Monte Carlo Permutation Search

Tristan Cazenave

Overview

- Move Selection: an existing Monte-Carlo algorithm generates w moves
- Sequence Evaluation: a new Monte-Carlo algorithm is used to evaluate depth d sequences of these w moves
- Virtual Global Search: Alpha-beta search is applied to evaluate the best possible sequence

Move Selection

- Uses previous work by Cazenave and Helmstetter
- Monte Carlo combined with tactical search
- Unlike previous paper, only connection searches are used
- The w moves with best evaluation are selected for global evaluation

Gobble

- All moves are considered to have a static value, regardless of when played
- Moves are ordered according to value
- Many games are simulated using this order, with some random variation
- A move is assigned the mean score of all games in which it is played

Sequence Evaluation

- All move **sequences** are considered to have a static value, regardless of when played, or in which order
- Many games are simulated using random move selection
- A **sequence** is assigned the mean score of all games in which it occurs (in any order)

Sequence Evaluation

- All possible depth d sequences are evaluated (d=3 in most experiments)
- Requires 2^{b*d} memory (b bits for index)
- In each random game, approximately $(w/2)^d$ sequences are matched
- Each matching sequence is updated incrementally (count and score)

Virtual Global Search

- Depth d alpha-beta search using w moves chosen by move selection
- Leaves are evaluated according to the value of the depth d sequence
- No need to actually play moves
- Instead just track the sequence index

Virtual Global Search

- Number of games required by standard global search to have g games at each leaf: $2gw^{(d/2)}$
- Number of games required by virtual global search to have g random games at each leaf: $g*2^d$
- Space complexity is linear for standard global search, but w^d for virtual global search

Experimental Results

Table 1. Comparison of times for the first move.

algorithm	w	d	*games*	*time*
standard	8	3	$g = 100$	15.8s
virtual	8	3	$g_1 = 800$	0.3s
standard	16	3	$g = 100$	118.2s
virtual	16	3	$g_1 = 800$	0.3s
virtual	81	3	$g_1 = 800$	0.6s

Experimental Results

Table 2. Comparison of algorithms.

max	*w*	*d*	g_1	%	*time*	*min*	*w*	*d*	*g*	*time*	*result*	*won*
virtual	8	3	2,000	0%	0.5s	*standard*	8	3	250	11.5s	-3.3	42
virtual	8	3	8,000	0%	2.1s	*standard*	8	3	100	7.2s	5.6	66
virtual	8	3	8,000	50%	2.2s	*standard*	8	3	100	7.0s	7.2	75
virtual	16	3	8,000	50%	1.8s	*standard*	8	3	100	6.4s	9.6	70
virtual	8	5	32,000	0%	13.1s	*standard*	8	3	100	7.6s	10	73

Experimental Results

Table 3. Results against gnugo 3.6.

max	*w*	*d*	*pre*	*g*	% *time*	*mean*	*σ*	*won*
virtual	8	3	100	100	80%	0.4s	-34.4	27.6	4
virtual	8	3	1,000	1,000	80%	3.7s	-26.6	27.7	10
virtual	8	1	1,000	4,000	80%	3.7s	-17.7	28.6	16
virtual	8	3	16,000	2,000	80%	4.7s	-16.1	23.1	17
virtual	8	3	1,000	10,000	80%	37.4s	-14.4	28.5	31
standard	8	3	100	100	80%	3.3s	-23.9	22.3	10
standard	8	1	1,000	4,000	80%	4.4s	-17.3	24.7	16
standard	8	3	1,000	1,000	80%	23.6s	-11.1	23.9	21

Conclusions

- Sequence permutations are ideal for games such as Hex where moves always permute
- In Go, moves don't always permute but this approach still gives good results
- Virtual global search requires $O(2^d)$ instead of $O(w^{d/2})$ simulations and $O(w^d)$ instead of linear memory