Expand $X$: returns whether $X$ is solved and if so whether $X$ is winning

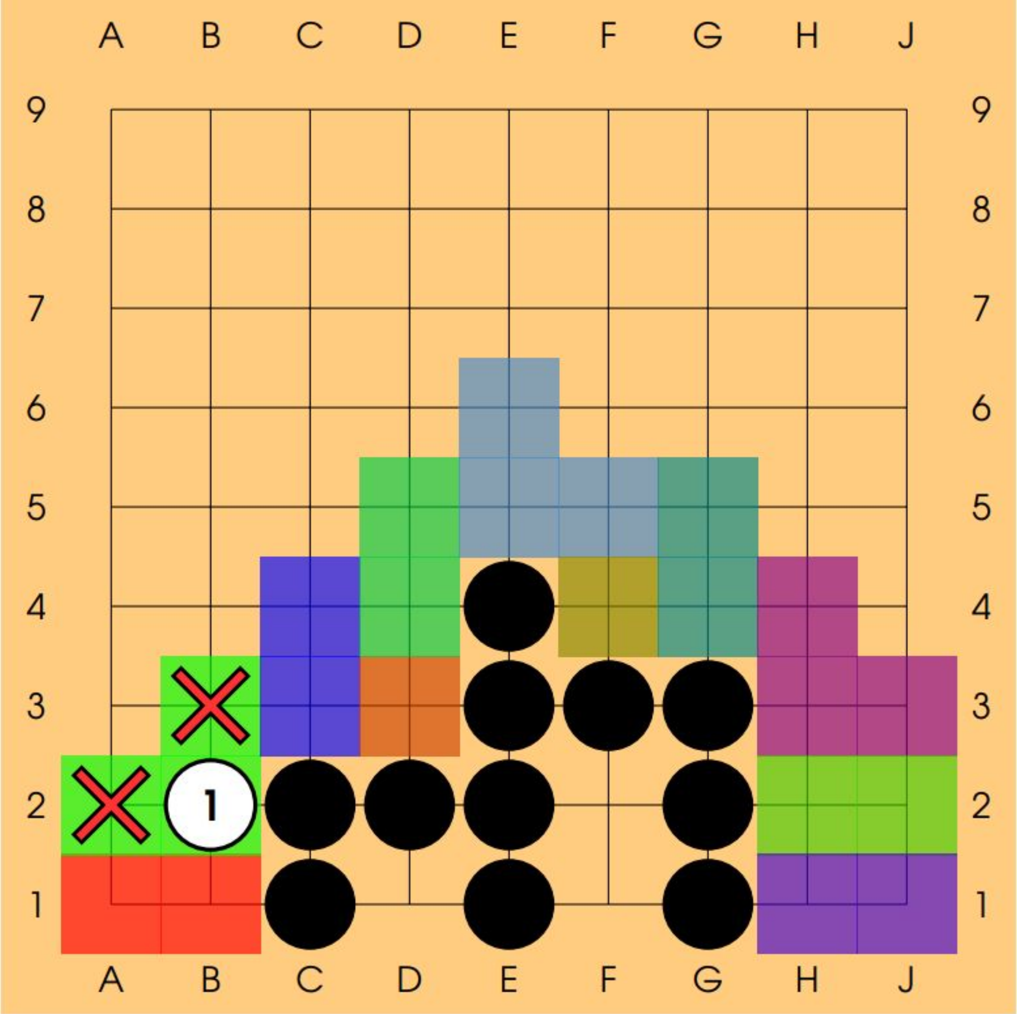
$X$.expanded := true
**for all** legal moves $m$ **do**
    **if** $m$ is a terminal winning move for $X$ **then**
        **return** true, true ▷ Solved win
    **else if** $m$ is **not** a terminal losing move for $X$ **then**
        Create node $C$
        $C$.expanded = false
        $C$.move := $m$
        Evaluate $C$.winRate
        Evaluate $C$.expectedWorkLoss
        Evaluate $C$.expectedWorkWin
        Add $C$ to $X$.children
**if** $X$.children is empty **then**
    **return** true, false ▷ Solved loss
**else**
    **return** false, false ▷ Unsolved

# Bounded Static Safety

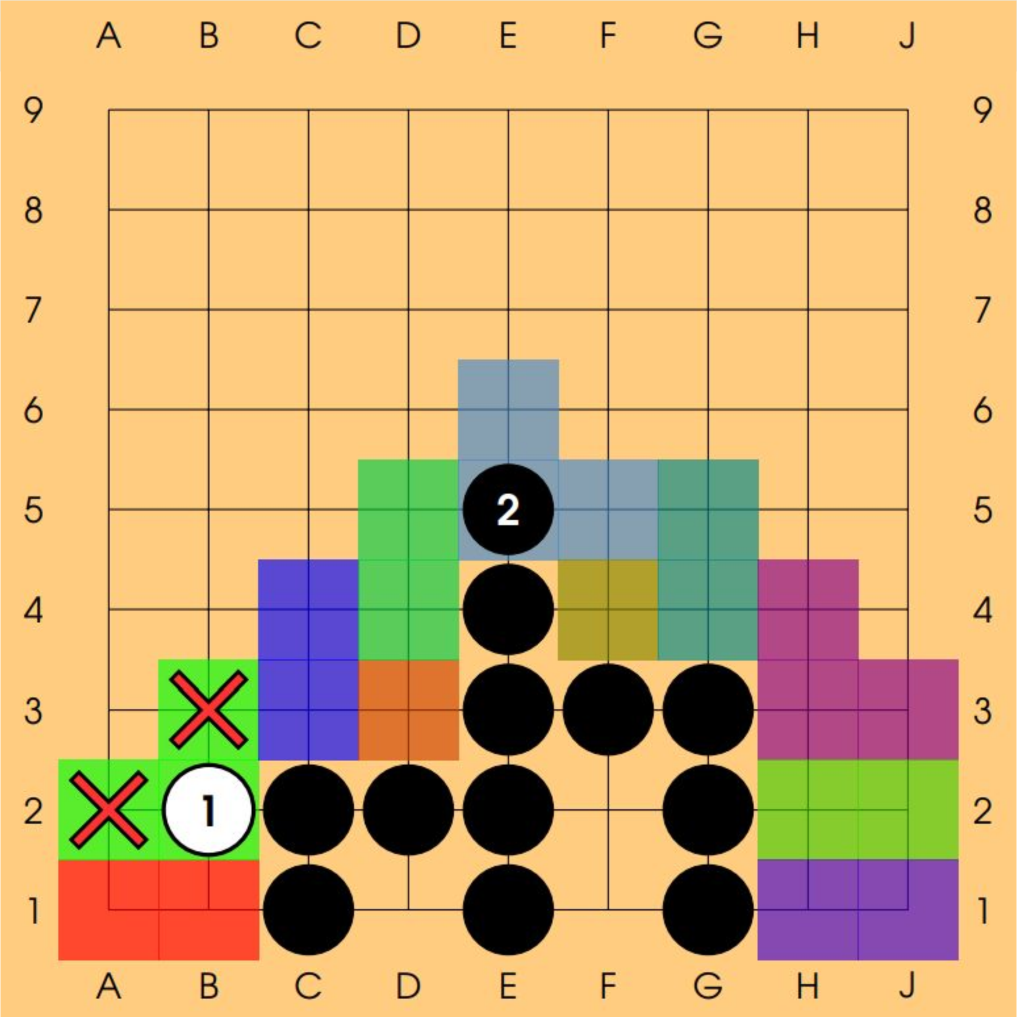
| Type | BSS | LAP | Benson | BSS +search | LAP +search | Benson +search | BSS no EL | BSS no IP | BSS no EL no IP |
|---|---|---|---|---|---|---|---|---|---|
| Av. first solve | **27.29** | 31.56 | 42.67 | **14.89** | 16.35 | 17.79 | 30.83 | 27.73 | 31.28 |

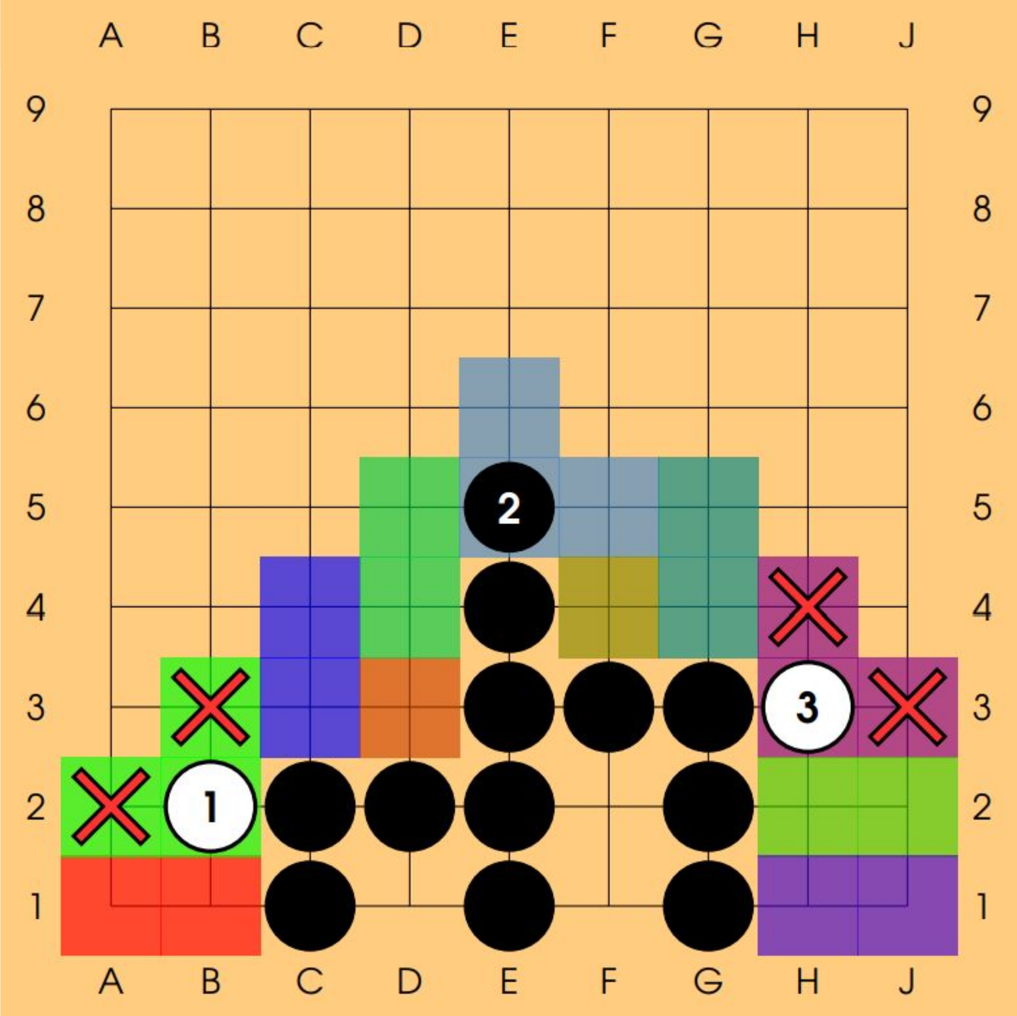






A B C D E F G H J
9 8 7 6 5 4 3 2 1

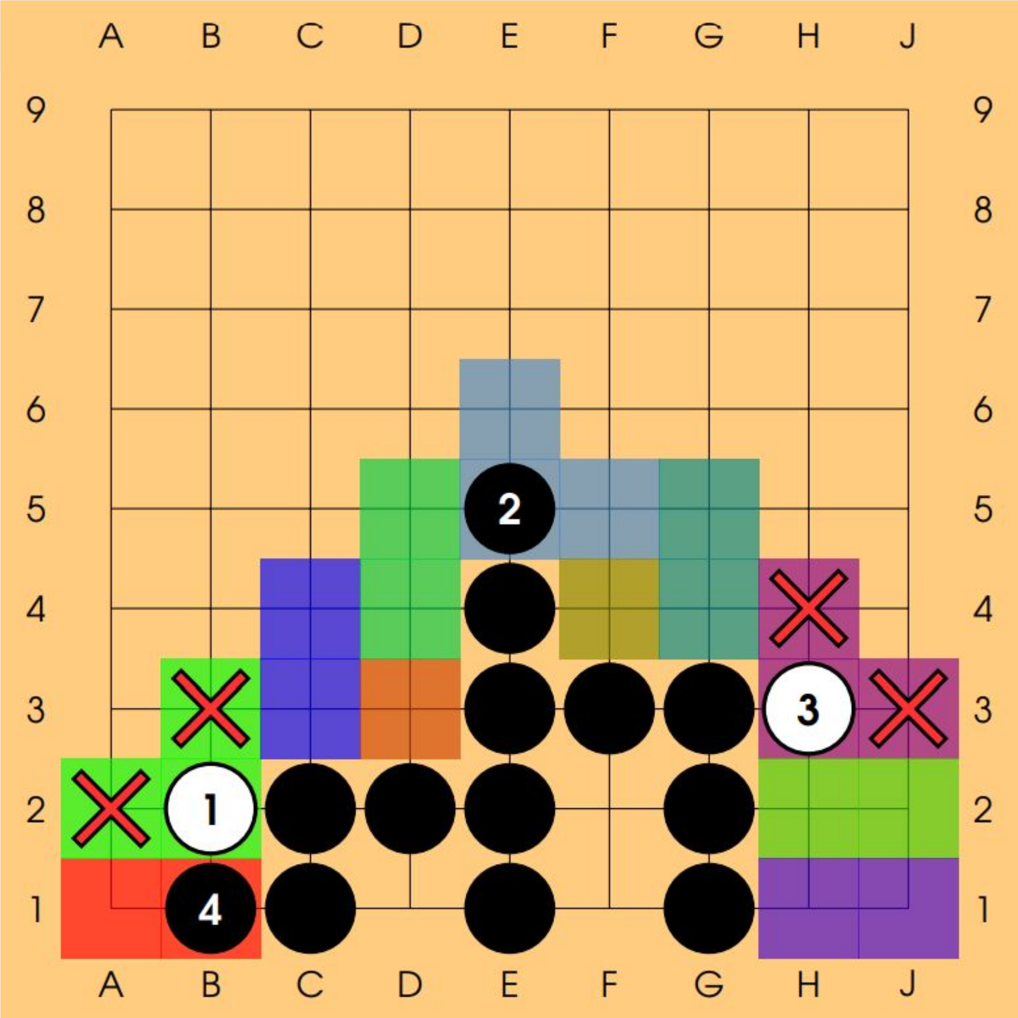


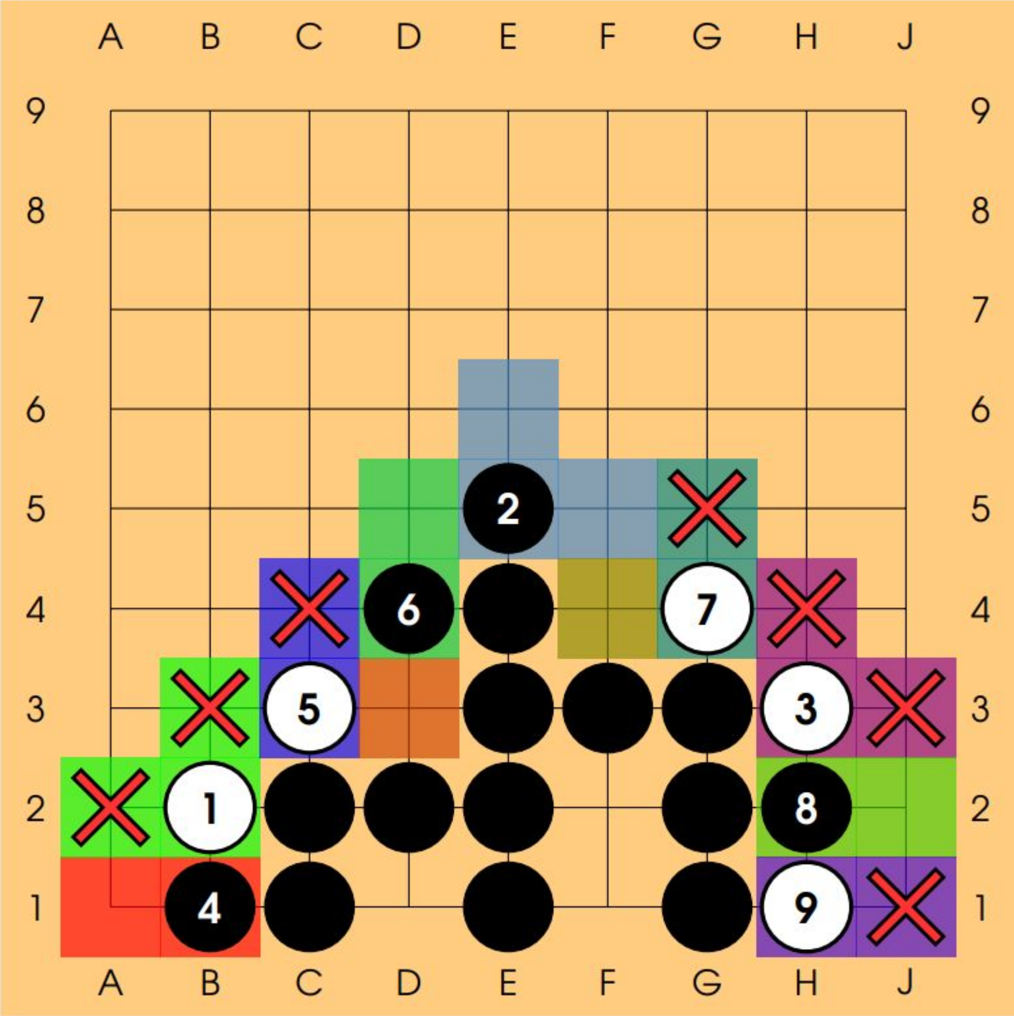
A B C D E F G H J
9 8 7 6 5 4 3 2 1
1

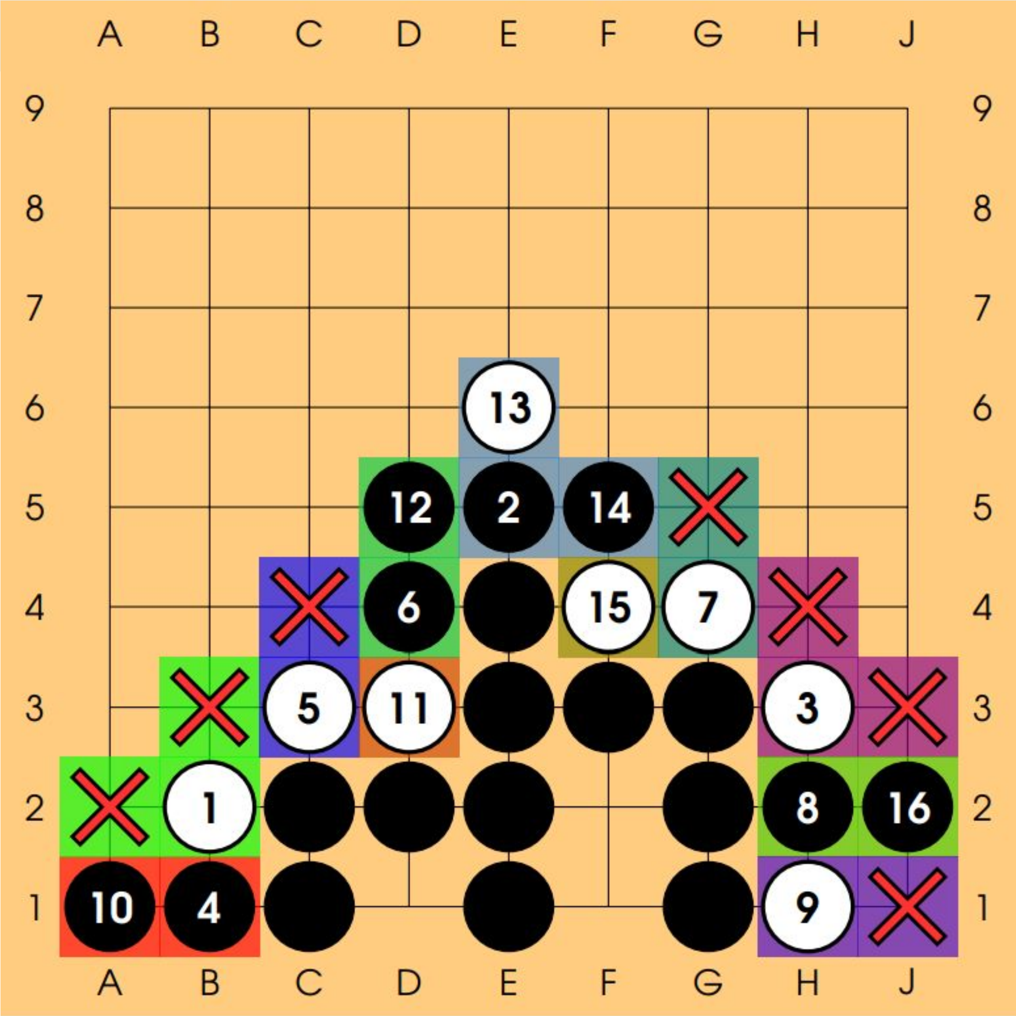

A B C D E F G H J
9 8 7 6 5 4 3 2 1
2
1



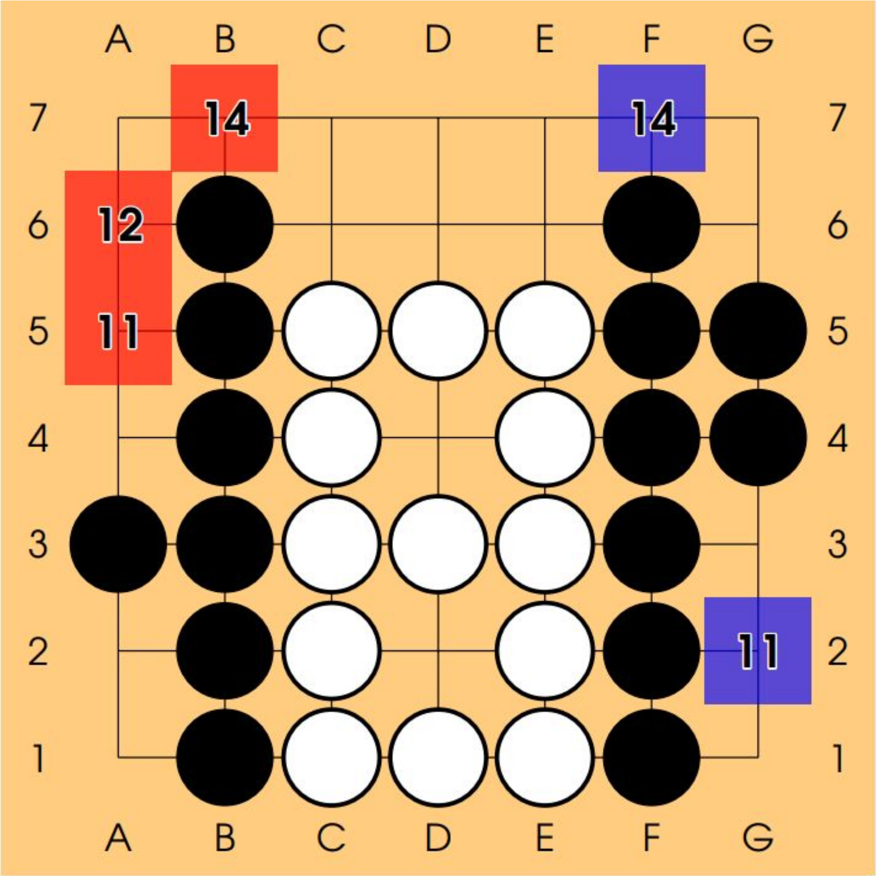
A B C D E F G H J
9 8 7 6 5 4 3 2 1
1
2
3


A B C D E F G H J
9 8 7 6 5 4 3 2 1
1 2 3 4

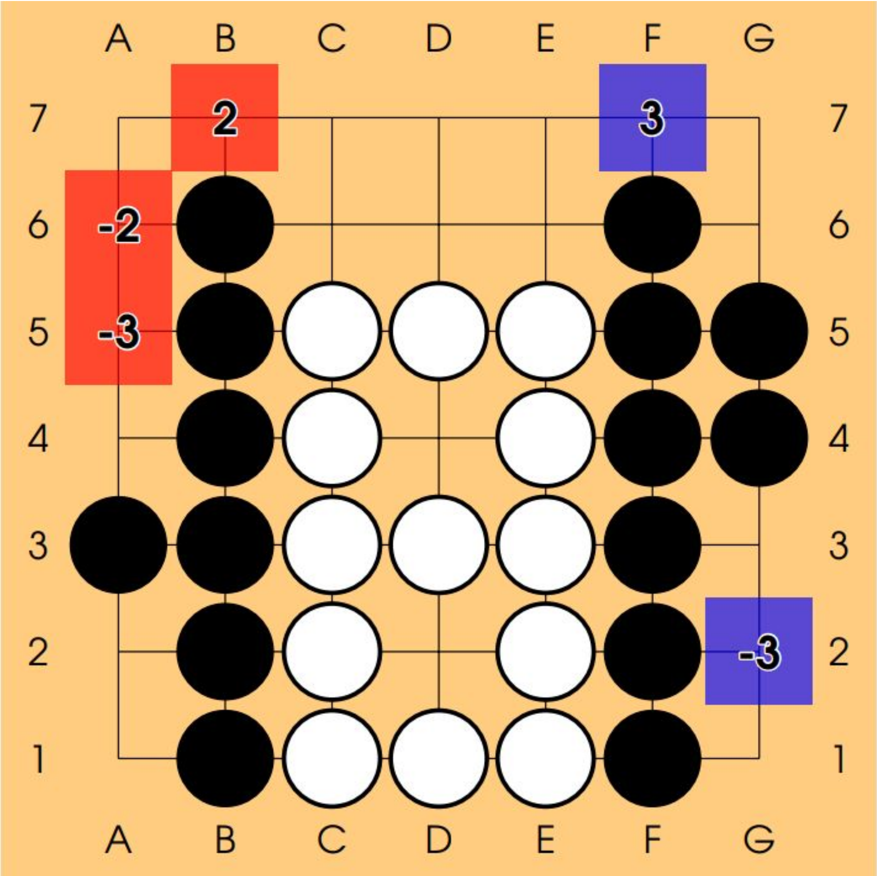

A B C D E F G H J
9 8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8 9


A B C D E F G H J
9 8 7 6 5 4 3 2 1
13
12 2 14
6 15 7
5 11 3
1 8 16
10 4 9

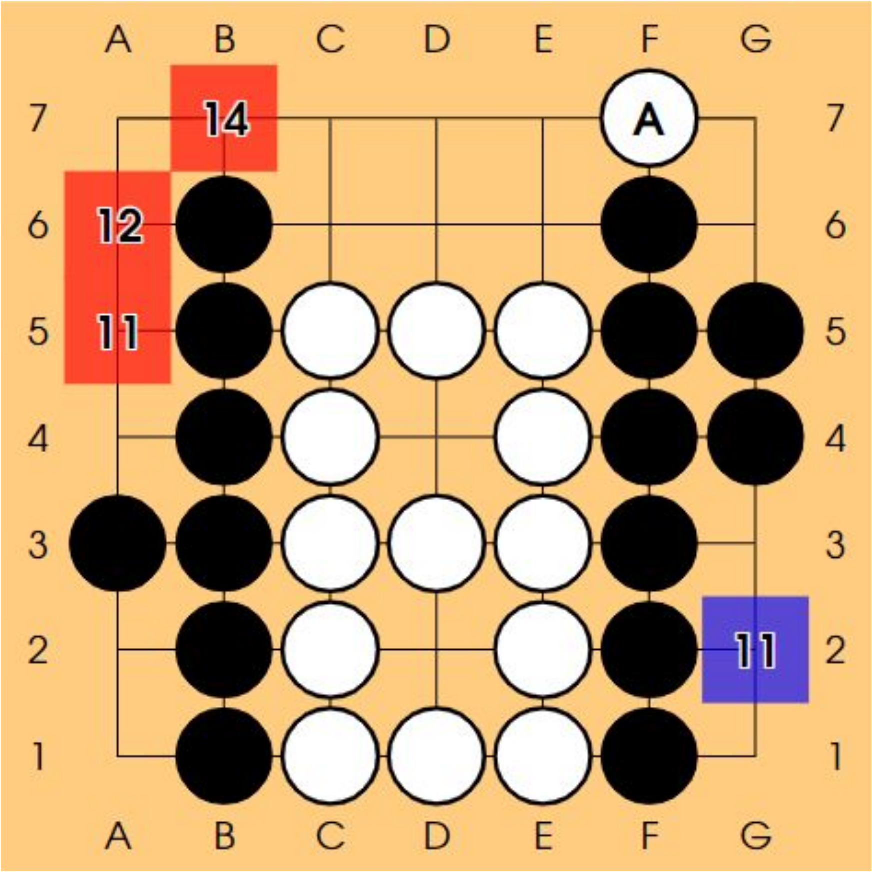
Absolute values



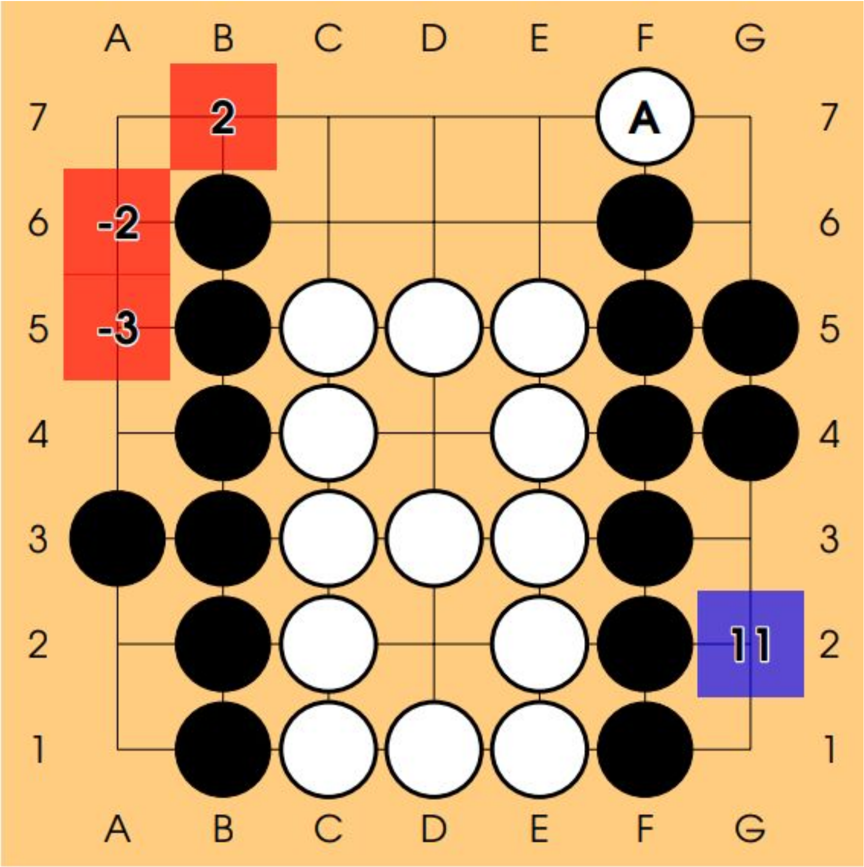

Relative values



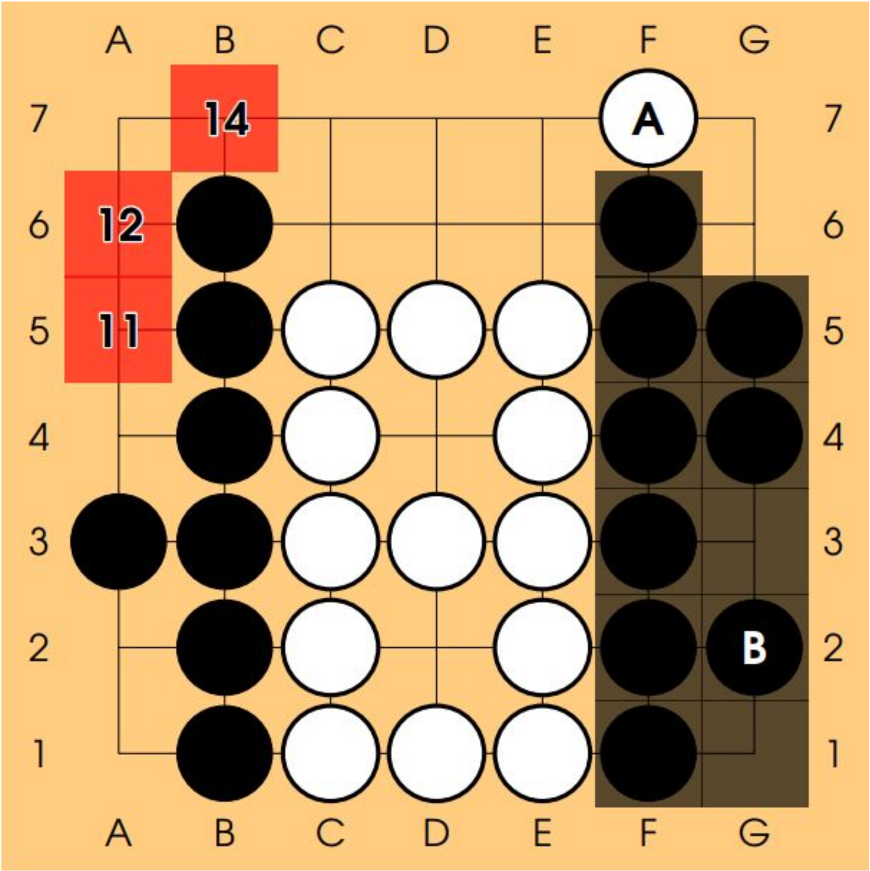
Absolute values



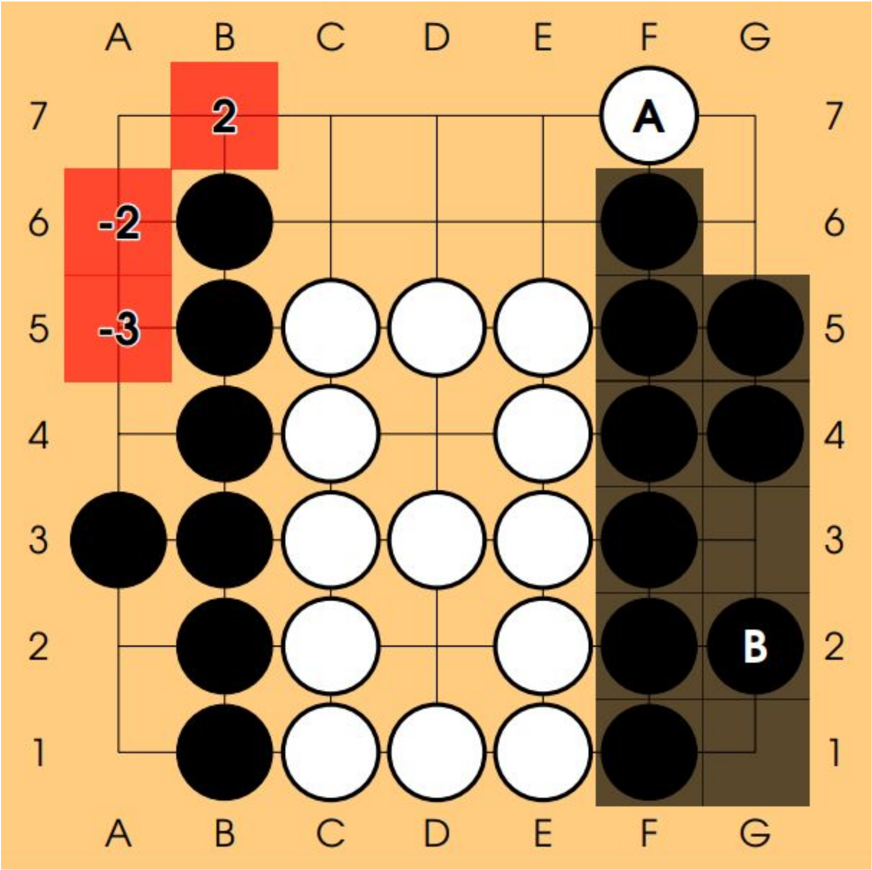

Relative values



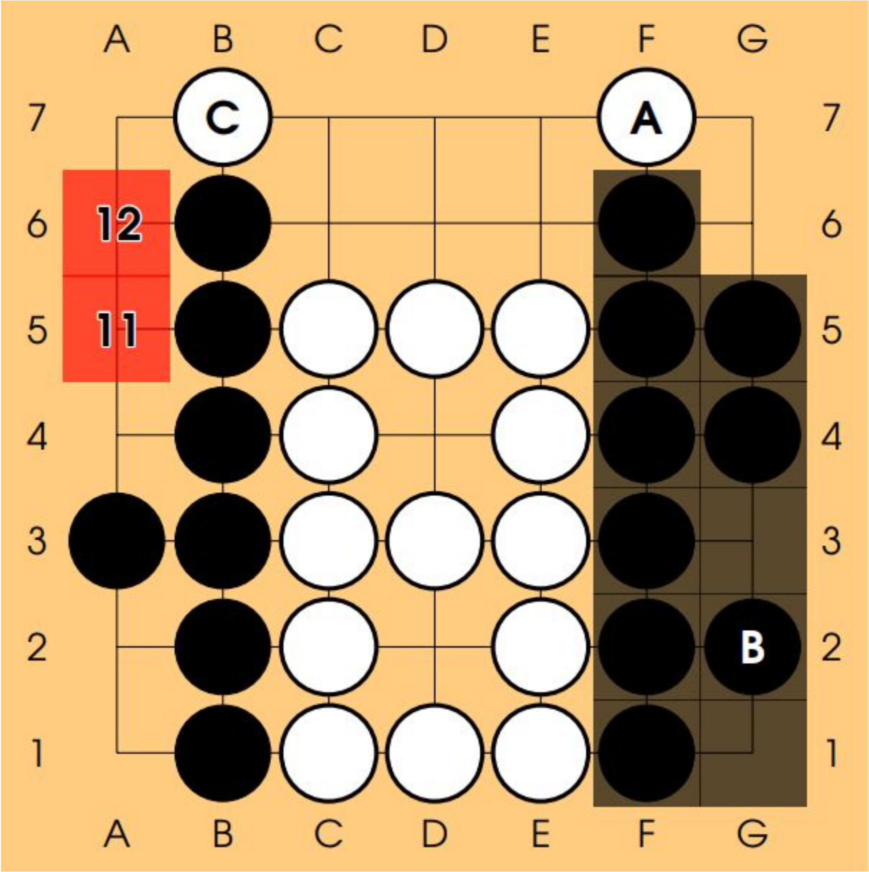
Absolute values



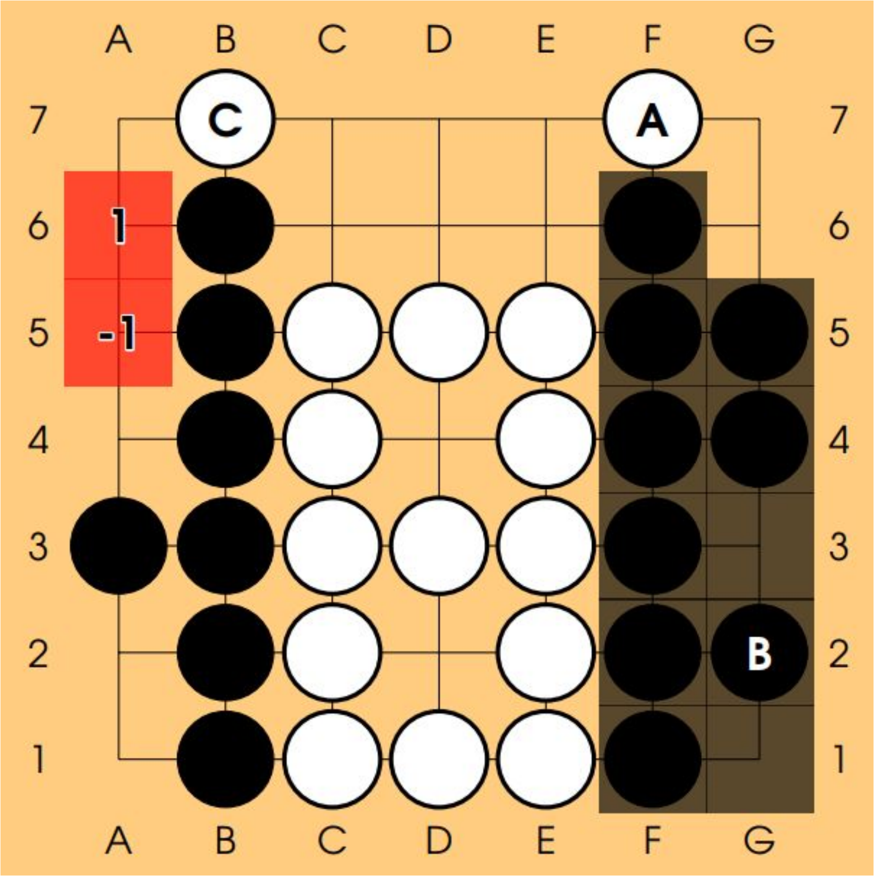

Relative values



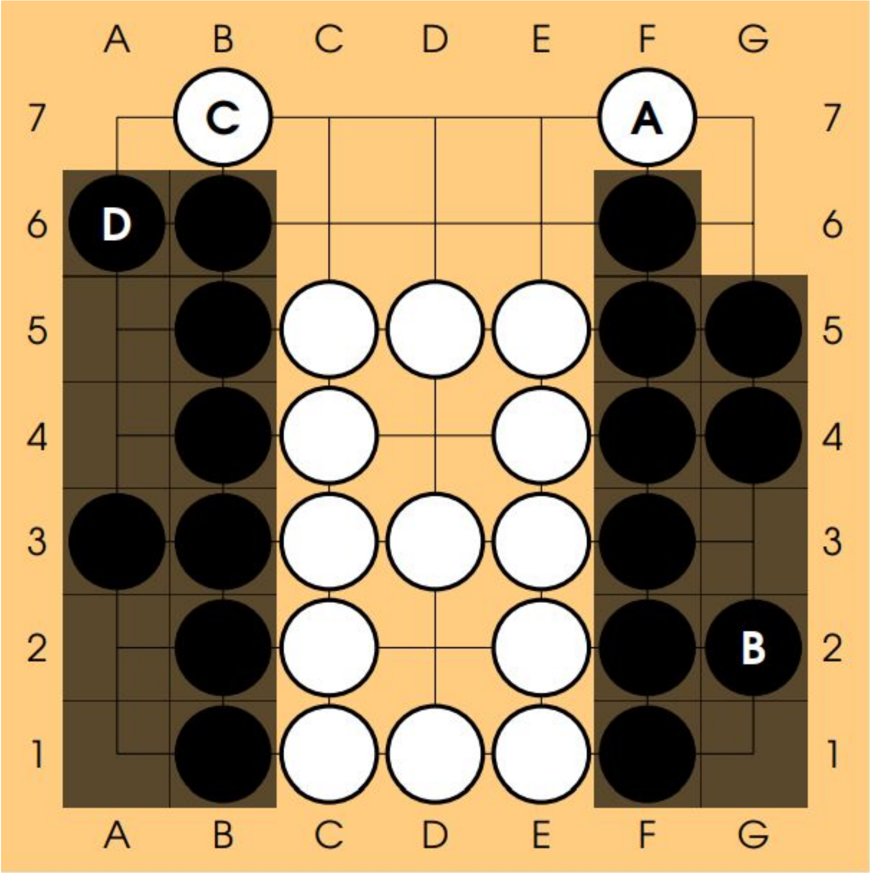
Absolute values



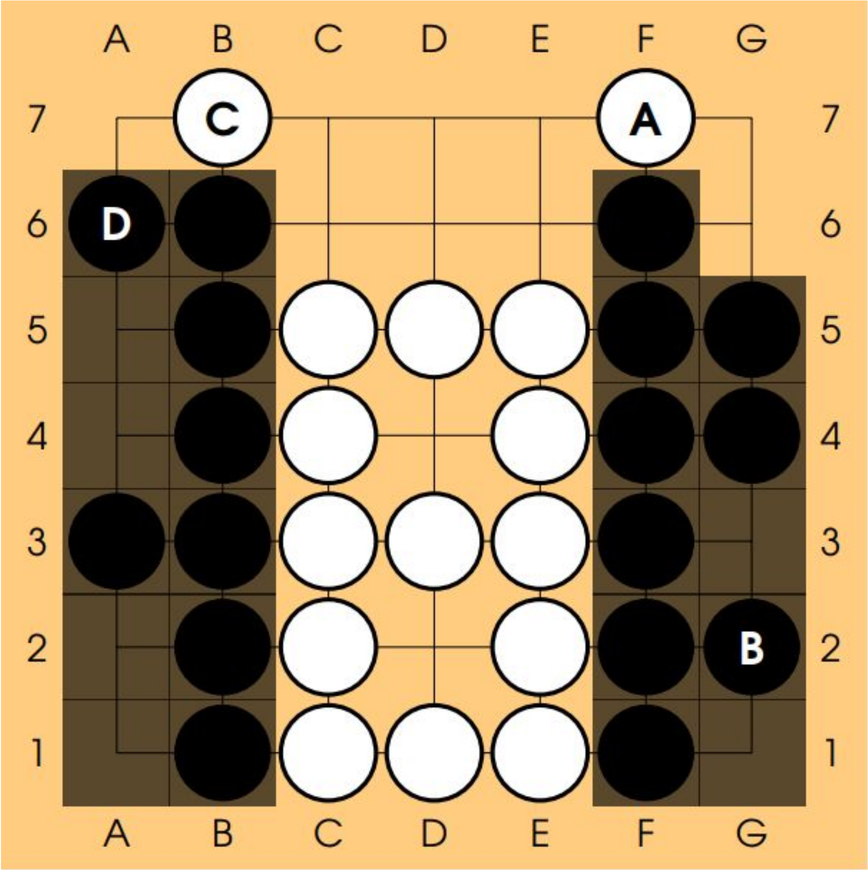

Relative values

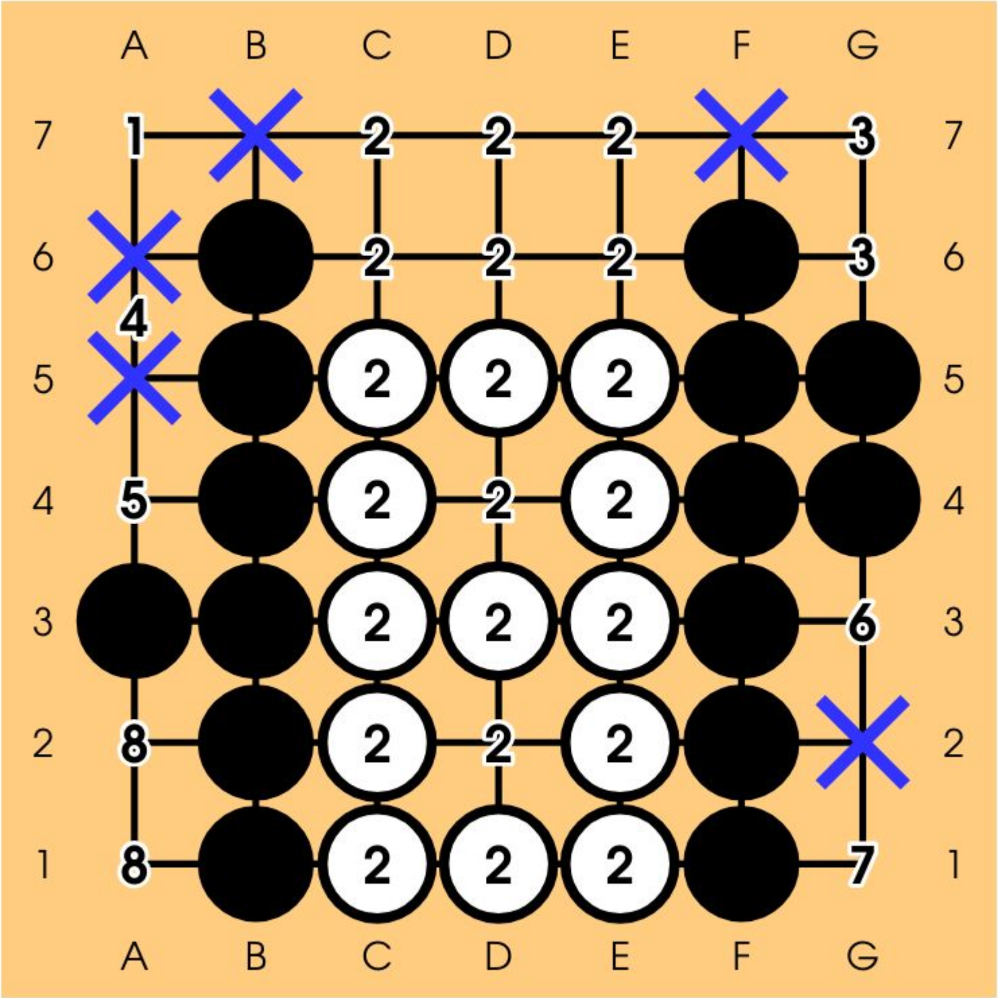


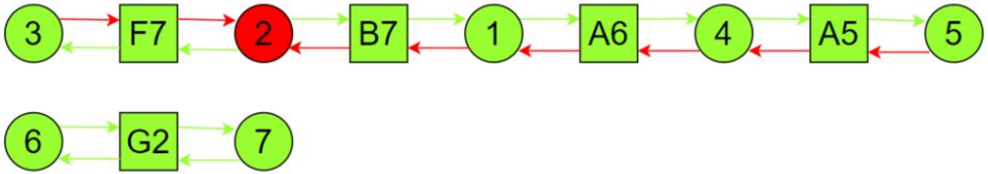
Absolute values



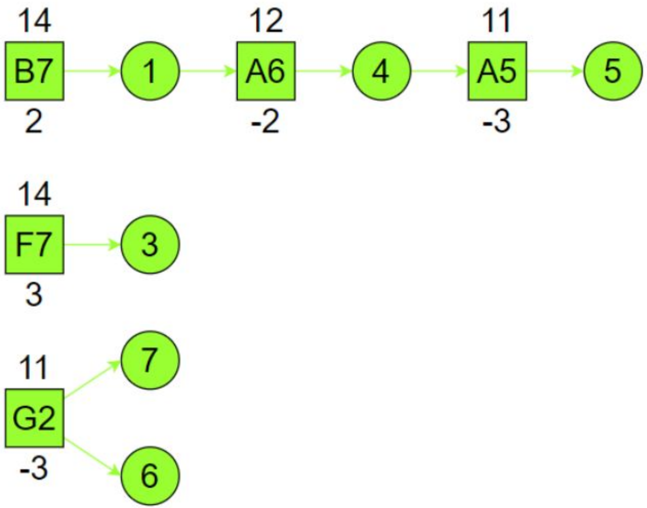

Relative values



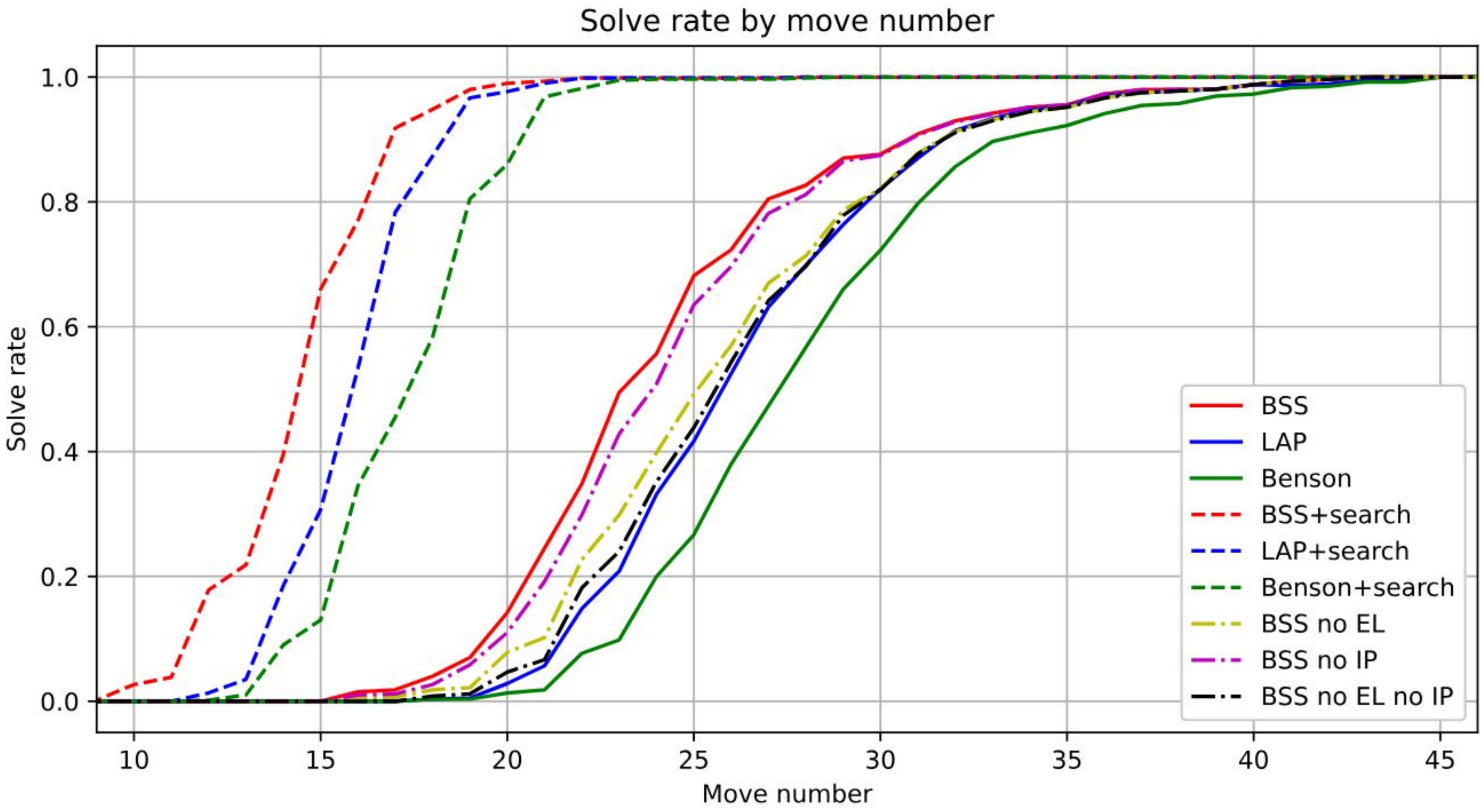

A B C D E F G
7 1 2 2 2 3 7
6 2 2 2 3 6
4
5 2 2 2 5
4 5 2 2 2 4
3 2 2 2 6 3
2 8 2 2 2 2
1 8 2 2 2 7 1
A B C D E F G




(a) IRGs of Fig. 3a




(b) Corresponding value trees


Solve rate by move number
Solve rate
Move number
1.0
0.8
0.6
0.4
0.2
0.0
10
15
20
25
30
35
40
45
BSS
LAP
Benson
BSS+search
LAP+search
Benson+search
BSS no EL
BSS no IP
BSS no EL no IP