NOTE

ON THE DANGERS OF RANDOM PLAYOUTS

Cameron Browne[1]

London, UK

This note presents a simple counterexample to dispel the illusion that increasing the number of random playouts necessarily implies better estimates for simulation-based move planners. Incorporating a tree structure into the search helps redress the problem.

With recent advances in Monte Carlo approaches to move planning in computer games – in particular Monte Carlo Tree Search (MCTS) proposed by Coulomb (2006) – more attention is now being paid to simulation-based methods such as the use of random playouts to estimate the utility of given board positions. The general assumption is that the accuracy of these estimates will increase with the number of simulations; we present a simple counterexample that demonstrates that this is not always the case.

Consider the Gomoku situation shown in Figure 1 with Black to play (left). In Gomoku, players take turns placing a piece of their colour on a free board intersection and win by making a line of five pieces of their colour. There is one obvious move, labelled *a*, that Black must play to stop White winning next turn. However, a flat (i.e. non-tree) Monte Carlo search using strictly random playouts will choose the losing move *b* (right) in preference to the correct move *a* regardless of how many simulations are run. What is happening here?

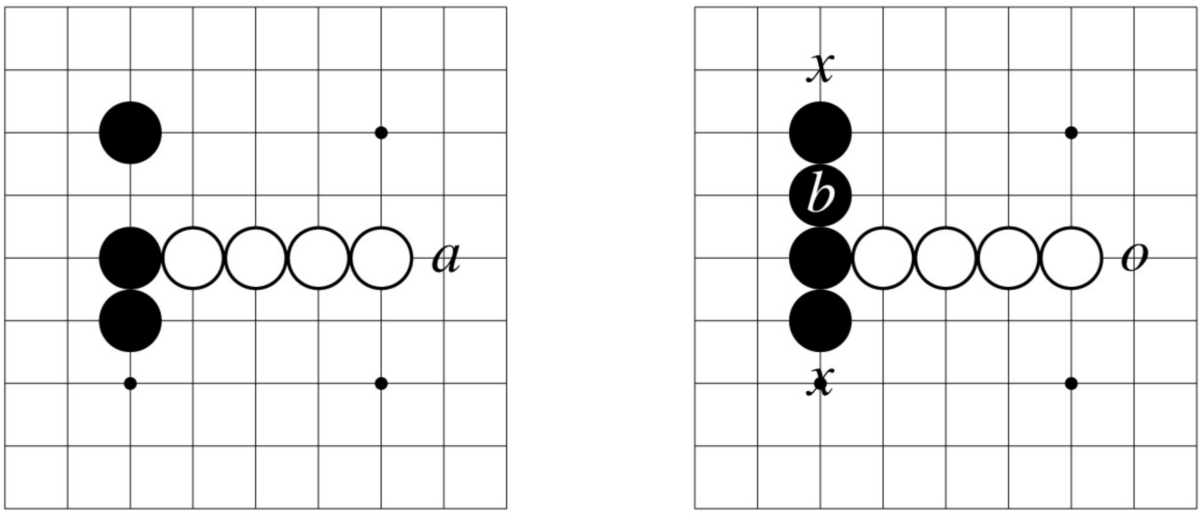

Figure 1. A game of Gomoku with Black to play (left) and a bad move choice (right).

Running 1,000,000 strictly random playouts from the board position shown in Figure 1 (left) reveals an estimated 36.29% win rate for Black (±0.19% with 95% confidence). The two standout moves for Black are *a* and *b*. If Black makes move *a* then the estimated win rate based on another 1,000,000 random playouts increases to 50.30% (±0.20%), whereas if Black makes move *b* then the estimated win rate increases to 56.53 (±0.19%), hence move *b* is incorrectly chosen for Black.

The problem is that move *b* creates two next-move wins for Black (marked *x*) as opposed to one next-move win for White (marked *o*). Since each empty point has an equal chance of being coloured either white or black during a random playout, there is a greater chance that one of the Black wins will be realised before the White win even though it is White's turn to play next; the opponent model has been lost.

[1] Computational Creativity Group, Imperial College London, UK. Email: camb@doc.ic.ac.uk

It should be pointed out that there is a slim chance that a flat Monte Carlo player will choose the correct move *a* if the process is halted after a small number of lucky playouts. However, this very small chance vanishes quickly as the number of playouts increases and the evaluation estimates converge and stabilise. Flat Monte Carlo search can be disastrous for such degenerate cases in which the number of potential wins outweigh the number of more urgent potential losses.

The situation improves dramatically when some tree structure is incorporated into the search even if the playouts are still strictly random. For example, a standard implementation of the Upper Confidence Bounds for Trees (UCT) method proposed by Kocsis and Szepesvari (2006) chooses the correct move *a* in the above example after an average of 2,059.64 search iterations (±45.48) measured over 10,000 searches. These results were obtained by sampling the UCT search every 100 iterations to check the current favourite move and terminating the search if the correct move *a* was the current favourite for two consecutive samples (in almost all cases move *a* would then remain the favourite). While flat Monte Carlo search converges quickly to the wrong result, UCT converges quickly to the correct result using the same random playout function.

While UCT does converge quickly to the correct result in this case (an average of 0.0195 ±0.000436 seconds per search on a standard laptop without optimisation – quite respectable) it should be understood that this is achieved through the eventual correction of inflated estimates for the 73 bad moves and recognition of the single good move mostly through elimination. As the tree structure develops and the search converges to the correct minimax result, the true value of this position overtakes the poor estimate initially provided by random playouts.

Depending on the domain being modelled, such cases may be detected directly for appropriate handling in a general way through the use of a simple two-ply minimax search prior to the main search, to detect winning moves and otherwise prune moves that lead to a winning reply by the opponent. Practical examples of this approach include the distinction between *winning*, *losing* and *non-losing* moves used by Browne (2008, Appendix I) for the game of Yavalath, and the *decisive* and *antidecisive* moves used by Teytaud and Teytaud (2010) for the game of Havannah, which both yielded positive results.

REFERENCES

Coloum, R. (2006) Efficient Backup and Selectivity Operators in Monte-Carlo Tree Search, In *Computers and Games 2006,* pp. 72-83.

Kocsis, L. and Szepesvari, C. (2006) Bandit based Monte-Carlo Planning, In Furnkranz, J.; Scheffer, T. and Spiliopoulou, M., eds., *Machine Learning: ECML 2006, LNCS 4212*, Springer, pp. 282–293.

Browne, C. (2008) *Automatic Generation and Evaluation of Recombination Games*, Ph.D. Thesis, Faculty of Information Technology, Queensland University of Technology, Brisbane.

Teytaud, F. and Teytaud, O. (2010) On the Huge Benefit of Decisive Moves in Monte-Carlo Tree Search Algorithms, *Computational Intelligence*, 1, pp. 359-364.