Adversarial Game Playing Agent: Advanced Search Analysis

Adam DeJans

Analysis of Game Tree Search Methods

Let's begin by tabulating experimental results from playing Knight Isolation.

Algorithm	Depth/Sims	Timeout (ms)	Rounds	Games	Wins	Opponent
minimax	4	150	200	400	57.5%	minimax
–	–	–	–	–	–	–
alphabeta	4	150	200	400	59.5%	minimax
–	–	–	–	–	–	–
iterative deep alphabeta	4	150	100	200	62.5%	minimax
iterative deep alphabeta	5	150	100	200	62.0%	minimax
iterative deep alphabeta	7	150	100	200	77.0%	minimax
iterative deep alphabeta	20	150	100	200	75.0%	minimax
–	–	–	–	–	–	–
mcts	50	150	50	100	12.0%	greedy
mcts	50	150	50	100	7.0%	minimax
mcts	50	150	50	100	52.0%	random
mcts	50	300	50	100	48.0%	random
mcts	200	600	50	100	54.0%	random
mcts	400	1,200	50	100	49.0%	random
mcts	1,600	4,800	10	20	65.0%	random
mcts	3,200	9,600	10	20	75.0%	random

Table 1. Adversarial Game Playing Agent Performance Playing Knight Isolation.

Questions

Choose a baseline search algorithm for comparison (for example, alpha-beta search with iterative deepening, etc.). How much performance difference does your agent show compared to the baseline?

Firstly, we choose to use the fair matches flag since our experiment is not revolved around opening books. We want to see how well we do compared to the best known algorithm.

For sake of experiment we choose to compare alpha-beta pruning with iterative deepening to our implemented Monte Carlo Tree Search (MCTS). We see that MCTS is outperformed in all cases. We don't gain any performance to the baseline of alpha-beta pruning with iterative deepening.

Why do you think the technique you chose was more (or less) effective than the baseline?

The poor performance of MCTS is surprising and my natural assumption is that the MCTS is implemented very inefficiently - that is, the MCTS implementation cannot complete enough simulations to be effective within 150ms. To verify this is the case we can compare MCTS directly to the random opponent. If MCTS cannot outperform the random choice opponent with several hundred simulations than something must be wrong.

To experiment we continually increase the timeout and number of simulations performed and compare against random. We see that even after 1600 simulations we only start to perform better than random. It isn't until we begin to complete around 3200 simulations that we begin to truly outperform the random opponent. This leads to the conclusion that the algorithm is correct but inefficient.

Future Enhancements/Experiments

Open Questions

- How many simulations are necessary for MCTS to perform well?
- How do we improve MCTS runtime efficiency?

https://www.adamdejans.com Detroit, MI adam.dejans@gmail.com