# Building an Intelligent Agent to Play 9x9 Go

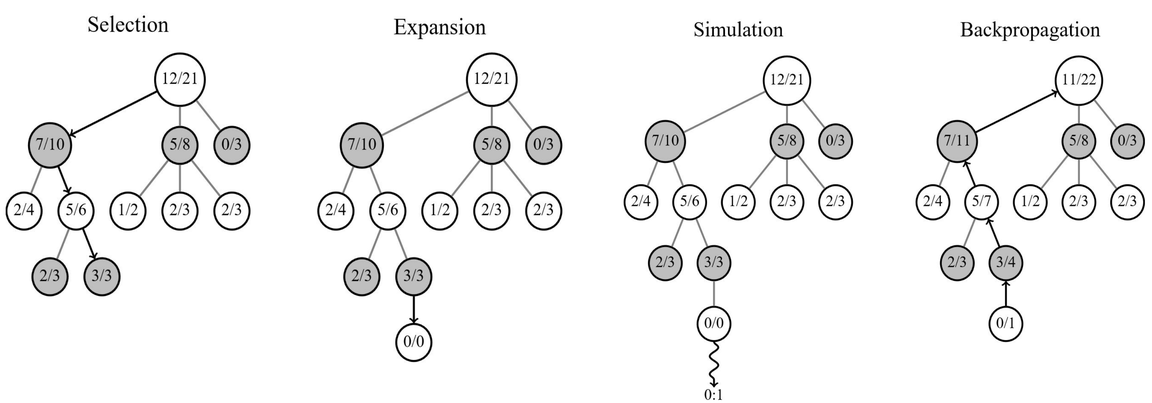
*Shawn Hu (shawnghu@stanford.edu)*

## Data

Our dataset consists of 13,175 SGF files which contain records of games played on CGOS servers. The games were played at 2500-2800 ELO (5-9 dan), a high amateur to low professional rating.

## Gameplay

Our Go agent uses the UCT variant of Monte Carlo Tree Search, the industry favorite for handling Go's large branching factor. The algorithm works by iteratively simulating games according to an initially random policy, then improving that policy using some statistical methods based on the results. In this project, the search is guided by an learned linear evaluation function.




## Features and Training

From the board state, we extracted indicators on the presence of 3x3, 2x2, and 1x1 patterns at all coordinates on the board. Symmetric patterns shared weights. From the actions taken, we extracted various features, including distance from last move, captures, and atari (threats). The weights for the features were learned from the plays and board states of the winners after every move in every game. Training was done using gradient ascent, with winning states arbitrarily assigned a value of 1 and η inversely proportional to the number of learned moves.

$$w := \eta[w \cdot \phi(s,a) - 1]\phi(s,a)$$

## Prediction

We used these weights to guide the MCTS search for efficiency purposes. Our implementation of UCT initializes tree nodes with "prior knowledge" of 1 win and 1 loss; we changed the priors for an action a and successor state s' by adding $\varphi$(s', a) extra wins (minimum 0).

## Results

Our final agent, operating at a speed of about 10 playouts per second, achieved an estimated skill level of 18kyu (low-mid amateur). For even 9x9 Go, this is decent as a first attempt; for reference, the upper bound in skill of a raw UCT agent (running at ~2000 pps) is ~5kyu. Further, the application of the features, which was the main interest of this project, made a notable difference.

| Estimated skill of agent (kyu/ELO) | |
|---|---|
| Both sets of features | 18kyu/~400 ELO |
| Action features only | 23kyu/~50 ELO |
| State features only | 20kyu/~100 ELO |
| Raw UCT-MCTS | 25kyu/0 ELO |

## Discussion & Further Approaches

- RAVE: Caching MCTS search trees for reuse in later computations
- Main difference between this bot and better ones is good, plentiful features. Higher end bots use CNNs or DNNs to do their feature learning
- Dynamic komi: Bot plays much worse when ahead because MCTS playouts are optimistic. Automatically adjust goals for petter perforamnce.
- Self-play not constructive: the agent is not good enough to learn anything
- Raw computational optimization: 10 playouts per second is actually considered extremely slow; for this reason, most mid-level Go agents are programmed in C or C++.

## References


- S. Gelly and D. Silver, "Achieving Master Level Play in 9 × 9 Computer Go," in Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)
- P.Baudis, "MCTS With Information Sharing", Masters Thesis, 2011
- E.C.D van der Werf, "Learning to Predict Life and Death from Go Game Records, 2005
- MCTS diagram: Mciura - CC BY-SA 3.0