AI Agent for Playing Gomoku

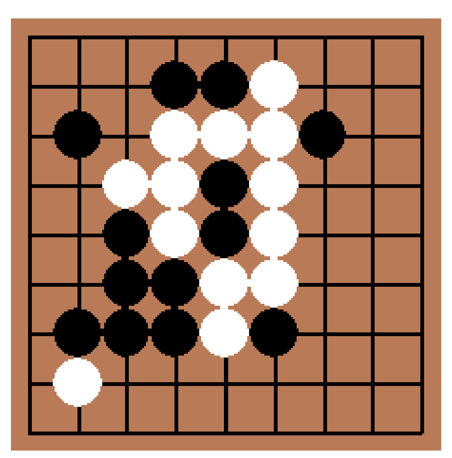
Bo Yu

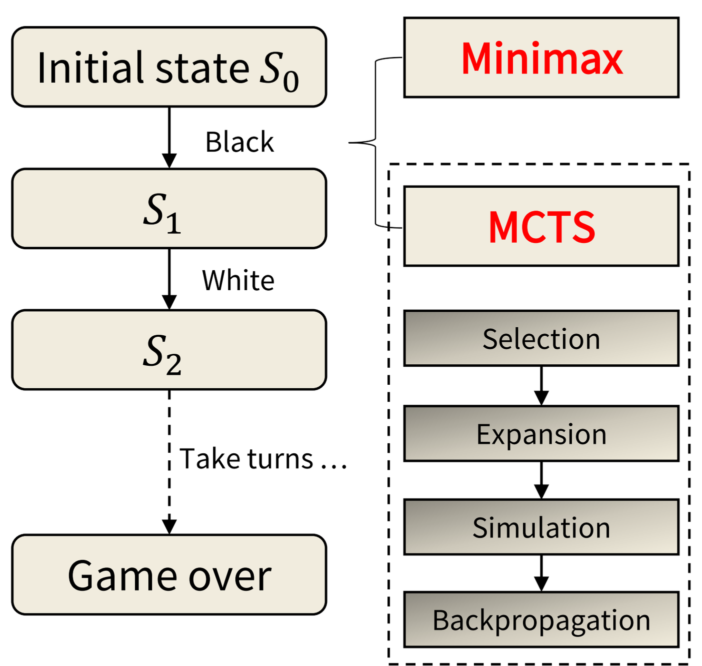
SUNet ID: boyu19

Stanford Computer Science

Problem Definition

- What is Gomoku?
  - A two-player turn-based zero-sum game
  - Played with Go pieces on a Go board
  - Winner is the player who can first obtain an unbroken chain of five pieces horizontally, vertically, or diagonally
  - White wins in the following example
- Challenges
  - Search space is huge (in a 9×9 board, there are around 80 moves and in a 15×15 board, there are around 225 moves at each state)
  - Good heuristic evaluation function requires deep domain knowledge of the game
- Scope
  - Build AI agents for playing Gomoku based on minimax and Monte Carlo tree search





Model

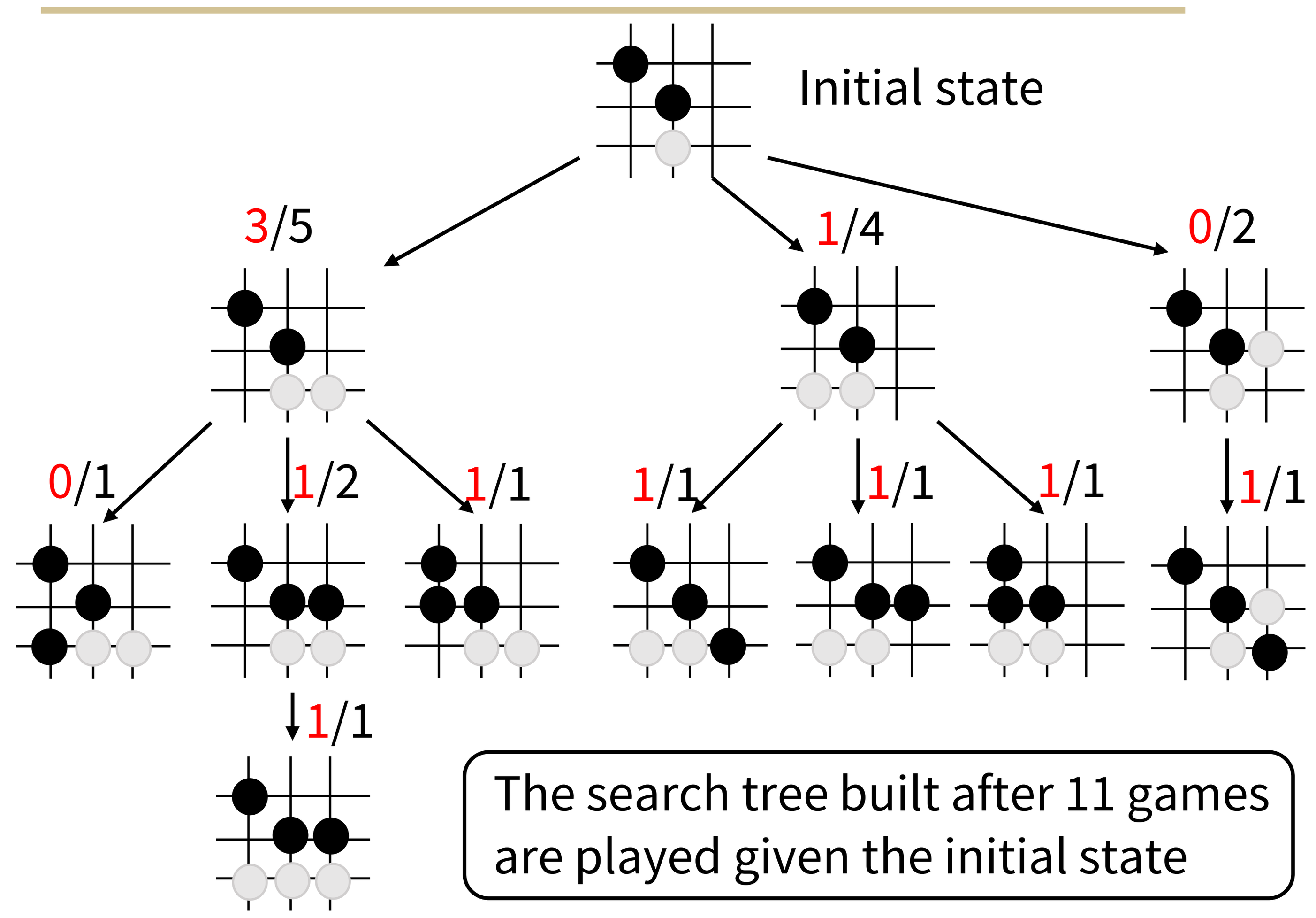
- Two-player, perfect-info, zero-sum game
  - **Players** = {black, white} (black moves first)
  - **State** s: (piece positions, whose turn it is)
  - **Actions(s):** all legal positions that Player(s) can place the piece
  - **Succ(s, a):** resulting game state if choosing action a in state s
  - **IsEnd(s):** whether s results in a five-in-a-row or draw
  - **Utility(s):** $+\infty$ if white wins, $-\infty$ if black wins
  - **Player(s) ∈ Players:** player who controls state s

Minimax with alpha-beta Pruning

- Limited depth minimax with alpha-beta pruning time complexity
  - $O(b^{2d})$ for branching factor b and depth d
  - For 15×15 board: $b \approx 225, b^2 \approx 50625 (d=1), b^4 \approx 2562890625 (d=2)$
  - For 9×9 board: $b \approx 81, b^2 \approx 6561 (d=1), b^4 \approx 43046721 (d=2)$
  - $d = 2$ is prohibitively complex
- Minimax with search space reduction
  - Not all legal actions need to be searched
  - We can reduce the branching factor using certain heuristics (like beam search) and then increase the depth of search tree
- Evaluate function
  - Features: different threat patterns

$$
\begin{aligned}
\mathrm{Eval}(s) = 6000 \cdot (N_5^{agent} - N_5^{opp}) &+ 4800 \cdot (N_{open4}^{agent} - N_{open4}^{opp}) \\
+500 \cdot \left(N_{half4}^{agent} - N_{half4}^{opp}\right) &+ 500 \cdot \left(N_{open3}^{agent} - N_{open3}^{opp}\right) \\
+200 \cdot \left(N_{half3}^{agent} - N_{half3}^{opp}\right) &+ 50 \cdot \left(N_{open2}^{agent} - N_{open2}^{opp}\right) \\
+10 \cdot \left(N_{half2}^{agent} - N_{half2}^{opp}\right) &
\end{aligned}
$$

Monte Carlo Tree Search



- Tree policy: using UCB1 algorithm
- Default policy: roll out simulations to completion
  - Random policy
  - Incorporate domain knowledge

Experiments and Results

- Environment: 9×9 and 15×15 board
- Agents
  - Baseline:
    - Greedy heuristic algorithm: randomly select one of the top K candidates according to the heuristic board evaluation function Eval(s)
  - Minimax with alpha-beta pruning:
    - Depth = 1 without search space reduction
    - Depth = 2 with search space reduction ($b = 5/10$)
  - Monte Carlo tree search:
    - Random roll-out policy with 1000/2000/4000/8000 sim. budget
    - Heuristic roll-out policy with 200/500 sim. budget
- Comparison with baseline:

Agents	Winning ratio	Average time per step (sec)
Minimax-d1	20/20	0.73
Minimax-d2 ($b = 5$)	20/20	3.3
MCTS-random-1000	7/20	15.1
MCTS-random-2000	9/20	45.5
MCTS-random-4000	13/20	105.1
MCTS-random-8000	20/20	140.7
MCTS-heuristic-200	8/20	90.3
MCTS-heuristic-500	13/20	191.3

- Comparison with minimax depth = 1

9×9 board	Winning ratio	Average time per step (sec)
Minimax-d2 ($b = 5$)	6W/10D/4L	3.9
Minimax-d2($b = 10$)	1W/9D	16.2

15×15 board	Winning ratio	Average time per step (sec)
Minimax-d2 ($b = 5$)	20L	10.7
Minimax-d2($b = 10$)	6W/3L	58.3

Analysis

- Minimax algorithm:
  - Depth = 1:
    - Performs the best considering both winning percentage and time
  - Depth = 2:
    - Need to reduce branching factor significantly, which degrades accuracy and performance
- Monte Carlo tree search algorithm:
  - With random roll-out policy:
    - No domain knowledge or handcrafted evaluation functions are needed
    - Need large number of random simulations (self-plays) to get good performance (converge to minimax policy), which is very time-consuming
  - With heuristic roll-out policy:
    - Incorporate some domain knowledge during roll-out stage instead of using random policy
    - Require a smaller number of simulations, but each simulation takes longer time

Future Work

- Minimax algorithm:
  - More sophisticated features for the evaluation function based on expert knowledge of the game
  - Use TD learning to learn the weights of the evaluation function
- Monte Carlo tree search algorithm:
  - Use deep neural networks to guide MCTS to reduce the effective depth and breadth of the search tree, thus improve the efficiency and accuracy of the algorithm

References

[1] Sylvain Gelly, David Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," Artificial Intelligence, vol. 175, Issue 11, 2011, pp. 1856-1875.

[2] C. B. Browne *et al.*, " A Survey of Monte Carlo Tree Search Methods," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no. 1, pp. 1-43, March 2012.