Analyzing Simulations in Monte Carlo Tree Search for the Game of Go

Sumudu Fernando and Martin Müller
University of Alberta
Edmonton, Canada
{sumudu,mmueller}@ualberta.ca

Abstract

In Monte Carlo Tree Search, simulations play a crucial role since they replace the evaluation function used in classical game tree search and guide the development of the game tree. Despite their importance, not too much is known about the details of how they work. This paper starts a more in-depth study of simulations, using the game of Go, and in particular the program Fuego, as an example. Playout policies are investigated in terms of the number of blunders they make, and in terms of how many points they lose over the course of a simulation. The result is a deeper understanding of the different components of the Fuego playout policy, as well as an analysis of the shortcomings of current methods for evaluating playouts.

1 Introduction

While Monte Carlo Tree Search (MCTS) methods are extremely successful and popular, their fundamentals have not been understood and tested well. In MCTS, statistics over the winrate of randomized simulations or "rollouts" are used as a state evaluation. One unstated but "obvious" assumption is that the quality of the simulation policy is closely related to the performance of the resulting search.

How to measure such quality? So far, three main studies have addressed this question. Gelly and Silver [7] note that a playout policy which is stronger when used as a standalone player does not neccessarily work better when used as a simulation policy in an MCTS player. Silver and Tesauro's experiments with small-board Go [11] indicate that it is more important that a simulation policy is unbiased (or balanced) rather than strong. Huang, Coulom and Lin [8] develop this idea further and create a practical training algorithm for full-scale Go.

This paper aims to re-open the study of simulation policies for MCTS, using the game of Go, and in particular the program Fuego [6], as a test case. Section 2 introduces notation, hypotheses, and research questions. Section 3 describes the methods used in this investigation, Section 4 presents the data obtained, and Section 5 outlines the conclusions drawn. Section 6 suggests future work to extend these results.

2 Some Hypotheses about Playout Policies

The following notation will be used throughout this paper: $\mathcal{P}$ denotes a playout policy. MCTS($\mathcal{P}$) is an MCTS player using policy $\mathcal{P}$ in its playouts. In the experiments, specific policies and MCTS players will be introduced.

A *blunder* can be defined as a game-result-reversing mistake. A simulation starting in position p gives the game-theoretically correct result if it plays an even number of blunders. For example, if neither player blunders, then the result of the game at the end of the simulation is the same as the game-theoretic result obtained with best play by both sides from p. If there is exactly one blunder in the simulation, then at some stage the player who was winning at p makes a crucial mistake, and is losing from that point until the end of the simulation.

The first two of the following hypotheses are investigated in this work:

1. *The strength of a playout for an MCTS engine correlates strongly with whether it preserves the win/loss of the starting position.* The correlation between evaluation before and after the playout is the most important factor in quality of playout. In Section 5.2, we report an experiment that approximately counts the number of blunders in many policies $\mathcal{P}_i$, and checks if it correlates with the playing strength of MCTS($\mathcal{P}_i$). Interestingly, the result is negative. We give a partial explanation.

2. *The size of errors made during simulation matters.* Approach: for each position, find out the score by a series of searches with varying komi. Then compare the estimated score before and after each move. Such an experiment is performed in Section 5.3, with promising results.

3. *Considering simulation balance, having no systematic bias is much more important than low error.* This hypothesis is addressed by the work on simulation balancing [8,11]. The quality of a playout can be described by bias and variance. Bias is the most important factor; after that, less variance is better.

3 Methodology

All experiments were conducted using Fuego SVN revision 1585, with some modifications outlined below, on a 9×9 board. When using MCTS, Fuego was configured to ignore the clock and to perform up to 5000 simulations per search. The opening book was switched off.

Detailed Fuego configuration files, the source code patches required to reproduce these experiments, and the raw data produced by this investigation are available for download from the Fuego wiki at `http://sourceforge.net/apps/trac/fuego/wiki/SimulationPolicies`.

3.1 Playout policies in Fuego

The playout policy used by Fuego to simulate games a leaf node of the MCTS tree is structured into several subpolicies which are applied in a top-down manner: the move

Table 1: Policy variations used in this investigation.

$\mathcal{F}$	Default policy	$\mathcal{R}$	Pure random policy
Subtractive policies		*Additive* policies	
no_ac	$\mathcal{F}$ - AtariCapture	only_ac	$\mathcal{R}$ + AtariCapture
no_ad	$\mathcal{F}$ - AtariDefense	only_ad	$\mathcal{R}$ + AtariDefense
no_bp	$\mathcal{F}$ - BiasedPatternMove	only_bp	$\mathcal{R}$ + BiasedPatternMove
no_patt	$\mathcal{F}$ - PatternMove	only_patt	$\mathcal{R}$ + PatternMove
no_ll	$\mathcal{F}$ - LowLib	only_ll	$\mathcal{R}$ + LowLib
no_cap	$\mathcal{F}$ - Capture	only_cap	$\mathcal{R}$ + Capture
no_fe	$\mathcal{F}$ - FalseEyeToCapture		

to play is chosen from the set of moves returned by the first subpolicy that proposes a non-empty set of moves. This choice is made uniformly at random, except in the case of the BiasedPatternMove subpolicy which weights moves according to machine-learned probabilities [5]. The final selected move may also be adjusted by one or more move correctors which are designed to replace certain "obviously" suboptimal moves. For specific details of Fuego's playout policy and how it works together with other MCTS components in Fuego, please refer to [5, 6].

In this investigation, variations of the default Fuego policy were obtained by activating subsets of the available subpolicies, as summarized in Table 1. In addition to the default policy (denoted $\mathcal{F}$) and the purely random (except for filling eyes) policy (denoted $\mathcal{R}$), the policy variations are divided into the *subtractive* policies (obtained by disabling one subpolicy from $\mathcal{F}$) and the *additive* policies (obtained by adding one subpolicy to $\mathcal{R}$). The subtractive policies also include the "no_fe" policy obtained by disabling the FalseEyeToCapture move corrector in $\mathcal{F}$. Note also that the "no_patt" policy excludes both unbiased and biased pattern moves.

The *gogui-twogtp* utility from the *GoGui* suite [4] was used to establish the relative strengths of MCTS($\mathcal{P}$) for all $\mathcal{P}$ in Table 1, via self-play. Specifically, each subtractive MCTS($\mathcal{P}$) was played against MCTS($\mathcal{F}$) 2000 times, and each additive MCTS($\mathcal{P}$) was played against MCTS($\mathcal{R}$) 500 times.

3.2 Blunders and Balance

Following Silver and Tesauro [11], the playing strength of MCTS($\mathcal{P}$) can be related to the *imbalance* of the playout policy $\mathcal{P}$. In simple terms, a playout policy that tends to preserve the winner of a game can be expected to produce good performance in MCTS, even if the moves played during the policy's simulation are weak.

To investigate this idea, each subtractive policy $\mathcal{P}$ (the additive policies proved too weak) was played against itself over 100 games (again using *gogui-twogtp*). Every position in the game was evaluated using MCTS($\mathcal{F}$), yielding an estimate (UCT value) of Black's probability of winning the game assuming strong play from that point. Ignoring the first 10 moves, any policy move producing a swing in UCT value from over 0.75 to under 0.25 (or vice-versa for White) was flagged as a *blunder*. This approach

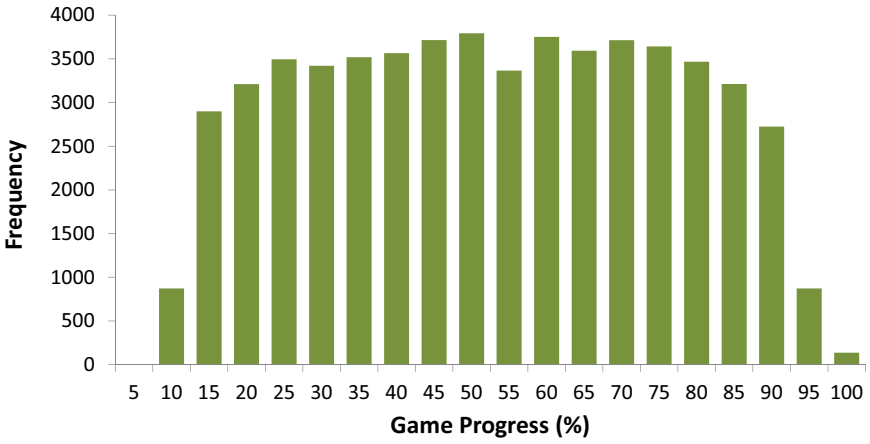

Figure 1: Distribution of sampled positions.

to empirical approximation of a blunder is similar to the one taken in [11].

3.3 Sampling Positions from Real Games

In order to obtain positions representative of actual games, random samples were taken from real 9x9 Go games played on CGOS [3]. Starting with the 57195 games from June 2011:

- All games shorter than 20 moves were removed, leaving 56968 games.
- The remaining games were labeled in increasing order of CGOS sequence number: for example, game 1616263 is labeled 1, game 1616264 is labeled 2, etc.
- From the game labeled i having n_i moves in total, the position after $10 + (i \bmod (n_i - 19))$ moves was saved as a sample. This excludes positions closer than 10 moves to either end of the game.

As shown in Figure 1, this sampling procedure produces a reasonable spread of early-game, mid-game, and late-game positions, while also being easy to reproduce using the publically available CGOS records.

3.4 Quantifying Policy Errors

Given a Go position including the player whose turn it is, a strong MCTS player such as MCTS($\mathcal{F}$) can be used as an oracle to determine Black's winning probability. Assuming that the oracle supports variation of the komi value, a binary search can be used to find the point at which Black's chances of winning change from below 50% to above 50%. The *fair komi* value where this happens is then an estimate of the actual value of the position to Black.

Simple binary search for the fair komi value is potentially frustrated by the use of an inexact oracle such as MCTS($\mathcal{F}$) since the estimates of winning probability are subject to variation. In particular, these estimates are expected to be noisiest near the fair komi value where the result of the game is most sensitive to the moves explored by MCTS. Far away from the fair komi, MCTS is expected to rapidly converge to the result that one player is guaranteed the win.

Generalized approaches to binary search given probabilistic or inaccurate evaluation functions have been previously reported in the literature [2, 9, 10]. However, in the present application, the MCTS oracle is well-behaved enough to allow a relatively simple approach. Binary search is used to find not the fair komi, but rather those komi values where Black's winning percentage passes through 40% and 60%. Then, linear interpolation is used to estimate the fair komi corresponding to a winning percentage of 50%.

For each position in the CGOS sample set, an estimate of its value was obtained using the procedure outlined above. Then, every legal move from the position was played and the value estimate for the resulting position computed: the change in position value is an estimate for the value of the move itself. These move ratings were saved as an "annotated" position file to allow for quickly rating any play from the position. Using MCTS($\mathcal{F}$) with 5000 simulations as an oracle, a typical middle game position takes about 5 minutes to annotate using this method. This functionality was added to the Fuego code itself in order to simplify the processing of the sample set.

Using the annotated positions from the test set, the play of each policy variation $\mathcal{P}$ was evaluated quantitatively. Since $\mathcal{P}$ is non-deterministic, new functionality was added to Fuego to return a list of all possible policy moves, along with their relative weighting, rather than a single randomly chosen move. This permitted the accurate calculation of the expected value of the policy's play from an annotated position.

In this setting, we redefine our approximation of a blunder as a move that changes a winning position into a loss, and that loses more than some set number of points. A threshold of 5 points was chosen based on a qualitative estimate of the precision of the move values estimated by the fair komi method.

4 Results

4.1 Playout Policies in Fuego

The statistics from policy self-play in Table 2 illustrate the relative effectiveness of the various subpolicies. In particular, the PatternMove and AtariDefense subpolicies are clearly rather effective based on the subtractive results. Also, the BiasedPatternMove subpolicy stands out among the additive results. Of note is the fact that disabling the AtariCapture subpolicy resulted in a slight but statistically significant advantage as compared to the default configuration.

Table 2: Relative strengths of MCTS($\mathcal{P}$) (2000 (left) / 500 (right) games, standard error as reported by *gogui-twogtp*).

$\mathcal{P}$	Win % vs. MCTS($\mathcal{F}$)	$\mathcal{P}$	Win % vs. MCTS($\mathcal{R}$)
$\mathcal{F}$	49.5 ± 1.1	$\mathcal{R}$	48.3 ± 2.2
no_ac	51.6 ± 1.1	only_ac	57.6 ± 2.2
no_ad	18.2 ± 0.9	only_ad	70.3 ± 2.0
no_bp	46.4 ± 1.1	only_bp	76.0 ± 1.9
no_patt	25.9 ± 1.0	only_patt	66.8 ± 2.1
no_ll	47.5 ± 1.1	only_ll	59.1 ± 2.2
no_cap	50.4 ± 1.1	only_cap	67.1 ± 2.1
no_fe	49.9 ± 1.1		

4.2 Blunders and Balance

Table 3: Blunder statistics for subtractive policy self-play (100 games).

$\mathcal{P}$	Blunder-free games	Even-blunder games	% of blunder moves
$\mathcal{F}$	53	72	3.00
no_ac	62	82	2.78
no_ad	69	85	2.14
no_bp	55	77	2.95
no_patt	38	65	4.15
no_ll	52	79	4.25
no_cap	56	71	2.67
no_fe	69	84	1.70

Blunder statistics from policy self-play are summarized in Table 3. In addition to the raw percentage of blunder moves, both blunder-free games and games with an even number of blunders were counted. If a policy tends to play an even (including zero) number of blunders, then though these moves are objectively weak the policy can still be effectively used in MCTS, since the blunders will cancel out and the policy will still preserve the winner of a game.

4.3 Quantifying Policy Errors

Figure 2 shows that a linear approximation to the neighbourhood of the fair komi is reasonably accurate, even for early-game positions. Figure 3 plots the fair komi estimate vs. move number for the CGOS samples, confirming that the sample set includes both one-sided and contested positions over a variety of game stages.

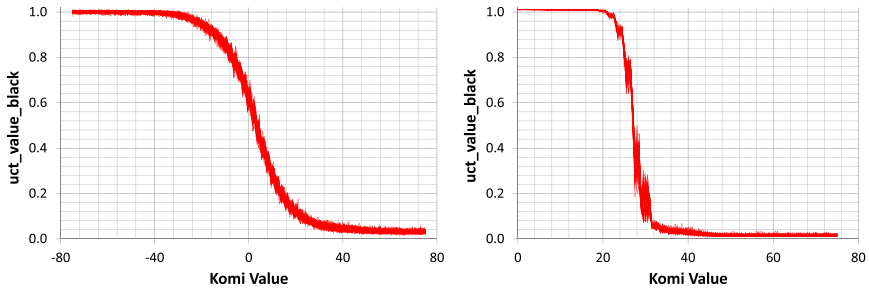

Figure 2: MCTS oracle values vs. komi for early-game position (left: 1616349 after 10 moves) and mid-game position (right: 1616808 after 45 moves).

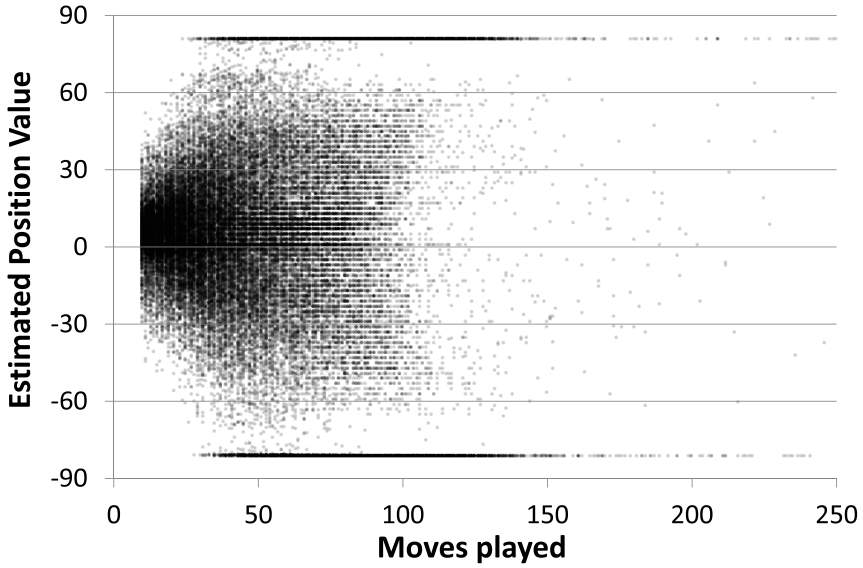

Figure 3: Fair komi estimate vs. move number in sampled positions. Darker regions represent more samples.

Table 4: Expected policy performance (per move, averaged over all CGOS samples).

$\mathcal{P}$	Blunder Chance (%)	Expected Point Loss
$\mathcal{F}$	4.14	4.52
no_ac	4.19	4.55
no_ad	4.67	4.81
no_bp	4.51	4.73
no_patt	6.55	5.57
no_ll	4.10	4.47
no_cap	4.20	4.62
no_fe	4.14	4.53
$\mathcal{R}$	9.95	7.48
only_ac	9.42	7.12
only_ad	8.22	6.46
only_bp	4.77	4.97
only_patt	5.82	5.55
only_ll	8.40	6.75
only_cap	8.63	6.61

Table 4 summarizes the data collected by evaluating policy moves on the annotated CGOS samples. Recall that in this context, a blunder was defined as a move from a winning position to a losing position that also loses at least 5 points (according to the fair komi estimate).

5 Discussion and Conclusions

5.1 Playout Policies in Fuego

The results summarized in Table 2 were for the most part as expected, although the small improvement obtained when disabling the AtariCapture subpolicy was interesting. A possible explanation for this is that the AtariCapture subpolicy sometimes eagerly suggests poor moves, while most or all of the good moves it finds would be covered by subpolicies later in the chain (e.g. [Biased]PatternMove)—in this case disabling the AtariCapture subpolicy could improve the accuracy of playouts. However, doing this increases the average playout time, since AtariCapture is a much simpler subpolicy than BiasedPatternMove, and is not effective in timed games.

5.2 Blunders and Balance

As shown in Figure 4, there was no appreciable correlation between the blunder statstics obtained from self-play of policy $\mathcal{P}$ and the relative strength of MCTS($\mathcal{P}$) for subtractive $\mathcal{P}$. It appears that a more nuanced approach is needed to test the idea that more balanced policies are more effective for MCTS. One obvious deficiency with the present method is that the positions arising in pure policy self-play differ greatly from the positions at the bottom of a typical MCTS tree. This could be addressed by

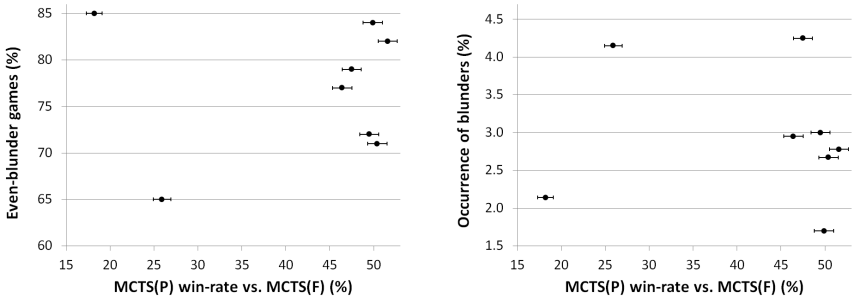

Figure 4: Policy self-play statistics for subtractive policies $\mathcal{P}$ vs. relative strength of MCTS($\mathcal{P}$) (100 playouts)

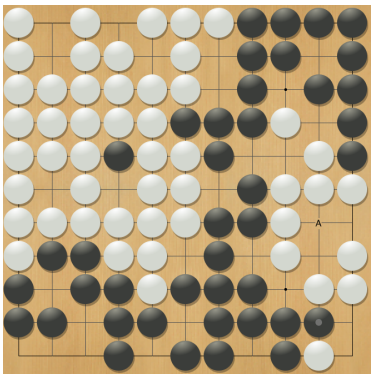

Figure 5: A nakade position leading to a series of "blunders".

initiating policy self-play from "real-world" positions instead of an empty board. A more critical issue, however, was the observed occurrence of long series of blunders in which the policy repeatedly missed a crucial play for both sides (which *is* balanced behaviour). In these cases the parity of the number of blunders was essentially determined by how many "distracting" moves were available on the board for each side.

Figure 5 shows a typical example. After Black's marked move (near the bottom right corner), the nakade move at A is the only non-losing move for both sides. Without specialized nakade knowledge in the playouts to generate that play, it is random whether Black will kill or White will make two eyes. In such situations it is difficult to ascribe significant meaning to the statistic "chance of committing an even number of blunders". The possibility of many smaller, non-blunder moves adding up to a game-reversing result also serves to frustrate the analysis focused on blunders. Predicting the strength of an MCTS player based on its policy's self-play behaviour is significantly more difficult than simply examining result-reversing moves.

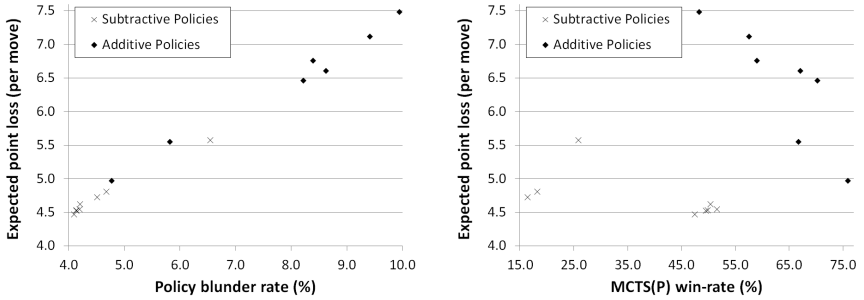

Figure 6: Expected point loss of policy $\mathcal{P}$ vs. policy blunder rate (left) and vs. relative strength of MCTS($\mathcal{P}$) (right).

5.3 Quantifying Policy Errors

Good estimates of the errors in move values estimated by the fair komi method are still needed. Simply repeating the annotation of a given position reliably produced variations in move values of at most a few points, suggesting that MCTS($\mathcal{F}$) using 5000 simulations produces reasonably precise (if not accurate) values. Time permitting, increasing the number of simulations could confirm and/or improve the current estimates. Furthermore, using other MCTS engines as oracles would provide material for an interesting comparative study; positions where estimates differ greatly could prove to be interesting test cases.

Figure 6 depicts statistics collected by analysing policy behaviour on the CGOS sample positions. First, a strong correlation between the expected point loss and the policy blunder rate was evident. This correlation is intuitively appealing, as it affirms that the occurrence of blunder moves (which were only a small portion of all the moves examined) can be linked to the strength of a policy even in non-blunder situations. Second, in comparing the expected point loss of policy $\mathcal{P}$ to the relative strength of MCTS($\mathcal{P}$), a suggestive negative correlation emerged, particularly for the additive policies (note that relative strength was measured against MCTS($\mathcal{F}$) for subtractive $\mathcal{P}$ and against MCTS($\mathcal{R}$) for additive $\mathcal{P}$: refer to Sections 3.1 and 4.1). This data supports the idea that the expected point loss of a policy $\mathcal{P}$ can be used as a predictor (though imperfect) of the strength of MCTS($\mathcal{P}$). Of course, it is also evident that expected point loss is far from the only factor.

The annotation and subsequent evaluation of policy moves on the CGOS sample positions has produced a wealth of quantitative data, but it is not yet clear how to use this data to improve policies for use in MCTS players such as Fuego. Among other ideas meriting further investigation, using the detailed statistics to optimize the gross structure of the playout procedure is particularly attractive. For example, at different stages of the game the order of evaluating subpolicies may be varied (or the subpolicies may be given different weights), guided by the expected point losses. More focused uses could include identifying pathological behaviour in specific subpolicies, and developing filters to improve them.

The annotated position files are well suited to the iterative development and/or testing of policies as the time-consuming annotation only needs to be done once (except to improve the accuracy of the annotations). They can also serve as fine-grained regression tests for full MCTS players, though the problem of identifying "ideal" test positions is not an easy one. Out of the 56968 sampled positions, MCTS($\mathcal{F}$) with 5000 simulations plays a blunder in 242 of them, which indicates that examining these positions in closer detail may help to identify weaknesses in MCTS($\mathcal{F}$).

6 Future Work

Many of the fundamental questions about simulation policies remain unanswered. Here is a long list of such topics for future work.

Quality metrics How to measure the quality of a playout policy, and of a single playout? Are there simple measures which correlate well with the observed playing strength of MCTS($\mathcal{P}$)?

From analysis to policy design Can analysis be used to identify problems with policies and lead to improvements? Can one predict whether a change in policy will lead to improved playing strength when used as part of an MCTS?

Effect of Randomization In practice, each policy $\mathcal{P}$ seems to require a "right" amount of randomization, and the performance of MCTS($\mathcal{P}$) decreases outside the optimum. Can we develop a theoretical understanding of why and how randomization works, and how to choose the correct amount?

Local vs global error Is accurate full-board win/loss estimation more important than detailed point evaluation? Can local errors in $\mathcal{P}$ cancel each other without affecting the playing strength of MCTS($\mathcal{P}$)?

Pairs of policy moves Silver and Tesauro [11] addressed the idea of two-step imbalance, which could be explored with our approach. The idea is to play two policy moves, then use binary search to estimate the resulting two-step imbalance.

Evaluation using perfect knowledge Policies can be evaluated 100% correctly if a domain-specific solver is available. Examples are Hex, where very strong general endgame solvers are available [1], and in a more restricted sense Go, where endgame puzzles with provably safe stones, which divide the board into small-enough regions, can be solved exactly.

Policy vs. short search Instead of just evaluating the move proposed by the policy $\mathcal{P}$ itself, it would be interesting to evaluate the move played by MCTS($\mathcal{P}$) with a low number of simulations, and compare with $\mathcal{P}$'s move.

Game phases Analyze the behavior of policies in different game phases, and design more fine-grained policies for each phase.

Other programs and other games Try similar experiments with other programs such as Pachi in Go, or even other games such as Hex.

Acknowledgements

Support for this research was provided by NSERC, and the Departments of Electrical and Computer Engineering and of Computing Science at University of Alberta.

References

[1] B. Arneson, R. Hayward, and P. Henderson. Solving Hex: Beyond humans. In van den Herik et al. [12], pages 1–10.

[2] M. Ben-Or and A. Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *FOCS*, pages 221–230, Washington, 2008. IEEE.

[3] D. Dailey. Computer Go Server, 2007–2013. Retrieved April 10, 2013, from `http://cgos.boardspace.net`.

[4] M. Enzenberger. GoGui, 2007–2013. Retrieved April 10, 2013, from `http://gogui.sourceforge.net`.

[5] M. Enzenberger and M. Müller. Fuego homepage, 2008-2013. Retrieved April 10, 2013 from `http://fuego.sf.net`.

[6] M. Enzenberger, M. Müller, B. Arneson, and R. Segal. Fuego - an open-source framework for board games and Go engine based on Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.

[7] S. Gelly and D. Silver. Combining online and offline knowledge in UCT. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM, 2007.

[8] S. Huang, R. Coulom, and S. Lin. Monte-Carlo simulation balancing in practice. In van den Herik et al. [12], pages 81–92.

[9] R. M. Karp and R. Kleinberg. Noisy binary search and its applications. In *SODA*, pages 881–890, Philadelphia, PA, USA, 2007. SIAM.

[10] R. Nowak. Generalized binary search. In *46th Allerton Conference on Communications, Control, and Computing*, pages 568–574, 2008.

[11] D. Silver and G. Tesauro. Monte-Carlo simulation balancing. In A. Danyluk, L. Bottou, and M. Littman, editors, *ICML*, volume 382, pages 945–952. ACM, 2009.

[12] J. van den Herik, H. Iida, and A. Plaat, editors. *Computers and Games*, volume 6515 of *Lecture Notes in Computer Science*. Springer, 2011.