Evaluation of Monte Carlo Tree Search and the Application to Go

Shogo Takeuchi, Tomoyuki Kaneko, and Kazunori Yamaguchi

Abstract— **Recent improvements to Monte Carlo tree search have produced strong computer Go programs. This paper presents a method of measuring the accuracy of Monte Carlo tree search in game programming. We use the win percentage of positions in a large database of game records as a benchmark and compare the win probability obtained by simulations with the benchmark. By applying our method to Monte Carlo tree search in Go, we found differences between search methods and their parameters, and the effect of the properties of positions such as the move numbers and the existence of stones in threats. This paper also introduces numerical metrics to evaluate the performance of search methods. Our experiments in Go, as well as Chess, Othello, and Shogi revealed that the metrics were quite close to our empirical understanding of the performance of various search methods and their parameters.**

I. Introduction

The most successful approach in game programming has been game tree searches with the assistance of evaluation functions [1]. An evaluation function with this approach estimates the goodness of a given position for the current player. This estimation is called an *evaluation value*. A popular way of constructing an evaluation function is to make it a (linear) combination of evaluation primitives called features, and adjust the weights of the combination. However, it has been difficult to construct practical evaluation functions in Go [2].

Recent improvements in methods involving Monte Carlo tree search have enabled strong computer Go programs to be created such as MoGo [3] and CrazyStone. Even though the methods have a sound theoretical background, the behaviors of the methods in various game positions have not yet been fully explored. While Monte Carlo tree search methods [4] are based on the win probability estimated by Monte Carlo simulation, the accuracy of the estimates, where accuracy is in respect to the win probability in actual game playing, has not yet been assessed. Recent strong programs use a combination of many enhancements such as patterns, RAVE, and progressive widening, but their effectiveness in accurately estimating the win probability has not yet been studied. We therefore need a method of measuring the quality of Monte Carlo simulation.

We had previously developed a method called evaluation curves to evaluate and visualize the accuracy of evaluation functions [5]. It involved using win percentages calculated from game records as a benchmark. In this paper, we discuss our application of the method to evaluating approaches involving Monte Carlo tree search methods. We modified our method so that it could be applied to Monte Carlo tree search methods, prepared suitable me records, and conducted experiments. The experiments revealed several facts. The enhanced UCT is always better than plain UCT and simple Monte Carlo simulations. The number of simulations affects performance significantly. The performance of enhanced UCT is relatively invariant with respect to the progress of the game while others depend on its progress.

Shogo Takeuchi, Tomoyuki Kaneko, and Kazunori Yamaguchi are with the Graduate School of Arts and Sciences at the University of Tokyo, JAPAN (email: {takeuchi,kaneko,yamaguch}@graco.c.u-tokyo.ac.jp).

In addition, we adopted the performance metrics used in machine learning as a simple means of comparing the accuracy of Monte Carlo tree search methods as well as that of other methods of evaluation.

We compared the performance of Monte Carlo tree search methods in Go and evaluation functions in Chess, Othello, and Shogi. As far as we know, this is the first time a comparison of the performance of evaluation functions in different games has been reported.

This paper is structured as follows. First, related work is reviewed. Then, our method of evaluating Monte Carlo tree search is presented, followed by experimental results. Finally, conclusions and future work are discussed.

II. Related Work

This section briefly reviews the literature on Monte Carlo Go/applying Monte Carlo methods to Go and evaluations of the accuracy of game tree search.

A. Applying Monte Carlo Methods to Go

1) History and Basic Model: In imperfect-information games including Bridge [6], Scrabble [7], and Poker [8], sampling-based approaches have been widely adopted. Abramson [9] presented a method of using random sampling for evaluation. Applying the Monte Carlo method to Go was first introduced by Brügmann [10], and was later investigated by Bouzy and Helmstetter [11]. Monte Carlo Go utilizes the results of random sampling to evaluate positions, instead of hand-coded evaluation functions of heuristic search used in two-player games.

In a classical and basic model, it performs a one-ply search and computes an “expected score” for each node. It then selects the move with the highest score. The expected score of a position is defined as the average of the final scores in the terminal positions of all random games starting from that position. We call these random games *playouts*. A fixed number of playouts is played at each leaf. In a random game, each player almost randomly plays a legal move, except for one filling in an eye point of that player,[1] until a position is reached where neither player has an effective move. However, diminishing returns with additional playouts has been confirmed for this basic model [12].

[1]Filling one’s eye is an extremely bad move in Go.

Many enhancements have been proposed. For example, many programs have confirmed that the win probability is more suitable for the expected score of a position, than the average of the final scores in the basic model. We will elaborate on this improvement later. The largest improvement in recent programs over the one-ply model is the development of methods that recursively extend nodes and run playouts intensively on effective moves [13], [4].

2) UCT: UCT [4] is a popular method in Monte Carlo tree search based on the theory on the multi-armed bandit problem. It recursively extends the most effective node in a best-first manner where effectiveness is evaluated by the win probability of a node while considering variance. Monte Carlo simulation is conducted at the leaf nodes in a UCT tree. There are some variations in the formula and in values to identify the effectiveness of a node in UCT [14], [15].

Let p_i be the win probability in s_i playouts undertaken for the i-th move at node a, and n be the sum of the number of playouts carried out at the descendants of node a. UCB1 selects node a to extend it, which maximizes:

$$p_i + \sqrt{\frac{2\log n}{s_i}}.$$

UCB1-Tuned is an improved version of UCB1 and selects node a to extend it, which maximizes:

$$p_i + \sqrt{\frac{\log n}{s_i} \min\left(1/4, p_i - p_i^2 + \sqrt{\frac{2\log n}{s_i}}\right)}.$$

Moreover, state-of-the-art programs are enhanced by many heuristics such as patterns or progressive pruning to obtain more reliable results. Patterns can be statically obtained by analyzing games records [16] or dynamically analyzing games during play [17]. The relationship between the strength of programs and the quality of patterns used to select moves in playouts has been reported to be unclear, where quality means the accuracy with which moves are predicted in game records [18].

B. Accuracy of Game-Tree Search

The accuracy of heuristic search is usually measured indirectly by comparing two programs in self-play. The problem with this method is that it is very time consuming to obtain statistically significant results.

We can occasionally directly compare evaluation values if more information is available. If a theoretically correct evaluation is available in a database or found by an exhaustive search, the errors in evaluation can be directly calculated. Examples are endgames in Othello [19] and Awari [20]. However, the domains where such analyses can be applied are limited. Similarly, we can see how the evaluation values for each position produced by an evaluation function agree on the preferences of human players, if positions with the assessments made by human players are available [21]. The applicability of this method is limited to domains in which such assessments can be carried out. Our method requires positions to be labeled with just their win/loss for the black player.

We first introduced an approximation of the winning probability by observing the probability in game records and visualizing this by using evaluation curves [5]. While our previous paper focused on heuristic search methods, this paper focuses on our evaluation of Monte Carlo tree search. In this paper, we also propose numerical performance metrics that enable the performance of evaluation methods including Monte Carlo tree search to be quickly compared. The metrics are adopted from those for measuring the quality of the binary prediction model in machine learning [22], [23], [24].

III. Win Probability and Evaluation Curves

A. Win Probability in Game Records

A program based on Monte Carlo tree search methods estimates the win probability by simulations for each position, in order to select the best move at the root node and to select the next frontier node to be extended. The estimated probability should be close to the "true" probability for that program to win to ensure the moves selected by the program are good moves toward winning the game. In this paper, we present a method of testing the accuracy of the win probability estimated by Monte Carlo simulations by comparing it with the "true" probability. Here, we use the winning percentage observed in game records instead of the "true" probability, because it is difficult to calculate the "true" probability. To clarify the distinction between the two probabilities, we call the win probability estimated by Monte Carlo simulations the *evaluation value*, and the win probability calculated from the game records simply the *win probability*. Now, assume that there are numerous game records, R, that contain unbiased positions. Utilizing R, we define the winning percentage as a function of evaluation value v and R as

$$\textit{Win probability(v, R)} = \frac{|B_v(R)|}{|B_v(R)| + |W_v(R)|}, \qquad (1)$$

where

$$\begin{aligned} P_v(R) &= \{p \in R | v - \frac{\delta}{2} \le \textit{eval}(p) < v + \frac{\delta}{2}\}, \qquad (2) \\ B_v(R) &= \{p \in P_v(R) | \textit{winner}(p) \textit{ is the black player}\}, \\ W_v(R) &= \{p \in P_v(R) | \textit{winner}(p) \textit{ is the white player}\}. \end{aligned}$$

Here, p is a position in R and δ is a non-negative constant standing for an interval. To compute this win probability, we first compute the evaluation value for each position in the game records. We also determine the winner of all positions. Although it is usually difficult to determine the theoretical winner of a position, we used that of a game record as the winner of all positions that appeared in the record. This worked sufficiently well in our experience. Finally, we aggregate the numbers of wins $|B_v|$ and losses $|W_v|$ for each interval $[v-\frac{\delta}{2}, v+\frac{\delta}{2})$, and calculate the fraction using Eq. (1). This calculation was first proposed in our previous paper [5] for assessing heuristic evaluation functions, where we used the value of evaluation functions as *eval*(p) in Eq. (2). For

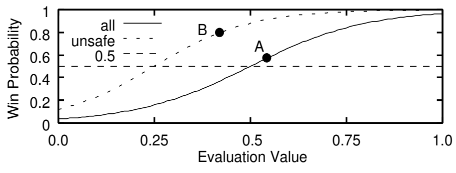

Fig. 1. Example of poor evaluation function

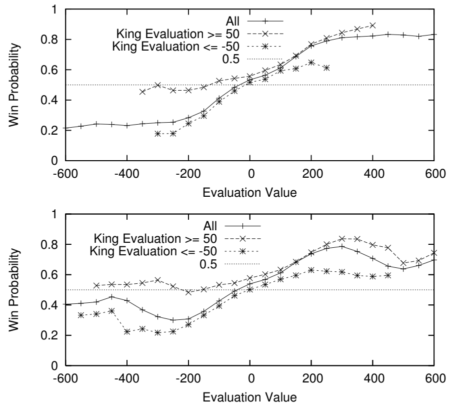

Fig. 2. Evaluation curves in Chess (King Safety) (upper: with quiescence search and lower: w/o quiescence search)

Monte Carlo tree search method, we use the results of the Monte Carlo simulations as $eval(p)$.

B. Evaluation Curves

The relationship between the win probabilities and evaluation values can be visualized by plotting it with evaluation values along the horizontal axis and the win probabilities along the vertical as shown in Fig. 1. We call this curve an *evaluation curve*. Of course, the evaluation curve of good simulations must be monotonically increasing. However, this is not sufficient to ensure that the simulations are sound.

Assume that we have a very simple Monte Carlo program for the basic model described in Section II. We then plot an evaluation curve (solid) for all positions and another evaluation curve (dotted) only for positions satisfying a certain condition such that the opponent's groups are unsafe. The plotted evaluation curves may split as seen in Fig. 1. Suppose that there are two positions X and Y, and that position X is at B and Y is at A in the figure. Then, the Monte Carlo simulations assign a higher evaluation value to Y even though X has a greater probability of a win, which is not beneficial. As demonstrated by this example, by plotting evaluation curves for positions satisfying various conditions, we can find a problems with simulation methods.

We call an evaluation curve using all positions a *total curve*. We call an evaluation curve using part of the positions for which a condition holds a *conditioned curve*. How well the evaluation method is working under the conditions can be found by comparing the total curve and conditioned curves. For example in Chess, the solid curve in the upper graph in Fig. 2 indicates that the total evaluation curve obtained by using a quiescence search with Crafty and other curves in the graph are conditioned curves. They split when the King is Unsafe. The lower graph in Fig. 2 is plotted for the same condition except for the absence of a quiescence search. As in the graph, the evaluation curves are not always monotonically increasing. We will present and explain the evaluation curves for Monte Carlo tree search in Section IV.

C. Application of Classification Performance Metrics

As a measure of performance of Monte Carlo tree search and other search methods, we employ performance metrics widely used in supervised learning. Here, we view a search method as a classifier that divides positions into win and lose, and the classified results are measured on game records.

From the nine metrics discussed by Caruana and Niculescu-Mizil [24], we selected six suitable metrics that did not make any assumption in the range of classifiers' output. First, let us introduce some basic definitions that will be used later. Let a_i be the theoretical win/lose represented by 1/0 for a position, p_i. Let v_i be a value produced by the classifier. v_i is in $[0, 1]$ if it is an estimated win probability produced by recent Monte Carlo search methods, and is in $(-\infty, \infty)$ if it is produced by old-style Monte Carlo searches or heuristic evaluation functions. To obtain binary output b_i from v_i, we set a threshold, t, for each classifier and calculate b_i as $b_i = 1(v_i \geq t), b_i = 0(v_i < t)$. For a simple case of v_i in $[0, 1]$, 0.5 is used for t. Let *TP* denote true positives, *FP* denote false positives, *TN* denote true negatives, and *FN* denote false negatives. They are the sets: $TP = \{i|a_i = 1, b_i = 1\}$, $FP = \{i|a_i = 0, b_i = 1\}$, $TN = \{i|a_i = 0, b_i = 0\}$, and $FN = \{i|a_i = 1, b_i = 0\}$. Then, precision and recall are defined as follows:

$$Precision = \frac{|TP|}{|TP| + |FP|},\ Recall = \frac{|TP|}{|TP| + |FN|}.$$

Precision is a fraction of true positives over classified positives, and recall is a fraction of classified positives over true positives. There is usually a trade-off between precision and recall.

Now, let us introduce the six metrics:

1) Accuracy (ACC): accuracy is defined as $\frac{|TP|+|TN|}{|Total|}$. The accuracy ranges from 1.0 to 0.5 by negating output if necessary.
2) F-score (FSC): F-score is defined as
$$\frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}.$$
The F-score ranges from 1.0 to 0.
3) Lift (LFT): lift is a fraction of true positives in the top T% of samples ordered by their v_is. Formally,
$$LIFT = \frac{|TP \text{ in the top T\% samples}|}{|\text{top T\% samples }|}.$$
We used T% = 25% as in Caruana and Niculescu-Mizil [24].
4) Area under ROC curve (ROC): The ROC curve is a plot of the fraction of true positives along the vertical axis and that of false positives along the horizontal axis

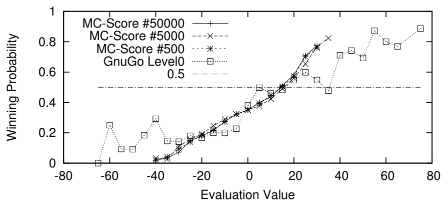

Fig. 4. Evaluation Curves for MC-Score and GnuGo

TABLE I
PERFORMANCE OF EVALUATION METHODS

method	#playouts	ACC	FSC	LFT	ROC	APR	BEP
Fuego		0.715	0.778	0.886	0.808	0.813	0.740
UCT	50,000	0.635	0.708	0.764	0.697	0.735	0.655
	5,000	0.610	0.665	0.733	0.667	0.729	0.636
	500	0.572	0.608	0.671	0.617	0.658	0.605
MC	50,000	0.620	0.692	0.739	0.675	0.706	0.641
	5,000	0.616	0.678	0.730	0.667	0.672	0.637
	500	0.587	0.592	0.679	0.620	0.632	0.616
MC-Score	50,000	0.575	0.659	0.480	0.490	0.536	0.490
	5,000	0.570	0.646	0.480	0.491	0.492	0.494
	500	0.553	0.606	0.489	0.495	0.495	0.498
GnuGo	level 4	0.655	0.694	0.584	0.531	0.500	0.536
	2	0.653	0.691	0.582	0.530	0.499	0.536
	0	0.650	0.690	0.580	0.530	0.497	0.536

for various threshold t. The area under the ROC curve ranges from 1.0 to 0.5 by negating output if necessary. See Fawcett [23] for details.

5) Average precision (APR): average precision is the average value of precision computed with thresholds where the values of recall are $\{0.0, 0.1, ..., 1.0\}$.
6) Precision/recall break even point (BEP): The BEP is the value of precision when we use a threshold where precision is equal to recall.

It should be noted that these metrics are sensitive to test cases (positions, in our case). For example, consider a program that always returns 1 (win) for all positions. Its accuracy is 1.0 if samples are all positives and it is 0.0 if the samples are all negatives. Thus, the metrics over different test cases should be carefully compared. However, visualizing evaluation curves for different sets of positions is useful for detecting problems with classifiers.

IV. EXPERIMENTAL RESULTS

A. Programs and Game Records

Let us first explain the game programs and records we used in our experiments. We used the following four Monte Carlo search methods and GnuGo.

- Fuego: We used Fuego[2] version 0.1.1. as an enhanced UCT program with various enhancements including patterns. This is a strong program with a rate of about 2,300 in the 9×9 version of Internet Go server (CGOS) [3]. Evaluation was carried out by using the "genmove" gtp command and we used the win probability of the best move. Fuego automatically determines the number of playouts, which averaged about 70,000 in our experiments.
- UCT: We used the implementation in libego (version 0.116)[4] as a plain UCT program. Its rate is about 1,800 in CGOS (9×9). The win probability of the best move was used for the evaluation value of a position, as in the experiments with Fuego.
- MC: We also used the Monte Carlo component of libego to measure the quality of simulations played at the leaves in UCT. The win probability of the root node was used as its evaluation value since that of the best move was not available. To enable a fair comparison with UCT, the number of playouts was adjusted to the given threshold divided by the number of legal moves. This was because almost the same number of playouts was undertaken for each legal move in MC.
- MC-Score: The MC-Score was a modified version of MC, which computed the averaged leaf scores instead of the win probability. The reason the MC-Score was used was to assess the quality of simulations in the basic (old) model explained in Section II-A.
- GnuGo: We used GnuGo[5] version 3.7.12 as a traditional program. The evaluation value of the root node and that of the best move are identical in GnuGo.

All programs were run with a Chinese counting rule because some did not support Japanese counting.

We used records played on a 9×9 board at the Kiseido Go Server (KGS) for game records. We obtained the records for games played from 2001 to 2005. The records collected were under various conditions of komi, the players' rate, and handicap. We mainly used the records under the condition of komi 0.5, a rate that is over 3 k (strong), and no handicap. There were 2,000 records that include 111,946 positions. We will discuss this issue of game records in Section IV-E.

B. Evaluation Curves and Performance Metrics

Here, we will present the evaluation curves for all programs. The vertical axis of an evaluation curve indicates the win probability for the black player computed from game records. The horizontal axis indicates evaluation values, which are the estimated scores for MC-Score and GnuGo and the win probability estimated by simulation for the other programs. We omitted intervals that consisted of fewer than 100 positions from all evaluation curves.

Fig. 3 (left) plots the evaluation curves for Fuego. The curves are very close to the line, $y = x$. This means that Fuego performed very well in predicting the probability of human players winning win in the game records. A difference at both ends ($x > 0.95$ or $x < 0.05$) appears in almost all other graphs and the reason will be discussed later in Section IV-E.

Figs. 3 (center) and (right) plot the evaluation curves for UCT and MC with various numbers of playouts. In both graphs, the curves for 500 playouts plotted with '*' are

[2]http://fuego.sourceforge.net/
[3]http://cgos.boardspace.net/9x9/
[4]http://www.mimuw.edu.pl/%7elew/hg/libego/
[5]http://www.gnu.org/software/gnugo/gnugo.html

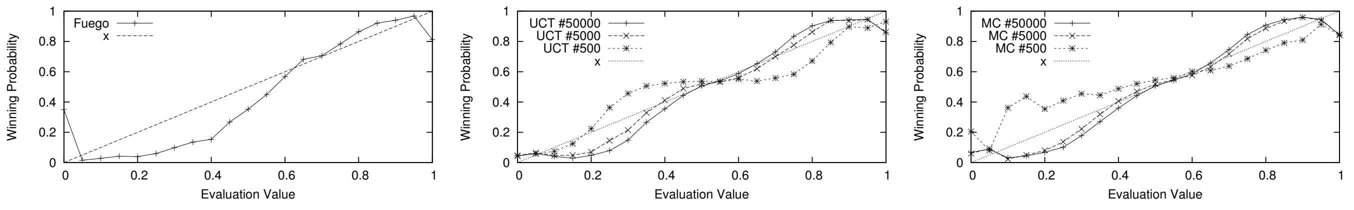

Fig. 3. Evaluation Curves for Fuego (left), UCT (center), MC (right)

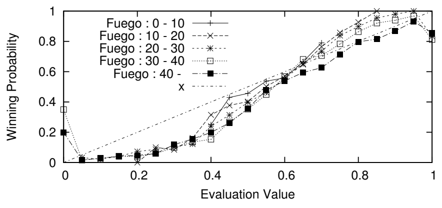

Fig. 5. Fuego with various move numbers

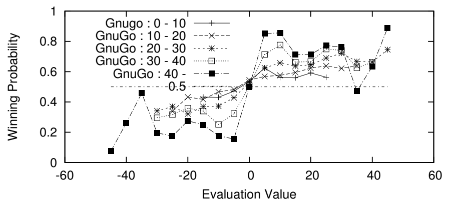

Fig. 7. Gnu Go (level 0) with various move numbers

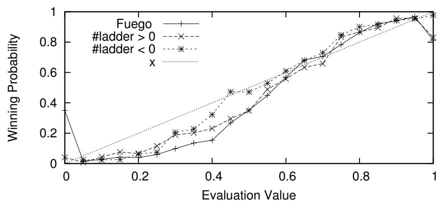

Fig. 8. Fuego with ladder position

different to the other curves, while the curves for 5,000 and 50,000 playouts are similar. Note that the win probability obtained from records will differ even when positions have the same estimated win probability in simulations if the number of playouts varies. We may reduce these differences by adding an adjusting term to Eqs. (II-A.2) and (II-A.2). This would be an interesting topic for further research.

Fig. 4 plots evaluation curves for MC-Score and GnuGo. The horizontal axis of this graph is for the score in the range of $[-81, +81]$. Surprisingly, the curves for the MC-Score are almost the same for all sample sizes. This might be the effect of diminishing of returns reported by Yoshimoto et al. [12]. The curve for GnuGo is not monotonously increasing. This suggests that the evaluation of GnuGo is different from the win probability calculated from the human players' records.

Let us now present the performance metrics in Table I, which were described in Section III-C. First, let us discuss the results on the accuracy (ACC) metric. Here, we can assume that the accuracy ranged over 1.0 to 0.5, because if accuracy is below 0.5 we can obtain a better results by negating the estimates. We can see from the table that UCT, MC, and the MC-Score with a larger number of playouts achieved better accuracy and GnuGo with higher levels achieved better accuracy. This is consistent with our observation that programs with larger numbers of playouts or higher levels are stronger. Fuego was the best programs followed by GnuGo, UCT, MC, and the MC-Score with respect to the accuracy metric. This order is consistent with our empirical assessment of the strengths of these programs. The results of FSC, LFT, ROC, APR were similar to that of ACC. The exception is that the results of the MC-Score and GnuGo were not consistently improved by increasing the number of playouts or promoting levels. The results of BEP were also similar. However, it is unnatural for MC to have achieved the best result with only 500 playouts. Consequently, using BEP is not recommended from the results of this experiment.

C. Evaluation Curves for Varying Game Progress

We examined the evaluation curves for various sets of positions to find what caused the differences in Table I.

Here, we focused on the move numbers of positions. Fig. 5 plots evaluation curves for Fuego. In the figure, '+' is for positions with less than 20 moves , and '$\times$' is for positions whose move numbers are in $[20, 30)$, and so on. We can see that the evaluation curves in Fuego almost fit into one curve meaning that its evaluation is consistent throughout the progress of the game.

Fig. 6 plot the evaluation curves for MC and UCT for sets of varying move numbers. We can see that the curves vary depending on progress in the game and the number of playouts. The curves for 500 playouts in the opening positions especially show that MC and UCT are not beneficial for estimating the win probability of human players. This suggests that we should handle the estimates of shallower and deeper nodes differently for these programs.

Fig. 7 plots the evaluation curves for GnuGo. The curves vary significantly depending on progress of the game.

D. Evaluation Curves at Strategic Positions

It is empirically known that programs based on Monte Carlo methods tend to play bad moves in strategic positions, where they need to search relatively long sequences to play good moves. Here, we discuss our examination of this empirical fact by using our method. First, we selected 12,388 positions in which some stones could be captured by a ladder (simple capturing race). Fig. 9 plots evaluation curves for

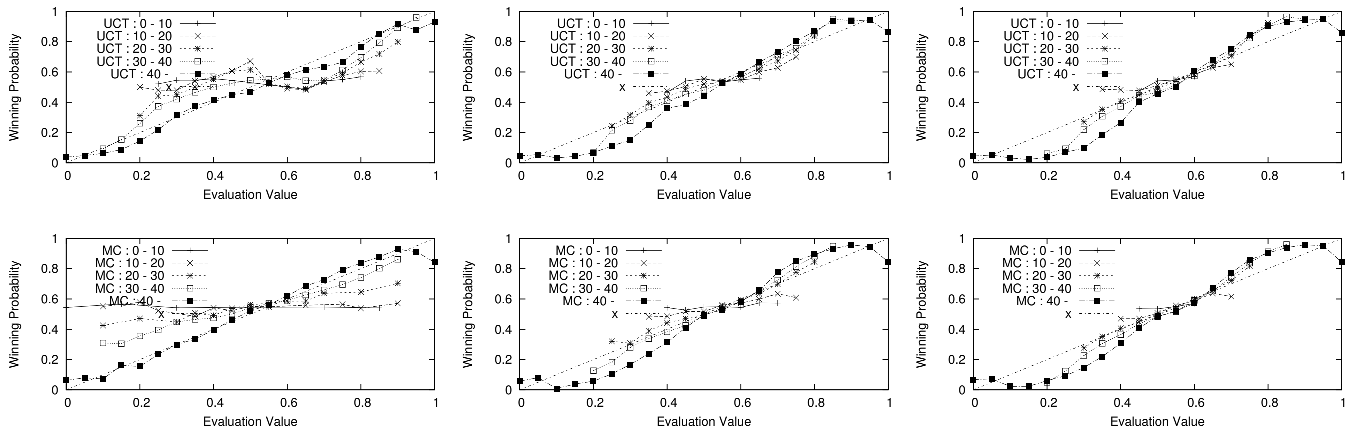
Fig. 6. UCT and MC with various move numbers. (up: UCT, low: MC, left: 500 playouts, center 5,000 playouts, and right: 50,000 playouts)

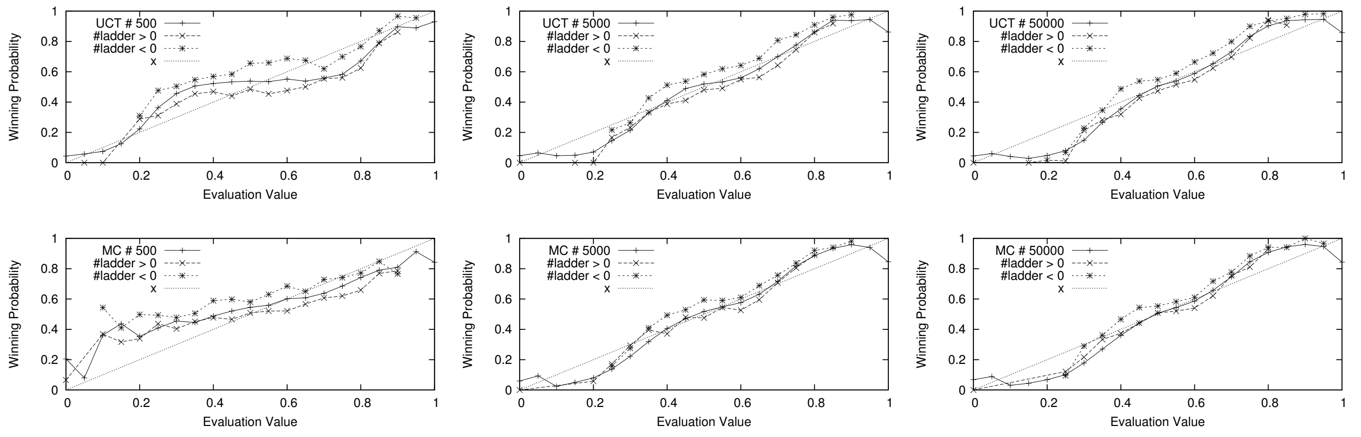
Fig. 9. UCT and MC with ladder position. (up: UCT, low: MC, left: 500 playouts, center 5,000 playouts, and right: 50,000 playouts)

UCT and MC for these positions. In the figure, "#ladder $>$ 0" means that there are black stones to be captured if the white player attacks by using a ladder, and "#ladder $<$ 0" means the same for a black player. In the figure, we can see that a split exists in the curves especially for UCT and MC in the small number of playouts, which confirms the empirical fact. Fig. 8 evaluation curves for the same positions in Fuego. The curves almost fit into one where $x > 0.5$. Therefore, we can see that this deficiency in UCT and MC is remedied in Fuego.

For graphs with split curves, if there is a white (black) stone to be captured, the winning probability of the black player computed by game records tends to be higher (lower) than that obtained by simulation. This result is consistent with a widely accepted observation that such stones may not be captured in Monte Carlo simulations.

E. Variations in Game Rules

In Go, there are some variations in rules including komi (0.5, 6.5, and 7.5) and counting rules (Japanese and Chinese). There are also some variations in the rates of players. A player with dan (d) is stronger than one with kyu (k). The highest rate for kyu players is 1 k and 40 k for the weakest.

TABLE II
WINNER OF GAME RECORDS

rate	komi	black wins	black losses	Chinese rules
(d, 6k)	6.5	680	1,320	0
(9k, 11k)		745	1,255	2
(d, 3k)	0.5	1,094	906	32
(9k, 11k)		1281	719	82

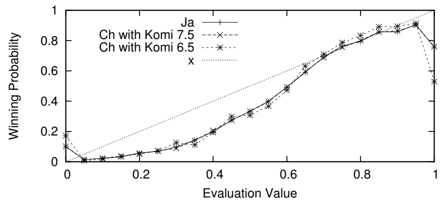
Fig. 10. Fuego (KGS, $\geq$ 6k, komi 6.5)

The number of KGS records played with komi 6.5 was less than the records played with komi 0.5, and those played with komi 7.5 were negligible. Also, the number of records played by dan players was less than that of kyu players. We collected four sets of records with different komi and players' rates as listed in Table II to conduct our experiments. In the table, the

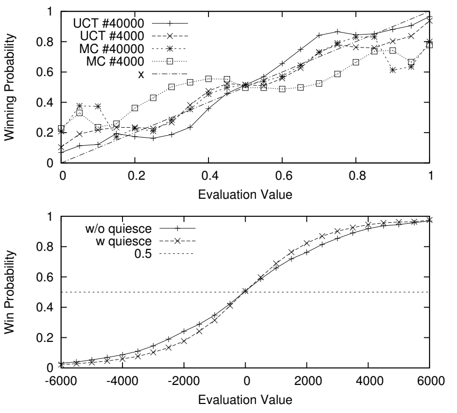

Fig. 11. Shogi (upper: UCT and lower: evaluation function with or w/o quiescence search)

number of records whose winner is a black player is indicated in the "black wins." The first set of records consists of those played by players over 6 k with komi 6.5. The second one consists of records played by players between 9 k and 11k and with komi 6.5. The third and fourth ones correspond to records played by players over 3 k and between 9 k and 11 k, with komi 0.5. We conducted our experiments for the four sets of records, but we have only presented the results of the third set in this paper. The reasons are as follows. (1) It is better to use the win probability of strong players to obtain a good evaluation. Therefore, the first and the third sets are preferable. (2) In the first set, white players win almost twice as often as black players. This introduces some bias. Actually, from the evaluation curves for Fuego computed by the first set of records plotted with '*' in Fig. 10, we can see the win probability is considerably less than 0.5 at $x = 0.5$. However, in Fig. 3 (left), the evaluation curves for Fuego computed by the third set of records takes almost 0.5 at $x = 0.5$. Consequently the deviation from 0.5 at $x = 0.5$ is caused by the imbalance of wins and losses by black players.

Let us consider the effect of the Chinese and Japanese rules. In Fig. 10, the evaluation curve with '+' plots the results for Japanese counting and komi 6.5 and the evaluation curve with '*' plots these for Chinese counting and komi 7.5. Three curves including the original one almost fit into one curve. Subsequently, we can ignore the small difference in komi and counting rules except for both ends ($x > 0.95$ or $x < 0.05$). We surmised that they were few exceptional positions where the winner differed depending on komi and counting rules, or where human players tended to miss optimal endgames.

F. Comparison of Performance in Chess, Othello, and Shogi

Finally, let us discuss the effectiveness of our methods in other games where we tried to compare how well they performed in evaluation in different games. The results are listed in Table III. In Chess, we ran Crafty (ftp://ftp.cis.uab.edu/pub/hyatt/) version 20.14 and analyzed evaluation values returned with and without quiescence search. We used about 2,000 records made available by ICCF (http://iccf.com/) for the game records except for records that resulted in a draw. The evaluation values were more accurate with quiescence search as seen in the table. It was also supported by the evaluation curves plotted in Fig. 2 because the total curve with quiescence search is monotonously increasing.

TABLE III
PERFORMANCE OF EVALUATION METHODS IN VARIOUS GAMES

game	method	ACC	FSC	LFT	ROC	APR	BEP
Chess	w quiesce	0.669	0.730	0.813	0.716	0.775	0.713
	w/o quiesce	0.653	0.715	0.790	0.689	0.751	0.702
Shogi	w quiesce (4)	0.621	0.638	0.727	0.691	0.641	0.627
	w quiesce (2)	0.611	0.629	0.706	0.674	0.629	0.618
	w/o quiesce	0.604	0.622	0.692	0.660	0.620	0.611
	UCT 40,000	0.607	0.684	0.719	0.685	0.713	0.625
	UCT 4,000	0.576	0.691	0.710	0.675	0.698	0.620
	MC 40,000	0.579	0.689	0.695	0.665	0.679	0.614
	MC 4,000	0.546	0.685	0.693	0.652	0.658	0.604
Othello		0.670	0.654	0.825	0.761	0.768	0.657

In Shogi, we used GPS Shogi[6], and used 2,000 records from the Shogi Club 24. The accuracy of the evaluation function with quiescence search with depths 0, 2, and 4, and also those of plain UCT and MC with 4,000 and 40,000 playouts were analyzed. We can see from the table that evaluations with deeper search and with larger numbers of playouts consistently achieved better accuracy. The main difference between evaluation methods in GPS Shogi was that evaluation with quiescence search was the best followed by UCT and, MC. These results were also supported by the evaluation curves plotted in Fig. 11.

In Othello, we used Zebra[7] and 2,000 records played at GGS[8]. More than one third of the records seemed to have been played by computer programs. The evaluation function in Othello achieved greater precision in the accuracy metrics, without any search, than the all other methods in Chess and Shogi. This excellent accuracy can be partially explained by the existence of the records played by computer programs if it is relatively easy for computer programs to predict the win/lose of records played by other computer programs. However, only Fuego listed in Table I achieved greater accuracy, but with much more computational effort to evaluate the positions. Consequently, the accuracy in Othello is still surprisingly high and well beyond that explained by the records used. The outstanding accuracy in Othello is consistent with the history of computer programs that reached grandmaster level in Othello earlier than other games.

V. CONCLUDING REMARKS

We presented a means of measuring the performance of methods involving Monte Carlo tree search by using the relationship between the win probability obtained by simulations and that observed in game records. By plotting evaluation curves for Monte Carlo tree search methods, we could see

[6]http://gps.tanaka.ecc.u-tokyo.ac.jp/gpsshogi/ (We used gpsshogi rev.1363 and osl rev.3203 for UCT, gpsshogi rev.1117 and osl rev.2602 for the others.)

[7]http://radagast.se/othello/

[8]http://www.cs.ualberta.ca /˜mburo/GGS/

they had various characteristics. If the curve differed from the $y = x$ line, then the win probability estimated by the simulations was poor. By plotting the curves for various search methods, we could see which methods were better than others. By plotting the curves for various numbers of playouts, we could assess what effect the numbers of playouts had. By plotting the curves for various phases of progress in the games, we could see how effective the simulations were at various stages of the game. By plotting the curves for positions with/without certain conditions, we could see how the conditions affected the effectiveness of the simulations. We demonstrated, as an example, that many methods involve difficulties in evaluating positions with stones in threats of a ladder proving the experience that Monte Carlo programs are relatively weak in such strategic positions.

We also introduced numerical metrics ACC, FSC, LFT, ROC, PRS, and BEP to evaluate the performance of search methods using game records. Our experiments revealed that ACC is quite close to our empirical understanding of the performance of various search methods. We can automatically compare various methods by using such metrics.

Utilizing these metrics to make strong programs is a fascinating topic of research. The first step toward this direction is to measure and compare the performance of individual enhancements in UCT such as RAVE and patterns. Once split curves are found in Monte Carlo tree search methods, developing enhancements to remedy problems is the next step. We can expect this remedy to improve search methods because Chess and Shogi programs become stronger by modifying evaluation functions to reduce the split in evaluation curves [5].

Acknowledgments

We would like to thank the referees for their helpful comments.

References

[1] J. Schaeffer, "The games computers (and people) play," *Advances in Computers*, vol. 50, pp. 189–266, 2000.

[2] M. Müller, "Computer go," *Artificial Intelligence*, vol. 134, no. 1–2, pp. 145–179, Jan. 2002.

[3] S. Gelly and D. Silver, "Achieving master level play in 9×9 computer go," in *Proceedings of AAAI*, 2008, pp. 1537–1540.

[4] L. Kocsis and C. Szepesvari, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006*, vol. 4212. Springer, 2006, pp. 282–293.

[5] S. Takeuchi, T. Kaneko, K. Yamaguchi, and S. Kawai, "Visualization and adjustment of evaluation functions based on evaluation values and win probability," in *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI-2007)*, 2007, pp. 858–863. [Online]. Available: http://www.graco.c.u-tokyo.ac.jp/%7ekaneko/papers/AAAI13TakeuchiS.pdf

[6] M. L. Ginsberg, "GIB: steps toward an expert-level bridge-playing program," in *Sixteenth International Joint Conference on Artificial Intelligence*, 1999, pp. 584–589.

[7] B. Sheppard, "World-championship-caliber Scrabble," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 241–275, 2002.

[8] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron, "The challenge of poker," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 201–240, 2002.

[9] B. Abramson, "Expected-outcome: A general model of static evaluation." *IEEE Trans. Pattern Analysis and Mach. Intell.*, vol. 12, no. 2, pp. 182–193, 1990.

[10] B. Brügmann, "Monte Carlo Go," Physics Department, Syracuse University, Tech. Rep., 1993.

[11] B. Bouzy and B. Helmstetter, "Monte Carlo Go developments," in *Advances in Computer Games. Many Games, Many Challenges*. Kluwer Academic Publishers, 2003, pp. 159–174.

[12] H. Yoshimoto, K. Yoshizoe, T. Kaneko, A. Kishimoto, and K. Taura, "Monte carlo go has a way to go," in *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006)*, 2006, pp. 1070–1075.

[13] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *Computers and Games*, ser. Lecture Notes in Computer Science, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., vol. 4630. Springer, 2006, pp. 72–83.

[14] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of uct with patterns in monte-carlo go," INRIA, Tech. Rep. RR-6062, 2006.

[15] Chaslot, M. H. Winands, I. Szita, and J. H. van den Herik, "Parameter tuning by the cross-entropy method," in *Proceedings of EWRL*. Springer, 2008. [Online]. Available: http://www.cs.unimaas.nl/g.chaslot/papers%5Cewrl.pdf

[16] R. Coulom, "Computing elo ratings of move patterns in the game of go," in *Computer Games Workshop*, Amsterdam / The Netherlands, 2007.

[17] D. Silver, R. Sutton, and M. Müller, "Sample-based learning and search with permanent and transient memories," in *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, A. McCallum and S. Roweis, Eds. Omnipress, 2008, pp. 968–975.

[18] N. Araki, "Move prediction and strength in monte-carlo go program," Master Thesis, the Graduate School the University of Tokyo, 2008. [Online]. Available: http://ark.qp.land.to/main.pdf

[19] M. Buro, "Improving heuristic mini-max search by supervised learning," *Artificial Intelligence*, vol. 134, no. 1–2, pp. 85–99, Jan. 2002.

[20] J. van Rijswijck, "Learning from perfection: A data mining approach to evaluation function learning in awari," in *Computer and Games*, ser. LNCS, T. A. Marsland and I. Frank, Eds., no. 2063. Hamamatsu, Japan: Springer-Verlag, Oct. 2001, pp. 115–132.

[21] D. Gomboc, M. Buro, and T. A. Marsland, "Tuning evaluation functions by maximizing concordance," *Theor. Comput. Sci.*, vol. 349, no. 2, pp. 202–229, 2005.

[22] M. Vuk and T. Curk, "Roc curve, lift chart and calibration plot," *Metodološki zvezki*, vol. 3, no. 1, pp. 89–108, 2006.

[23] T. Fawcett, "An introduction to roc analysis," *Pattern Recogn. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.

[24] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms using different performance metrics," in *ICML '06: Proceedings of the twenty-third International Conference on Machine Learning*, 2006, pp. 161–168.