Investigating the Effects of Playout Strength in Monte-Carlo Go

Seth Pellegrino, Peter Drake

April 21, 2010

Abstract

The ancient Asian strategy game of Go has an interesting property—the best human players are still better than the best computer players. Recently, there has been a large amount of research investigating the possibility of using a Monte-Carlo simulation to estimate the relative merits of moves, and so far this technique has been met with great success. In this paper, we investigate how the accuracy of any given Monte-Carlo simulation (i.e. how close it is to a perfect play path) affects the overall playing strength of a Monte-Carlo Go program.

1 Introduction

Go, the ancient Asian game of strategy, is considered a grand challenge of artificial intelligence because of the vast search space, the lack of a known efficient static evaluation function, and the increasing endgame complexity (among others). The size of the space makes a simple alpha-beta search infeasible and the lack of an evaluation function means we cannot short circuit the search. Further, because each end game is likely to be distinct rather than one of a set of known endings, we usually cannot predict the winner of a game before the very final move.

For these reasons and more, Go artificial intelligence playing strength has lagged behind that of humans. The Monte-Carlo Go technique has eliminated some of that imbalance [1, 2, 3, 6, 11], but the best human players still defeat the best computers.

1.1 The Rules of Go

The game of Go's profound strategic depth comes from a few simple rules. The game is played on a grid of intersecting lines, typically in a 19x19 configuration. Each player, starting with Black and followed by White, takes turns playing a stone of their color on a vacant grid intersection. The game ends when both players pass consecutively, at which point each players' score is the number of stones they have remaining on the board plus the vacant points those stones surround.

Orthogonally (i.e. non-diagonally) adjacent vacant intersections are called liberties. If a group of stones has no liberties (i.e. it is surrounded), then that group is captured and removed from the board. No player may cause his stones to be captured in this way, and neither player may cause the board to have the same configuration as it did on a previous turn.[1]

1.2 Monte-Carlo Go

A Monte-Carlo tree search looks a lot like a minimax search, at first glance, with statistical sampling used as an evaluation function. Typically, the tree will grow asymmetrically as the program attempts to spend most of its time evaluating the best possible moves. Once the program reaches a leaf, it will evaluate the leaf by playing a series of random moves until the end of the game (where we may easily compute the winner). The strongest board positions will yield the greatest win/loss ratio with a sufficient number of these random playouts.

We are confident this works because, if a winning sequence of moves exists, we will eventually play it in a playout. Further, if there are multiple winning sequences (a stronger position over our opponent), we will find one member of that set more frequently than from a weaker position [1].

1.2.1 RAVE

Rapid Action Value Estimation (RAVE) is an algorithm designed to direct the asymmetric tree growth mentioned earlier [7]. It is based on the observation that, in Go, good moves are frequently temporally independent. In other words, the RAVE algorithm estimates the present value of a move based on all playouts where that move was played—it makes no distinction, in the RAVE data, between that move being the first move out-of-tree and the last move before the end of the game.

Obviously, this method obscures some subtle moves—not all moves are truly temporally independent. However, the speed with which we can gather this data (1 data point per move per playout as opposed to 1 data point per playout) outweighs the inaccuracy when it comes to estimating which moves seem the most promising.

1.2.2 Heavy Playouts

In addition to the problem of tree growth, we must also consider how we select the moves in our playouts. As early as 1993, Brugmann suggested that the strength of a Monte-Carlo-based program might be improved by adding Go-specific patterns to the playout moves [1]. In the years since, the topic has been thoroughly discussed [8, 9, 5, 4].

[1] Out of necessity, these rules are kept succinct. For a more in-depth explanation, the reader is directed to the many online tutorials such as http://playgo.to/iwtg/en/

Intuitively, the idea of strengthening playouts makes sense. If we improve the probability that a move we pick will be on the perfect play path, then we reduce the noise of our data. Thus, we may make more informed decisions with fewer playouts—indeed, if we were able to consult an oracle and always pick a move on the perfect play path, then a single Monte-Carlo run from each leaf would be sufficient.

Further, heavy playouts meld extremely well with RAVE. If a given move is on multiple perfect play paths (as the RAVE hypothesis contends), then our move predictor will have many chances to show the value of that move. So, a large benefit can be gained from finding good moves, even if only some of the time.

However, we must be aware of the pitfalls of the above approach. First, the obvious is that the more computationally intensive our "heavy" playouts are, the fewer we are able to do in the same amount of time. Thus, we must ensure that we decrease the noise of our data sufficiently to be able to make better decisions with our smaller pile of data.

Second, there is a much more subtle issue at play. The reason Monte-Carlo Go is successful is because it essentially compares the number of ways we may win from each board position. Unfortunately, if our playouts become too determined, then we are really measuring the frequency of winning moves from each board position *assuming a particular play style*. So, in making our playouts stronger, we must ensure they are still measuring the actual strength of our position rather than simply the strength of our playing style from our position.

2 Elo-Based Heavy Playouts

In investigating the relationship between playout strength and overall playing strength, we used a system devised by Rémi Coulom [4]. We chose this system because it generates a probability distribution across the legal moves using precomputed weights. Thus, we may change the strength of the playout without affecting the time each playout takes by simply varying the relative quality of the weights.

2.1 Elo Ratings

The system we used is based off of the Bradley-Terry model, which assigns each individual i a weight, γ_i. The probability that an individual i will beat an individual j is

$$P(i \text{ beats } j) = \frac{\gamma_i}{\gamma_i + \gamma_j} \tag{1}$$

and we say the Elo rating of that individual is $r_i = 400 \log_{10}(\gamma_i)$. We may generalize the model to multiple teams competing against each other by considering the γ of each team to be the product of each team members' γ. In other words:

$$P(\text{team}_{1,2,3} \text{ beats team}_{4,5} \text{ and team}_{1,3,4,6}) = \frac{\gamma_1\gamma_2\gamma_3}{\gamma_1\gamma_2\gamma_3 + \gamma_4\gamma_5 + \gamma_1\gamma_3\gamma_4\gamma_6} \tag{2}$$

In terms of Monte-Carlo Go, we may think of each individual as a chunk of Go-specific knowledge, and legal moves are teams comprised of these individuals. For example, suppose that we've defined our individuals as property 1, property 2, and property 3 (see below for some examples), and further suppose there are two legal moves on the board. If move m_1 satisfies properties 2 and 3 and move m_2 satisfies properties 1 and 2, then we say the likelihood of an expert playing move m_1 is given by $P(m_1) = \frac{\gamma_2\gamma_3}{\gamma_2\gamma_3 + \gamma_1\gamma_2}$.

In our project, when considering move m, the following were members of the team if:

1. Extension – If m was the liberty of a group put into atari[2] by the last move, and playing m would increase the number of liberties of the group.
2. Capturing – If playing m would capture
   - 2.1 an enemy group contiguous to a friendly group placed in atari by the last move.
   - 2.2 the last move played.
   - 2.3 a group that would otherwise be able to connect to the last move played.
   - 2.4 any enemy group.
3. Self-Atari – If playing m would place the resulting friendly group into atari.
4. Contiguity – If m is orthogonally adjacent to the last move played.
5. Patterns – If the 3x3 pattern around our point matched one of our common patterns as described below.

Note that the distinct levels of the capturing heuristic are considered to be mutually exclusive—any team for a capturing move could include 2.4, but if any of the other levels apply we use that individual instead.

2.1.1 Patterns

As a brief aside, let us define the orbit of a pattern to be the set of all patterns isomorphic to that pattern under any series of reflections or rotations. Due to the symmetries of a Go board, we know that if a particular pattern is good or bad for a player, then all of the patterns in that pattern's orbit are either good or bad for that player.

To compute our pattern feature, we look at the 3x3 pattern around a point (all of the neighbors, orthogonal and diagonal) and compute its orbit. Any point we will be considering is vacant, so we know there are eight points we must store. Because, in our implementation, each point is one of only four colors—vacant, black, white, or off board—this will take a total of 16 bits. Note that this is similar to the approach outlined in [10].

[2] A group with a single remaining liberty is said to be in atari

Conveniently, then, each pattern has a numerical "value" if we consider the representative bit string as a 16-bit integer. So, to compute the orbit of a pattern, we perform all possible transformations[3] and take the element of the orbit with minimal numeric value to identify the set. Using this technique, we stored the 1548 orbits that appeared at least 5000 times in our data set and considered each orbit to be an individual.

2.2 Computing Gammas

We computed our gammas in the same way as in [3]. Dr. Coulom was kind enough to make his majorization-minorization utility available[4], so we used that tool for simplicity.

2.3 Mean Log Evidence

To measure our success, we use a metric known as Mean Log Evidence (MLE). For every move in our test set, we know the board state (which allows us to compute the likelihood of an expert choosing any particular move) and the next move that the expert actually chose in the original game (call this M_i). Thus, we define the log evidence of a particular move to be $\ln(P(M_i))$, where $P(M_i)$ is our computed likelihood of the expert choosing move M_i. If we were able to perfectly predict the expert's response, then, we see that our log evidence would be zero.

So, if there are n competitions (i.e. moves), and for the ith competition the move actually made by the expert was M_i, then the MLE is given by

$$\text{MLE} = \frac{\sum_{i=0}^{n} \ln(P(M_i))}{n}. \tag{3}$$

Thus, MLE is a number in the range $(-\infty, 0]$, and the closer the MLE is to zero, the better our predictor is at emulating the expert's moves.

3 Experiments

3.1 Elo Gammas

In order to compute our individual strengths, we used a collection of 2596 games[5]. We reserved every 10th game to be a member of the test set. The remaining games were used as our data set, yielding a total of 491094 competitions. In order to determine the effect of the independence of the data upon the resulting MLE score, we varied how much of our data set we used. By using every second point, for example, we would increase the

[3] If the architecture supports circular shifting operations, each rotation may be a single instruction, and since reflection only needs to occur once, this process can be quite fast.

[4] See http://remi.coulom.free.fr/Amsterdam2007/

[5] Available at http://sites.google.com/site/byheartgo/

independence of each data point, but our data set would be halved. We skipped various numbers of moves between zero moves and 15001 moves, and the resulting MLEs can be seen in the second column of Figure 1. Also included in that table is the MLE if we assume an even distribution across the legal moves (i.e. "light" playouts).

3.2 Effect on Playing Strength

We ran our program, Orego, against Gnugo 3.7.11 for a total of 910 games. The results can be seen in Figures 1 and 2.

4 Conclusions

Interestingly, our MLE score decreased monotonically with increasing the independence of the data. This result seems to indicate that the benefit from the gained independence is outweighed by the reduction in the volume of data.

However, we did not achieve our peak win rate with our peak MLE. In fact, the two predictors with MLEs better than -3.9366 resulted in players slightly worse than the one from the move predictor with that MLE. This result seems to indicate either that MLE is not the best predictor of overall playing strength, or we very quickly reach a plateau as we increase MLE.

4.1 Future Work

Despite their apparent oddities, our playing strength results display a decidedly upward trend as the playout strength increases—purely random playouts only won 34% of their games, but our best player was able to win 88% of its games. So, as in other work, it seems clear that improving the playout strength of a program improves overall playout strength.

However, in order to facilitate the above research, we must also consider the necessity of comparing different playout strategies. Ideally, this metric would be how likely a particular strategy is to produce moves on or near the perfect play path. Then, quite simply, the strategy more likely to produce such moves would be better. However, as such a metric is both nonobvious and likely impossible, it is clear that there is much more work to be done in evaluating the relative strengths of different playout strategies.

References

[1] Bernd Brügmann. Monte carlo go. Technical report, Syracuse University, 1993.

[2] Guillaume Chaslot, Jahn-Takeshi Saito, Bruno Bouzy, Jos W. H. M. Uiterwijk, and H. Jaap van den Herik. Monte-Carlo Strategies for Computer Go. In Pierre-Yves

Moves Skipped	MLE	MLE Variance	Win Rate
Even Distribution	-5.5015	0.0799	0.34
15001	-4.7954	1.6086	0.498
5501	-4.4943	3.2789	0.741
3001	-4.3217	3.3232	0.796
2501	-4.2991	3.2531	0.804
1001	-4.1721	3.6755	0.848
200	-4.0466	3.7187	0.876
35	-3.939	3.7865	0.851
33	-3.9366	3.7336	0.88
7	-3.9044	3.7733	0.865
2	-3.8965	3.8088	0.863

Figure 1: Our Results

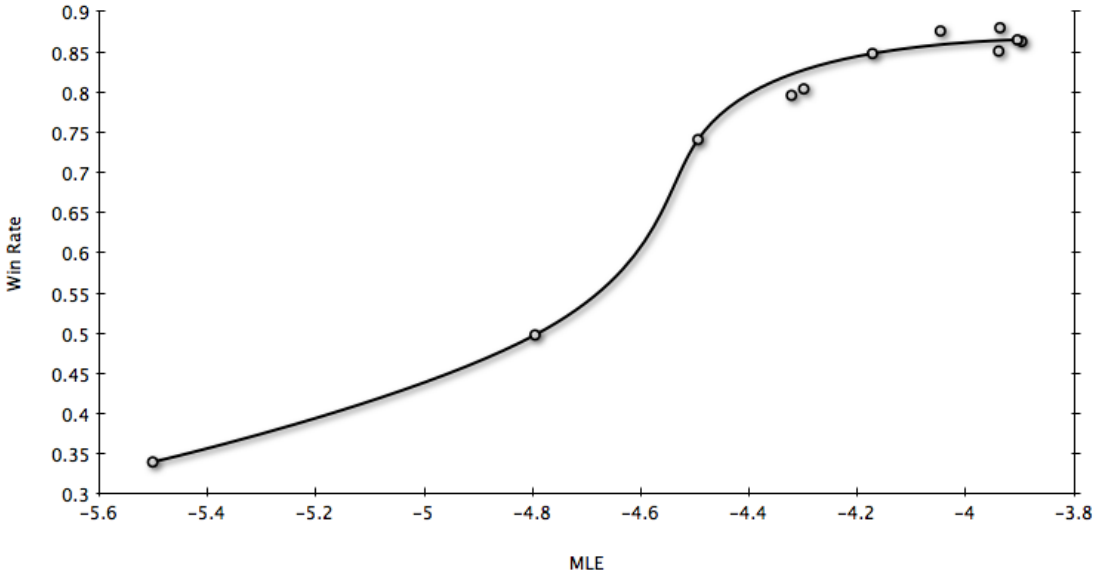

Figure 2: Win Rates playing GnuGo

Schobbens, Wim Vanhoof, and Gabriel Schwanen, editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91, 2006.

[3] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In H. J. van den Herik, Paolo Ciancarini, and H. H. L. M. Donkers, editors, *Proceedings of the 5th International Conference on Computer and Games*, volume 4630/2007 of *Lecture Notes in Computer Science*, pages 72–83, Heidelberg, Germany, June 2006. Springer.

[4] Rémi Coulom. Computing Elo ratings of move patterns in the game of Go. *ICGA Journal*, 30(4):198–208, December 2007.

[5] Peter Drake and Steve Uurtamo. Heuristics in Monte Carlo Go. In *Proceedings of the 2007 International Conference on Artificial Intelligence*. CSREA Press, 2007.

[6] Peter Drake and Steve Uurtamo. Move ordering vs heavy playouts: Where should heuristics be applied in Monte Carlo Go? In *Proceedings of the 3rd North American Game-On Conference*, 2007.

[7] Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 273–280, New York, NY, USA, 2007. ACM.

[8] Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 273–280, New York, NY, USA, 2007. ACM.

[9] Sylvain Gelly and David Silver. Achieving master level play in 9×9 computer go. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, pages 1537–1540. AAAI Press, 2008.

[10] David Stern, Ralf Herbrich, and Thore Graepel. Bayesian pattern ranking for move prediction in the game of go. icml-2006. In *in the Game of Go." 2005*, pages 873–880, 2006.

[11] Haruhiro Yoshimoto, Kazuki Yoshizoe, Tomoyuki Kaneko, Akihiro Kishimoto, and Kenjiro Taura. Monte Carlo has a way to Go. In *Twenty-First National Conference on Artificial Intelligence (AAAI-06)*. AAAI Press, 2006.