Evaluation of Game Tree Search Methods by Game Records

Shogo Takeuchi, Tomoyuki Kaneko, and Kazunori Yamaguchi

***Abstract*—This paper presents a method of evaluating game tree search methods including standard min–max search with heuristic evaluation functions and Monte Carlo tree search, which recently achieved drastic improvements in the strength of *Computer Go* programs. The basic idea of this paper is to use an averaged win probability of positions having similar evaluation values. Accuracy measures of evaluation values with respect to win probabilities can be used to assess the performance of game tree search methods. A plot of win probabilities against evaluation values should have consistency and monotonicity if the evaluation values are produced by a good game tree search method. By inspecting whether the plot has the properties for some subset of positions, we can detect specific deficiencies in the game tree search method. We applied our method to *Go*, *Shogi*, and *Chess*, and by comparing the results with empirical understanding of the performance of various game tree search methods and with the results of self-plays, we show that our method is efficient and effective.**

***Index Terms*—Evaluation function, games, game tree search, Monte Carlo tree search.**

I. Introduction

TWO approaches are used in game tree searches. One is game tree search with evaluation functions, and the other is Monte Carlo tree search. The former has a long history and has been widely used in *Chess* and many other games. An evaluation function with the former approach estimates the quality of a given position. 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 is difficult for computers and humans to find an appropriate set of features.

In the latter approach, recent improvements in methods involving Monte Carlo tree search [1] have made it possible to create strong *Computer Go* programs [2] such as MoGo [3], CrazyStone [4], and Fuego. Even though the methods have a sound theoretical background, the effectiveness of many enhancements such as patterns, rapid action value estimation (RAVE), and progressive widening, has not been studied yet. So, we need a method to measure the quality of Monte Carlo simulation.

The method proposed in this paper can be applied to both approaches. In order to make our explanation applicable to both of them, we refer to them as *game tree searches*, which estimate evaluation values of a position irrespective of the method of estimation throughout this paper. In Monte Carlo simulation, evaluation values are the win probabilities estimated by the simulation.

In this paper, we propose a novel method for an evaluation of the accuracy of evaluation values produced by game tree searches. The basic idea of our method is as follows. We associate a position with their win/loss/draw from the game records or by search. Then, we approximate the win probability of evaluation value by the average of wins of the positions having the evaluation value within some tolerance. Using this relationship between evaluation values and win probabilities, we can perform various assessments of the game tree searches.

We call a plot of win probabilities against evaluation values an *evaluation curve*. Evaluation curves of a good game tree search should have consistency and monotonicity. By visually inspecting a plot as to whether it has the properties, we can determine whether the game tree search is good or poor. A plot allows a local inspection such as that the game tree search is poor when win probabilities are less than 50%. Also, from evaluation curves for positions under some condition such as King is safe or not, we can determine whether the game tree search is affected by the condition.

By viewing evaluation values as estimates of the win probabilities, we introduce several accuracy metrics. From these metrics, we can compare game tree searches numerically.

We confirmed that evaluation curves and the accuracy metrics are useful by numerous experiments on *Go*, *Chess*, *Othello*, and *Shogi*.

This paper is structured as follows. First, related work is reviewed in Section II. Then, our method of evaluating game tree search is presented in Section III, followed by experimental results in Section IV. How we applied our method to new games is discussed in Section V. Finally, conclusions and future work are discussed in Section VI.

II. Related Work

A. Accuracy of Game Tree Search

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

If more information is available, we can directly evaluate evaluation values. For example, if theoretically correct evaluation values are available in a database or are found by an exhaustive search, the errors in evaluation values can be directly calculated. Examples are endgames in *Othello* [5] and *Awari* [6].

Manuscript received April 15, 2010; revised July 12, 2010; accepted November 29, 2010. Date of publication December 23, 2010; date of current version January 19, 2011. This work was supported in part by Grant-in-Aid for JSPS Fellows 21.10594.

The authors are with the Graduate School of Arts and Sciences, University of Tokyo, Tokyo, Japan (e-mail: takeuchi@graco.c.u-tokyo.ac.jp; kaneko@graco.c.u-tokyo.ac.jp; yamaguch@graco.c.u-tokyo.ac.jp).

Digital Object Identifier 10.1109/TCIAIG.2010.2102022

However, the domains where such information is available are limited. For another example, if the preference of human players is available, we can evaluate game tree search methods from how an evaluation value for each position agrees on the preference of human players [7]. The applicability of this method is limited to domains in which such a preference is available. Our method does not require such a preference.

B. Learning of Evaluation Functions

The purpose of the evaluation of evaluation functions is to improve evaluation functions. In this sense, research on evaluation of evaluation functions is related to research on learning of evaluation functions.

Much research has been devoted to the learning of evaluation functions in game programming since Samuel's seminal work on *Checkers* [8]. Supervised learning can be effectively used to adjust weights, when appropriately labeled training positions are available. Supervised learning in *Othello* produced one of the strongest programs available then [9]. However, no evaluation functions have successfully been trained in *Chess* and *Shogi* by directly applying supervised learning due to the difficulty of obtaining labeled positions.

There is a method based on the correlation of preferences for positions in *Chess* [10]. However, this requires many positions to be assessed by grandmasters to determine which of the two positions are preferred. Thus, its application is limited to domains in which such assessments can be carried out. Our method requires no positions to be labeled.

Temporal difference learning is another approach to adjust weights. It was successful with *Backgammon* [11]. Learning variants have also been applied to *Chess* [12]. However, temporal difference learning has not been adopted in top-level programs for deterministic games. This method involves much computational cost because it has to update weights by playing numerous games. Our method requires no need for plays and is computationally efficient.

C. Monte Carlo Tree Search

In imperfect-information games including *Bridge* [13], *Scrabble* [14], and *Poker* [15], sampling-based approaches have been widely used. Abramson [16] presented a method of using random sampling for evaluation. Applying the Monte Carlo method to *Go* was first introduced by Brügmann [17], and was later studied by Bouzy and Helmstetter [18]. Monte Carlo *Go* utilizes the results of random sampling in evaluating positions, instead of hand-coded evaluation functions of heuristic search.

In the classical model, it performs a one-ply search and computes an "expected score" for each node. 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. We call these random games *playouts*. A fixed number of playouts is played at each leaf. The law of diminishing returns with additional playouts has been confirmed for this classical model [19]. 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. It then selects the move with the highest score.

Many enhancements have been proposed to this classical model. In some enhancements, the win probability is used as the expected score of a position instead of the average of the final scores as used in the classical model. We will elaborate on this enhancement later.

Currently, the most effective enhancement in recent programs is the recursive extension of nodes and intensive playouts on effective moves [20], [1].

A Monte Carlo method combined with tree search is called a *Monte Carlo tree search* in general. Upper confidence bounds applied to trees (UCT) [1] is a popular Monte Carlo tree search method. It is based on the theory of the multiarmed bandit problem. In a UCT tree, Monte Carlo simulation is conducted at leaves. UCT recursively extends the most effective node in a best-first manner where the effectiveness of a node is estimated by the win probability and exploration term of the node. There are some variations in the use of variance in UCT [21], [22], from which we explain UCB1 and UCB1-tuned in the following.

Let p_i be the win probability in s_i playouts undertaken for the ith move at node a, and let n be the total playouts carried out at the descendants of node a. UCB1 extends move i, which maximizes

$$p_i + \sqrt{\frac{2\log n}{s_i}}. \tag{1}$$

UCB1-tuned is an improved version of UCB1. UCB1-tuned extends move i, which maximizes

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

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 game records [23] or dynamically analyzing games during play [24]. The relationship between the strength of programs and the quality of patterns used to select moves in playouts has been reported to be unclear [25], where quality means the accuracy with which moves are predicted in game records.

It should be noted that the accuracy of the win probability estimated by the Monte Carlo simulation, where accuracy is with respect to the win probability in actual game playing, has not yet been assessed, except for our preliminary work [26].

III. Our Methods

We first introduce a method to approximate the win probability for evaluation values from game records in Section III-A. Then, we introduce classification performance metrics applied to the estimation of win/loss by evaluation values in Section III-B. A plot of the relationship called evaluation curves is introduced in Section III-C. Then, accuracy metrics on evaluation values over the win probabilities are introduced in Section III-D. Finally, the expected value of win is introduced in Section III-E.

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

A. Win Probability in Game Records

The key to our evaluation of game tree search methods is the relationship between win probabilities and evaluation values. For Monte Carlo tree search, this is the comparison between the simulated win probability and the win probability from the game records. In our framework, the former is called an evaluation value, and the latter a win probability. Now, assume that there are numerous game records R that contain positions. We define the win probability as a function of evaluation value v and R as

$$\text{win probability}(v, R) = \frac{|B_v(R)|}{|B_v(R)| + |W_v(R)|} \tag{3}$$

where

$$\begin{aligned} P_v(R) &= \left\{ p \in R \mid v - \frac{\delta}{2} \le \text{eval}(p) < v + \frac{\delta}{2} \right\} \\ B_v(R) &= \{ p \in P_v(R) \mid \text{winner } (p) \text{ is the black player} \} \\ W_v(R) &= \{ p \in P_v(R) \mid \text{winner } (p) \text{ is the white player} \}. \end{aligned} \tag{4}$$

Here, p is a position in R and δ is a nonnegative tolerance of evaluation values. 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. Because 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 if the exhaustive search cannot determine the winner. This worked sufficiently well in our experiment. Finally, we aggregate the numbers of wins $|B_v|$ and losses $|W_v|$ for each interval $[v-(\delta)/(2), v+(\delta)/(2))$, and calculate the fraction using (3). This calculation was first proposed in our previous paper [27] for assessing heuristic evaluation functions, where we used the value of evaluation functions as $\text{eval}(p)$ in (4). For Monte Carlo tree search method, we use the results of the Monte Carlo simulations as $\text{eval}(p)$.

B. Application of Classification Performance Metrics

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

From the nine metrics discussed by Caruana and Niculescu-Mizil [28], we selected six metrics having the compatible output range with the game tree search. It is because the rest of the metrics are not suitable for analyzing some search methods, such as Monte Carlo score and heuristic evaluation functions. First, let us introduce some basic definitions that will be used later. Let a_i be the theoretical win/loss 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 classical 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 $b_i = 1(v_i \ge t)$, $b_i = 0(v_i < t)$. For v_i in $[0, 1]$, 0.5 is used as t. Let TP denote true positives, FP denote false positives, TN denote true negatives, and FN denote false negatives. They are the sets: $\text{TP} = \{i \mid a_i = 1, b_i = 1\}$, $\text{FP} = \{i \mid a_i = 0, b_i = 1\}$, $\text{TN} = \{i \mid a_i = 0, b_i = 0\}$, and $\text{FN} = \{i \mid a_i = 1, b_i = 0\}$. Then, precision and recall are defined as follows:

$$\text{precision} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FP}|} \quad \text{recall} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FN}|}.$$

Precision is a fraction of true positives over classified positives, and recall is a fraction of classified positives over true positives and false negatives. There is usually a tradeoff between precision and recall.

Now, let us introduce the six metrics.

1) Accuracy (ACC): Accuracy is defined as
$$\frac{|\text{TP}| + |\text{TN}|}{|\text{total}|}.$$
Total is the set of all records. The accuracy ranges from 1.0 to 0.5 by negating output if necessary.
2) F-score (FSC): F-score is defined as
$$\frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}.$$
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_i s. Formally, lift is defined as
$$\frac{|\text{TP in the top } T\% \text{ samples}|}{|\text{top } T\% \text{ samples}|}.$$
We used $T\% = 25\%$ as in [28].
4) Area under ROC curve (ROC): ROC curve is a plot of the fraction of true positives along the vertical axis and that of false positives along the horizontal axis for various threshold t. The area under the ROC curve ranges from 1.0 to 0.5 by negating output if necessary. See [29] for details.
5) Average precision (APR): Average precision is the averaged precisions for thresholds whose recalls are $0.0, 0.1, \ldots, 1.0$.
6) Precision/recall break even point (BEP): BEP is the precision for the threshold whose 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 0.0 if these are all negatives. Thus, the metrics over different test cases should be carefully compared. Evaluation curves, which will be introduced in Section III-C, for different sets of positions, are useful for detecting this kind of problem.

C. 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*. The evaluation curves of good game tree searches must be monotonically increasing. So, monotonicity is required. However, actual evaluation curves are not always monotonically increasing.

The monotonicity is not sufficient to ensure that the game tree search method is sound. Suppose that we have evaluation curves

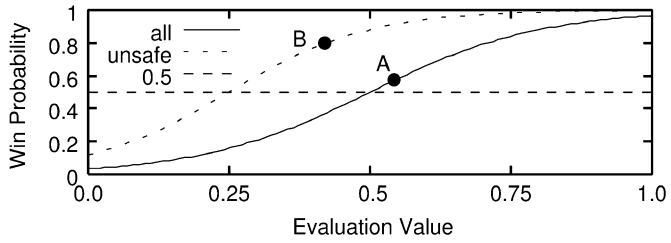

Fig. 1. Example of poor game tree search.

for all positions and only for positions satisfying a certain condition as shown in Fig. 1. For the position X having the evaluation value at A and position Y having the evaluation value at B, the game tree search estimates X better than Y, while Y has a larger win probability. As demonstrated by this example, evaluation curves for positions satisfying different conditions should overlap. If they do, we call the evaluation curve consistent. We call an evaluation curve for all the positions a *total curve*, and an evaluation curve for positions satisfying some condition a *conditioned curve*. If a total curve and a conditioned curve do not overlap, we say that they *split*. How well the evaluation method is working for some conditions can be observed by how they split.

D. Kendall's τ Rank Correlation Coefficient

We employ Kendall's τ as a measure of correlation of win probabilities and evaluation values. Kendall's τ is a measure of rank correlation defined as follows:

$$\tau = \frac{n_c - n_d}{\frac{1}{2}n(n-1)}$$

where n_c is the number of pairs of the same order[2] in both ranks, and n_d is the number of pairs of the different order. In measuring the correlation of win probabilities and evaluation values, n_c is the number of position pairs whose orders of evaluation values and win probabilities agree, and n_d is that of position pairs whose orders disagree. If the orders of evaluation values and win probabilities completely agree, $\tau = 1$, and if the orders of them completely disagree, $\tau = -1$.

E. Expected Value of Values

In game tree search, we can prune unimportant moves, if an evaluation value of a move is sufficiently smaller than that of the best move. So, it is desirable that the variance of evaluation value be large, but the variance of evaluation values is not considered in the monotonicity and consistency discussed in Section III-C. We need another metric to assess the variance.

We also need to assess the reliability of evaluation curve. If evaluation values concentrate on the center around 0.0 (in evaluation functions) or 0.5 (in Monte Carlo tree search methods), there are only a few positions at either end of the evaluation curve. This reduces the reliability of the evaluation curve. So, we need a metric to assess the reliability of evaluation curves.

For these purposes, we use an expected value of evaluation values as a metric. Let v_i be the evaluation value for position i, and r_i be win/loss represented by 1/0. For an ideal evaluation function, $v_i = 1$ for $r_i = 1$ (win), and $v_i = 0$ for $r_i = 0$ (loss). Then, we calculate the expected value of v_i for $r_i = 1$ as

$$E(v)_{r=1} = \frac{\sum_{\{i|r_i=1\}} v_i}{\sum_{\{i|r_i=1\}} 1}.$$

Similarly, we calculate it for $r_i = 0$ as

$$E(v)_{r=0} = \frac{\sum_{\{i|r_i=0\}} v_i}{\sum_{\{i|r_i=0\}} 1}.$$

We use the difference of them as the metric defined by

$$E(v) = |E(v)_{r=1} - E(v)_{r=0}|.$$

It is desirable that $E(v)$ be large because the variance of evaluation values is large then. Note that v_i and r_i are available from the data used in Sections III-C and III-D, and $E(v)$ can be obtained with no extra cost.

IV. Experimental Results

In this section, we first show the results of experiments with Monte Carlo tree search methods. Then, we also show that our methods were successfully applied to the analysis of evaluation functions for min–max search methods.

A. Evaluation of Monte Carlo Tree Search Methods in Go

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[3] version 0.4.1 as an enhanced UCT program with various enhancements including patterns. Fuego is a strong program with a rating of about 2500 in the 9×9 version of *Internet Go* server (CGOS).[4] Evaluation was carried out by using the "genmove" gtp command and we used the win probability of the best move.
- UCT: We used the implementation in libego (version 0.116, on August 1, 2008)[5] as a plain UCT program. Its rating is about 1800 in CGOS (9×9, on August 1, 2008). 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: MC-Score was a modified version of MC, which computed the averaged leaf scores instead of the win probability. The reason MC-Score was used is to

[2]A pair whose evaluation value or win probability ties is counted in n_c in our experiment for simplicity.

[3]http://fuego.sourceforge.net/

[4]http://cgos.boardspace.net/9x9/

[5]http://github.com/lukaszlew/libego

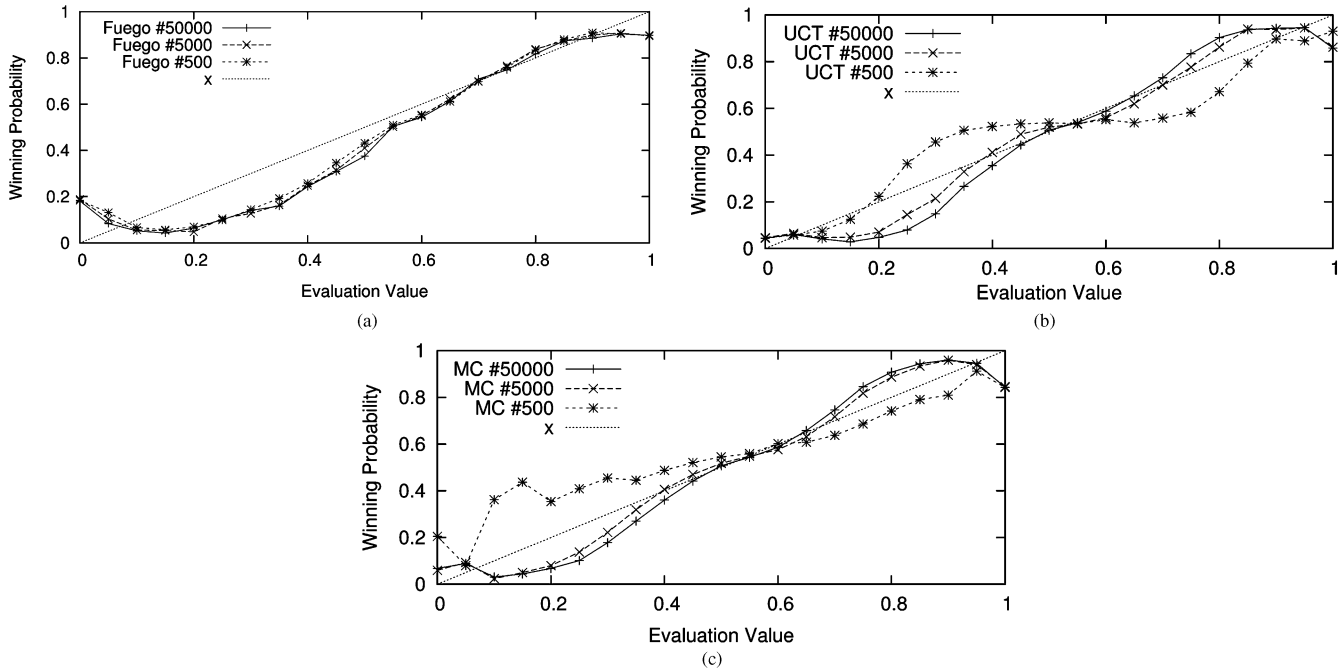

Fig. 2. Evaluation curves for (a) Fuego, (b) UCT, and (c) MC.

assess the quality of simulations in the classical model explained in Section II-C.

- GnuGo: We used GnuGo[6] (version 3.7.12, on August 1, 2008) as a traditional program. The evaluation values of the root node and that of the best move are identical in GnuGo.

All programs were run with the Chinese counting rule because some did not support the 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, players' rating, and handicap. We mainly used the records under the condition of komi 0.5, a rating that is over 3 k, and no handicap, because most records were played in this configuration, as summarized in Table VII. We also analyzed and discussed other configurations.

There were 2000 records that include 111 946 positions. In our experiments, Monte Carlo simulations took the most of the time and our analysis took only a few minutes. The run time of simulations depends on the number of games and the number of playouts. For Fuego with 50 000 playouts (the most time-consuming experiment), it took a day. These experiments were performed on an Intel Xeon 3.0-GHz processor.

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. 2(a) 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' win in the game records. A deviation at both ends ($x > 0.95$ or $x < 0.05$) appears in almost all other evaluation curves and the reason will be discussed in Section V-A2.

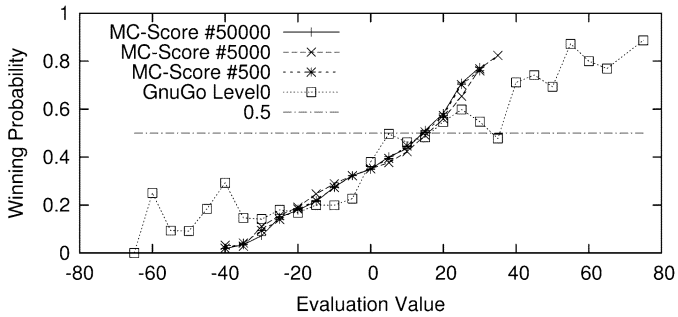

Fig. 3. Evaluation curves for MC-Score and GnuGo.

Fig. 2(b) and (c) plots the evaluation curves for UCT and MC with various numbers of playouts. In both evaluation curves, the curves for 500 playouts plotted with "$*$" are different from the other curves, while the evaluation curves for 5000 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 (1) and (2). This would be an interesting topic for further research.

Fig. 3 plots evaluation curves for MC-Score and GnuGo. The horizontal axis of this evaluation curve is for the score in the range of $[-81, +81]$. Surprisingly, the curves for MC-Score are almost the same for all sample sizes. This might be the effect of diminishing of returns reported by Yoshimoto *et al.* [19]. 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.

[6]http://www.gnu.org/software/gnugo/gnugo.html

TABLE I
PERFORMANCE OF EVALUATION METHODS

method	#playouts	ACC	FSC	LFT	ROC	APR	BEP
Fuego	50,000	0.714	0.775	1.781	0.803	0.810	0.775
	5,000	0.710	0.771	1.792	0.798	0.807	0.768
	500	0.704	0.764	1.813	0.789	0.800	0.758
UCT	50,000	0.631	0.665	2.017	0.704	0.708	0.637
	5,000	0.608	0.682	1.783	0.690	0.697	0.627
	500	0.570	0.688	1.475	0.677	0.678	0.619
MC	50,000	0.617	0.578	2.675	0.680	0.678	0.618
	5,000	0.611	0.578	2.593	0.671	0.670	0.614
	500	0.577	0.533	2.591	0.614	0.601	0.579
MC-Score	50,000	0.574	0.534	2.551	0.611	0.611	0.577
	5,000	0.569	0.539	2.438	0.604	0.605	0.571
	500	0.552	0.537	2.232	0.574	0.568	0.552
GnuGo	level 4	0.653	0.714	1.509	0.691	0.651	0.643
	2	0.650	0.711	1.510	0.688	0.648	0.642
	0	0.647	0.709	1.509	0.683	0.642	0.641

TABLE II
KENDALL'S τ AND EXPECTED VALUE OF EVALUATION VALUE: *GO*

methods	#playouts	τ	$E(v)$
Fuego	50,000	0.621	0.183
	5,000	0.611	0.176
	500	0.591	0.168
UCT	50,000	0.432	0.101
	5,000	0.400	0.093
	500	0.367	0.087
MC	50,000	0.384	0.088
	5,000	0.367	0.088
	500	0.273	0.081
MC-Score	50,000	0.231	2.493
	5,000	0.217	2.451
	500	0.154	2.492
GnuGo	level 4	0.396	3.797
	2	0.389	3.660
	0	0.380	3.556

Let us now present the performance metrics in Table I, which were described in Section II-B. First, let us discuss the results on the accuracy (ACC) metric. Here, we can assume that the accuracy ranged from 1.0 to 0.5, because if accuracy is below 0.5, we can obtain a better result by negating the estimates. We can see from the table that UCT, MC, and 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 in higher levels are stronger. Fuego was the best program followed by GnuGo, UCT, MC, and 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 ROC, APR, and BEP were similar to that of ACC. The results of LFT and FSC were also similar. However, it is unnatural that FSC for UCT and LFT for Fuego were reverse orders, and in FSC for MC-Score, 5000 playouts was the best result. FSC and LFT are not recommended from the results of this experiment.

We calculated the expected value of evaluation values. The results are summarized in Table II. We can see that programs with a larger number of playouts or in higher level achieved larger expected value. When the number of playouts is fixed, Fuego achieved the largest expected value, and UCT achieved the second largest one. These results seemed plausible.

The evaluation value of Fuego, UCT, and MC ranges from 0 to 1, and that of MC-Score and GnuGo ranges from -81 to 81. We cannot compare the expected value of the former with that of the latter because the range of the expected value depends on the range of the evaluation value.

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. 4 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. 5 plots the evaluation curves for MC and UCT for sets of varying move numbers. We can see that the curves vary depending on the progress of the game and the number of playouts. The curves for 500 playouts in the opening positions especially show that MC and UCT are not useful for estimating the win probability calculated from human players' records. This suggests that we should handle the estimates of shallower and deeper nodes differently for these programs.

Fig. 6 plots the evaluation curves for GnuGo. The curves split significantly depending on the progress of the game.

It is empirically known that programs based on Monte Carlo methods tend to play bad moves in tactical 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. 8 plots evaluation curves for UCT and MC for these positions. In the figure, "$\#\text{ladder} > 0$" means that there are black stones to be captured if the white player attacks by using a ladder, and "$\#\text{ladder} < 0$" means the same for a black player. In the figure, we can see that a split exists in the evaluation curves especially for UCT and MC in the small number of playouts,

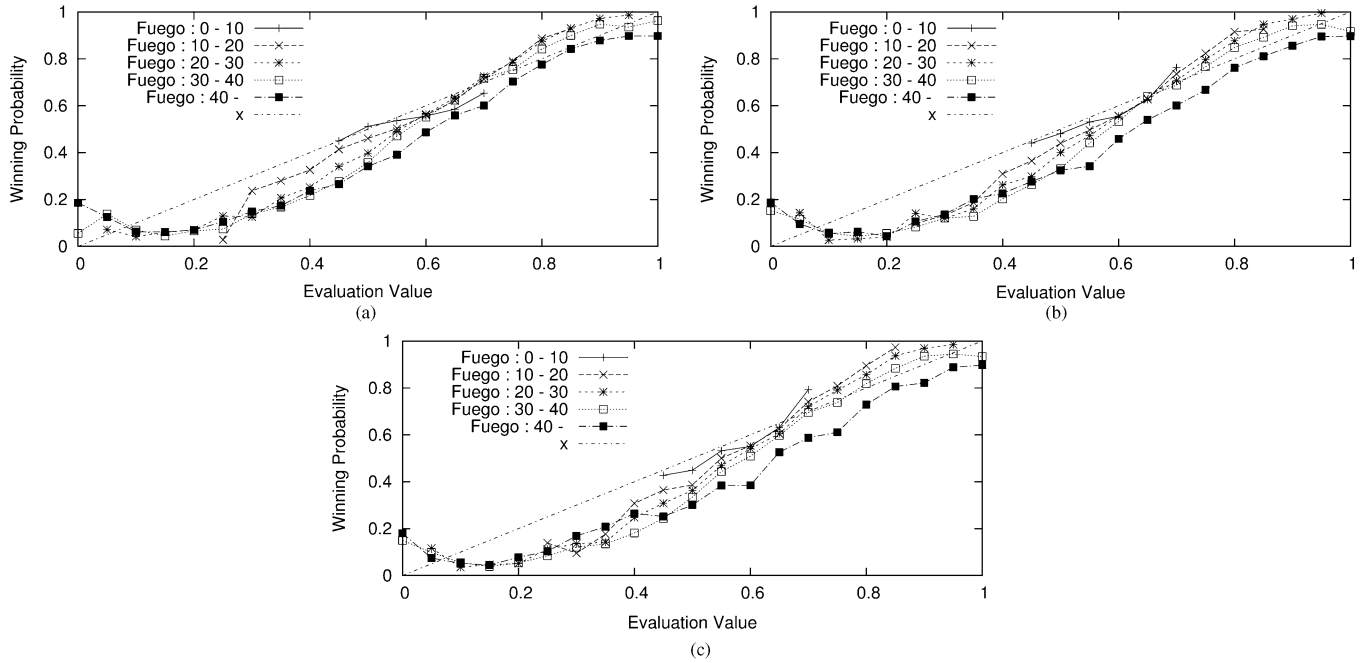

Fig. 4. Fuego with various move numbers: (a) 500 playouts, (b) 5000 playouts, and (c) 50 000 playouts.

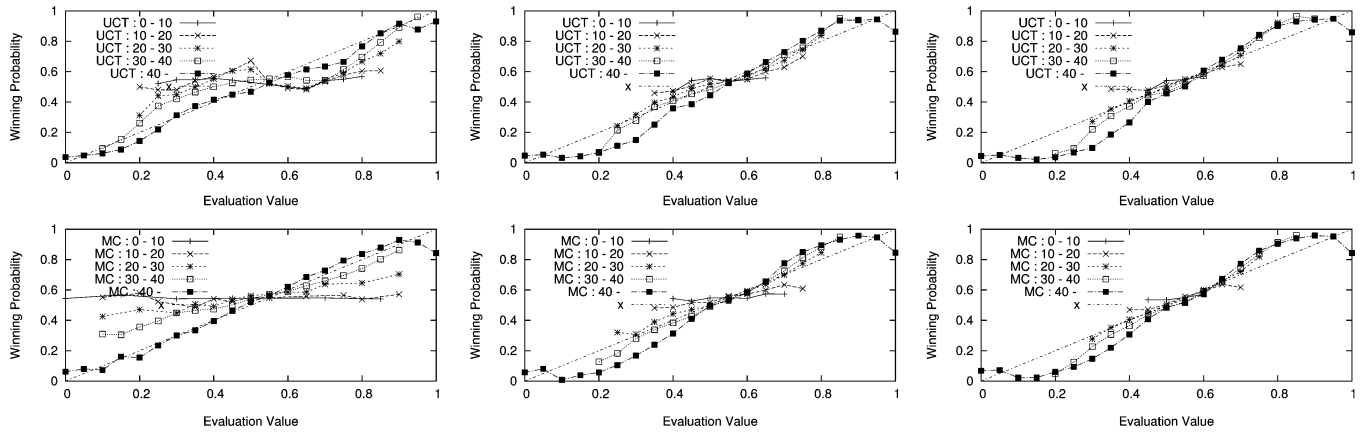

Fig. 5. UCT and MC with various move numbers: (top) UCT and (bottom) MC. (Left) 500 playouts. (Center) 5000 playouts. (Right) 50 000 playouts.

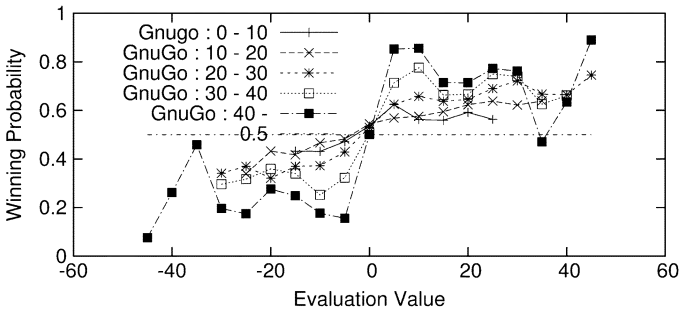

Fig. 6. Gnu Go (level 0) with various move numbers.

which confirms the empirical fact. Fig. 7 plots 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 split evaluation curves, if there is a white (black) stone to be captured, the win 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.

It is said that the Monte Carlo tree search methods work poorly at complex positions such as tactical positions than other positions. The Monte Carlo tree search methods, which make only a depth-one search, failed to evaluate positions correctly because deep search is required in order to get a good move at tactical positions. In our experiment, we selected 2218 tactical positions involving Ko. Evaluation curves of Fuego, UCT, MC, MC-Score, and GnuGo are shown in Figs. 9 and 10. The evaluation curves of these programs split and they fail to handle tactical positions of Ko. This agrees on the aforementioned experimental knowledge. The evaluation curve of Fuego splits only in the range that an evaluation value ≥ 0.5 and relatively better.

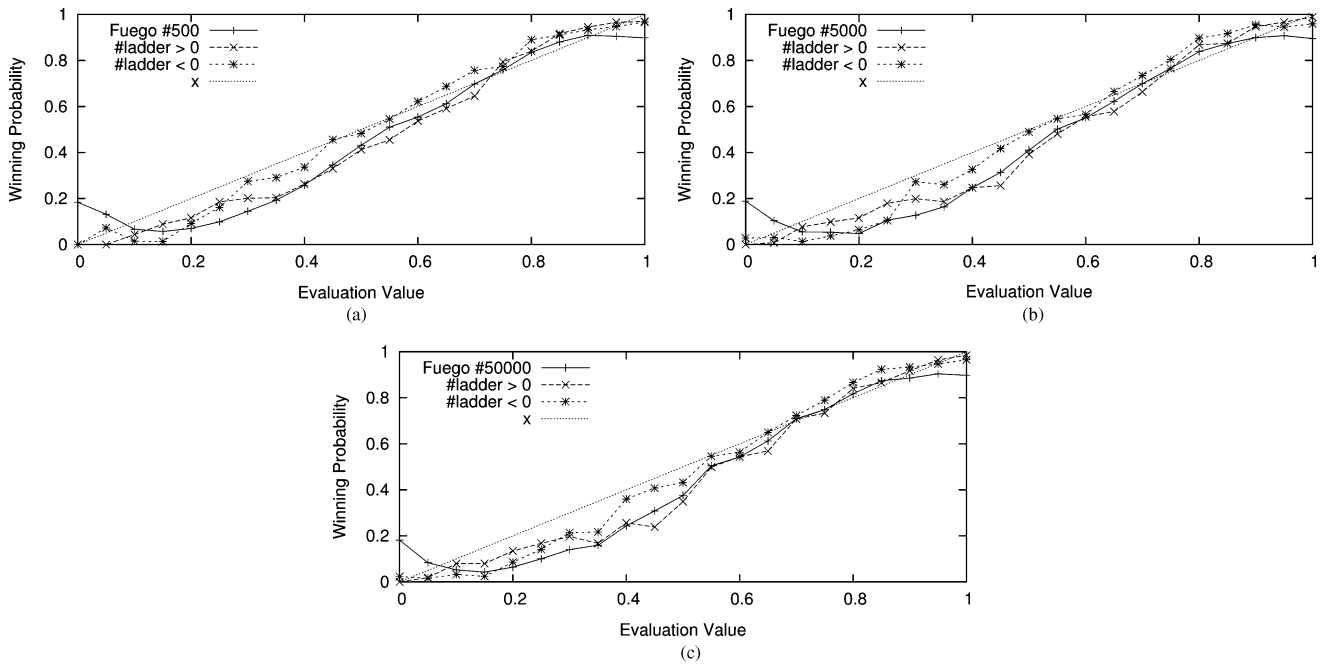

Fig. 7. Fuego at ladder positions: (a) 500 playouts; (b) 5000 playouts; and (c) 50 000 playouts.

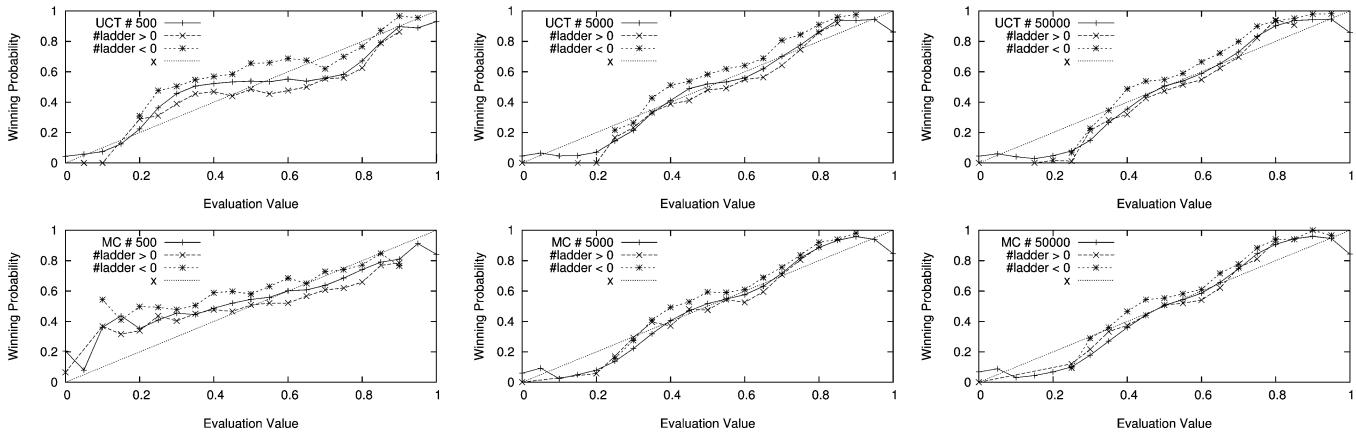

Fig. 8. UCT and MC at ladder positions: (top) UCT and (bottom) MC. (Left) 500 playouts. (Center) 5000 playouts. (Right) 50 000 playouts.

This observation agrees on the fact that Fuego is the strongest of them. The evaluation curve of MC-Score is poorer especially when the number of playouts is small.

B. Evaluation of Evaluation Functions

Here, we show the analysis of evaluation functions in *Chess* and in *Shogi*. Since parameters in evaluation functions have a direct effect on the results of position evaluation, we also show how our methods visualize the improvements of the parameters in evaluation functions.

1) Chess: We worked with Crafty[7] (version 20.14, on November 29, 2006). We used 45 955 records made available by the International Correspondence Chess Federation (ICCF[8]) as the game records. Those games were played from 1988 to 2006. It took a few hours to obtain evaluation values on an AMD Opteron 2.2-GHz processor.

a) Draw: In *Chess*, games often end in draws. In order to see how draws affect evaluation curves, we plot the evaluation curve using draws as 0.5 win. Fig. 12 shows that draws have little effect on evaluation curves. So, we did not use records of draws in our experiments in order to avoid complications with determining the win probability.

b) Quiescence Search: Most programs in various games including *Chess* use quiescence searches because evaluation values are unreliable for tactical positions. We used evaluation values as in game tree searches for the leaves of principal variations obtained by a quiescence search to draw the evaluation curve labeled "with QS" in Fig. 11. Also, we used evaluation values without a quiescence search to draw the evaluation

[7]ftp://ftp.cis.uab.edu/pub/hyatt/

[8]http://www.iccf.com/content/index.php

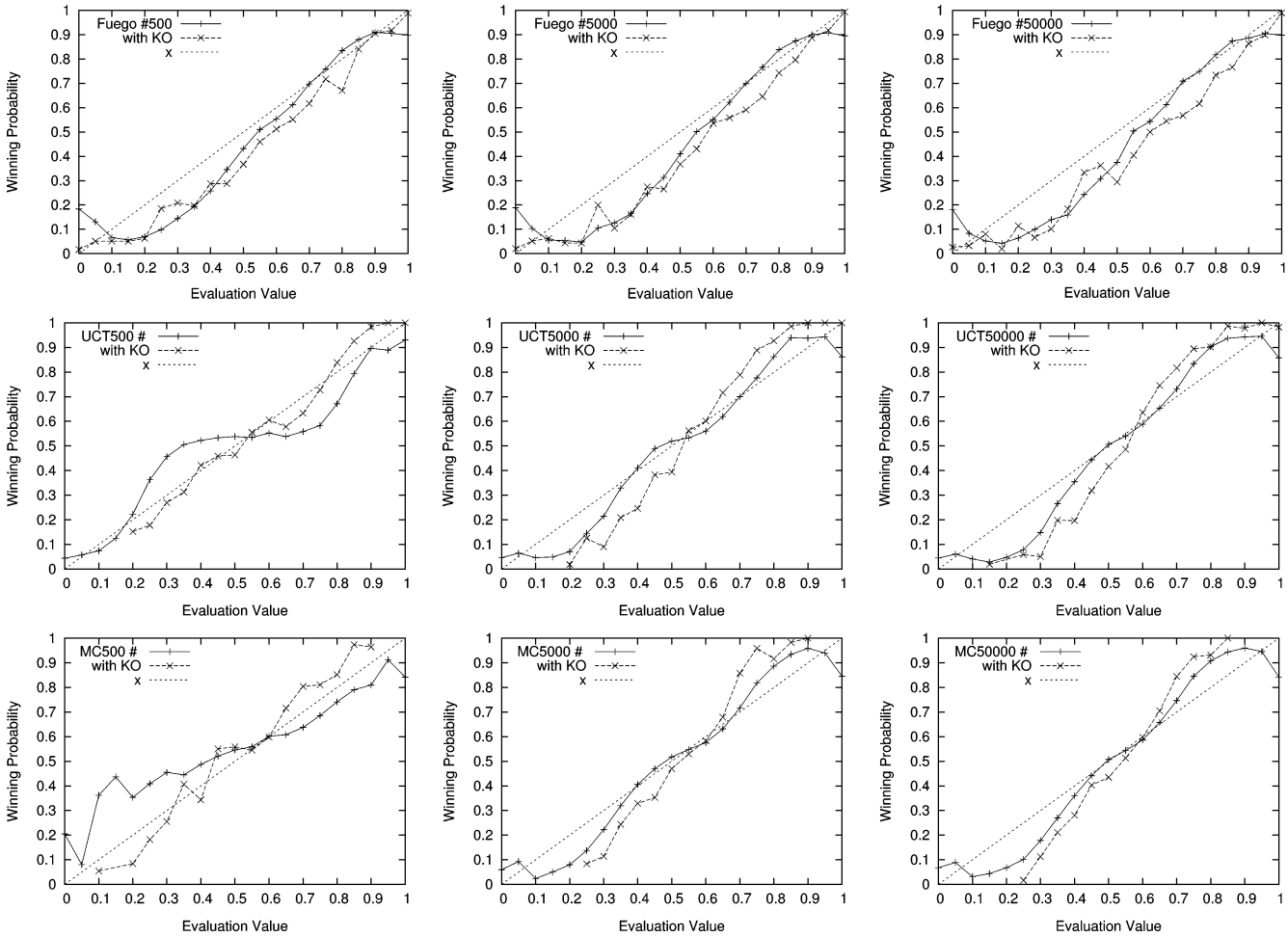

Fig. 9. Fuego, UCT, and MC at Ko positions: (top) Fuego, (middle) UCT, and (bottom) MC. (Left) 500 playouts. (Center) 5000 playouts. (Right) 50 000 playouts.

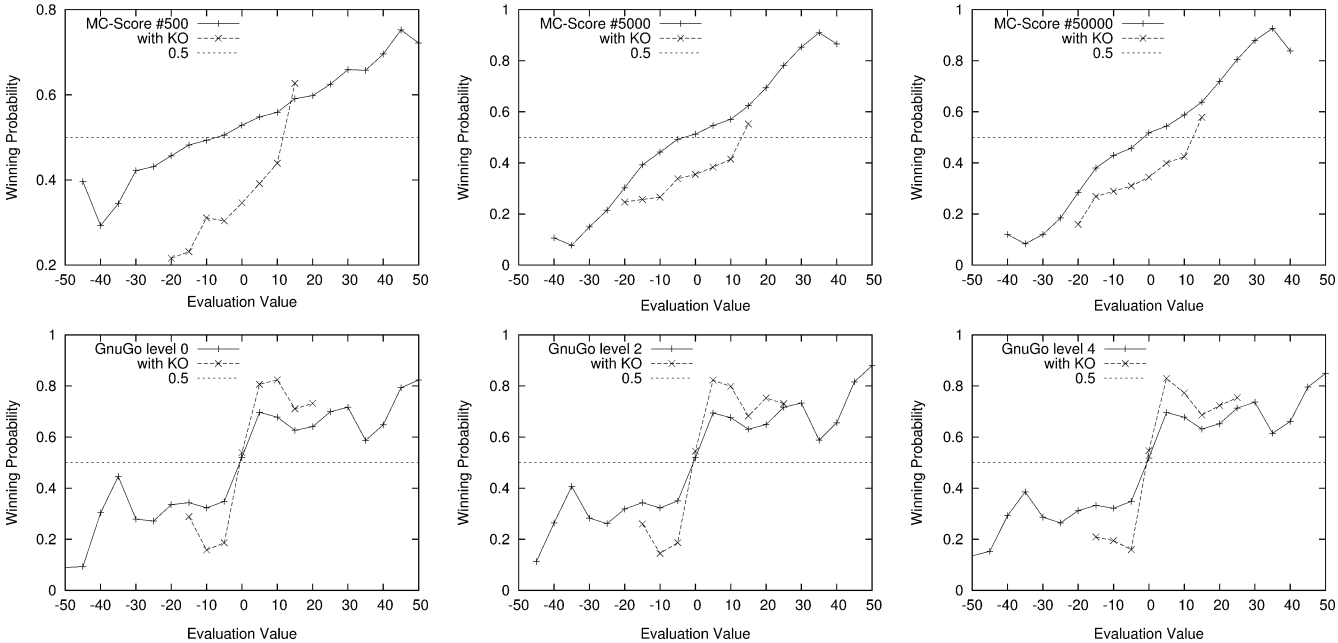

Fig. 10. MC-Score and GnuGo at Ko positions: (top) MC-Score and (bottom) GnuGo. (Left) 500 playouts/level 0. (Center) 5 000 playouts/level 2. (Right) 50 000 playouts/level 4.

curve labeled "without QS" in Fig. 11. The curves with a quiescence search are monotonously increasing; on the other hand, the curves without a quiescence search are, surprisingly, not monotonously increasing.

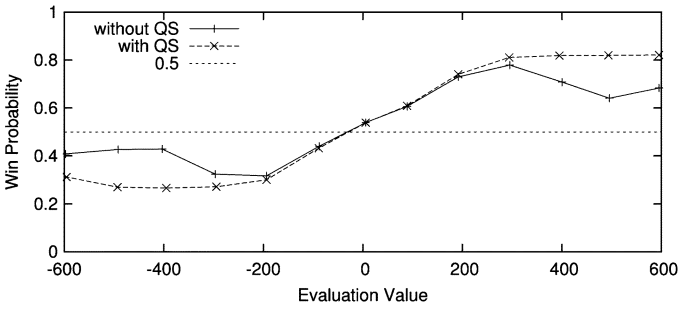

Fig. 11. Evaluation curves: *Chess* with/without quiescence search.

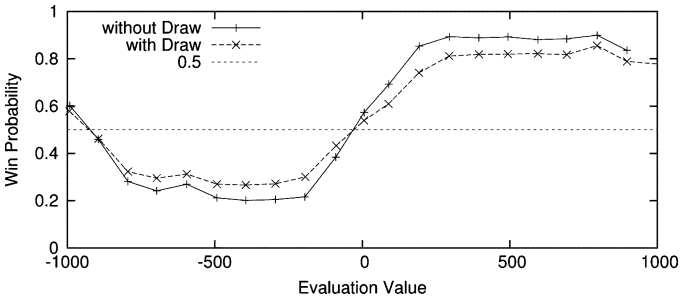

Fig. 12. Evaluation curve: *Chess*, with/without draw.

A comparison of both evaluation curves suggests that these fluctuations are caused by unreliable evaluations of tactical positions.

c) Bishop Evaluation: Here, we present an experiment on new features in *Chess*. Because we did not think of new features to add to Crafty, we simulated the addition of new features by disabling some existing features and then adding the existing features. Fig. 13 plots the evaluation curves for the feature of "Bishop Evaluation" (BE), which evaluates the mobility and development of Bishops. In the three evaluation curves, the broken (dotted) curve is an evaluation curve for the positions whose BE is more (less) than or equal to 50. The evaluation curve on the right is for the original evaluation function of Crafty, and the evaluation curve on the left is for a modified one whose BE was turned off. We can see that the conditioned curves differ from the total curve in the evaluation curve on the left. We then adjusted the weights of BE with MLM and LS, details of which are explained in the Appendix. The center evaluation curve in Fig. 13 plots the curves for the evaluation function adjusted by LS. We can see that the conditioned curves in the evaluation curve are much closer to the total curve. Table IV summarizes the weights relative to those of the original Crafty adjusted by MLM and LS.

We conducted 1000 self-plays between programs before adjustment, two programs after adjustment, and the original Crafty to find whether there were any improvements. Each player was given 10 min per game. The results are summarized in Table III. The programs after adjustment (MLM and LS) had more wins than those before adjustment (turned off) and they were statistically significant with a significance level[9] of 5%. Therefore adjustments done by our method effectively improved the evaluation functions. The original program had more wins than those after adjustment (MLM and LS).

[9]These were measured with a program written by Amir Ban that took draws into account (http://groups.google.com/group/rec.games.chess.computer/msg/764b0af34a9b4023, posted to rec.games.chess.computer).

TABLE III
RESULTS FOR SELF-PLAY IN *CHESS* (WINS–LOSSES–DRAWS)

MLM v.s. Turn off 423 - 290 - 287	MLM v.s. Crafty 246 - 312 - 442
LS v.s. Turn off 410 - 232 - 358	LS v.s. Crafty 269 - 303 - 428
Crafty v.s. Turn off 402 - 202 - 396	

TABLE IV
RESULTS OF ADJUSTING WEIGHTS IN *CHESS*

Method	MLM	LS	Original
Bishop Evaluation	1.75	1.33	(1.00)

2) Shogi: We used GPS *Shogi* (rev. 1117 with Open Shogi Library rev. 2602, on September 20, 2006),[10] which took eighth place at the World Computer *Shogi* Championship in 2005, a winner in 2009, and third place in 2010. We used 90 000 records from the Shogi Club 24 [30] in order. We employed a checkmate search for *Shogi* in up to 10 000 nodes for each position, from the first position to the last in a record to determine the winner of each record. If a checkmate was found, the player for the position was determined to have won. It took several hours to obtain evaluation values with checkmate search on an AMD Opteron 2.2-GHz processor.

In our previous work [27], we observed that evaluation curves with and without quiescence search are quite similar. Thus, we plot evaluation curve without quiescence search in this experiment.

a) Progress bonus [King Unsafety (KU)]: We introduced a new evaluation feature to *Shogi*, the difference in the "King's Unsafety" (KU) for both players.

The conditioned curves of the evaluation function in GPS *Shogi* differ from the total curve as shown in Fig. 14, when there is a large difference between the KUs of both players. We therefore prepared a new evaluation function and adjusted its weights with our methods. GPS *Shogi* originally had two kinds of evaluation functions. The first one was for the opening (e_o) and for evaluating the material balance, as well as the combination of pieces to take the development of pieces into account. The second one was for the endgame (e_e) and for evaluating the relative positions of the Kings and the other pieces. They were combined by a progress rate pr whose range was $[0, 1]$

$$e(p) = (1 - \mathrm{pr}) \cdot e_o + \mathrm{pr} \cdot e_e. \tag{5}$$

We then designed a new evaluation function that incorporated two new features, i.e., f_a and f_d

$$e'(p) = (1 - \mathrm{pr}) \cdot e_o + \mathrm{pr} \cdot (e_e + w_1 \cdot f_a + w_2 \cdot f_d) \tag{6}$$

where f_a represents the difference in KUs measured using attacking pieces and f_d represents the difference measured using the defending pieces. Here, the differences are multiplied by

[10]http://gps.tanaka.ecc.u-tokyo.ac.jp/{gpsshogi,osl}

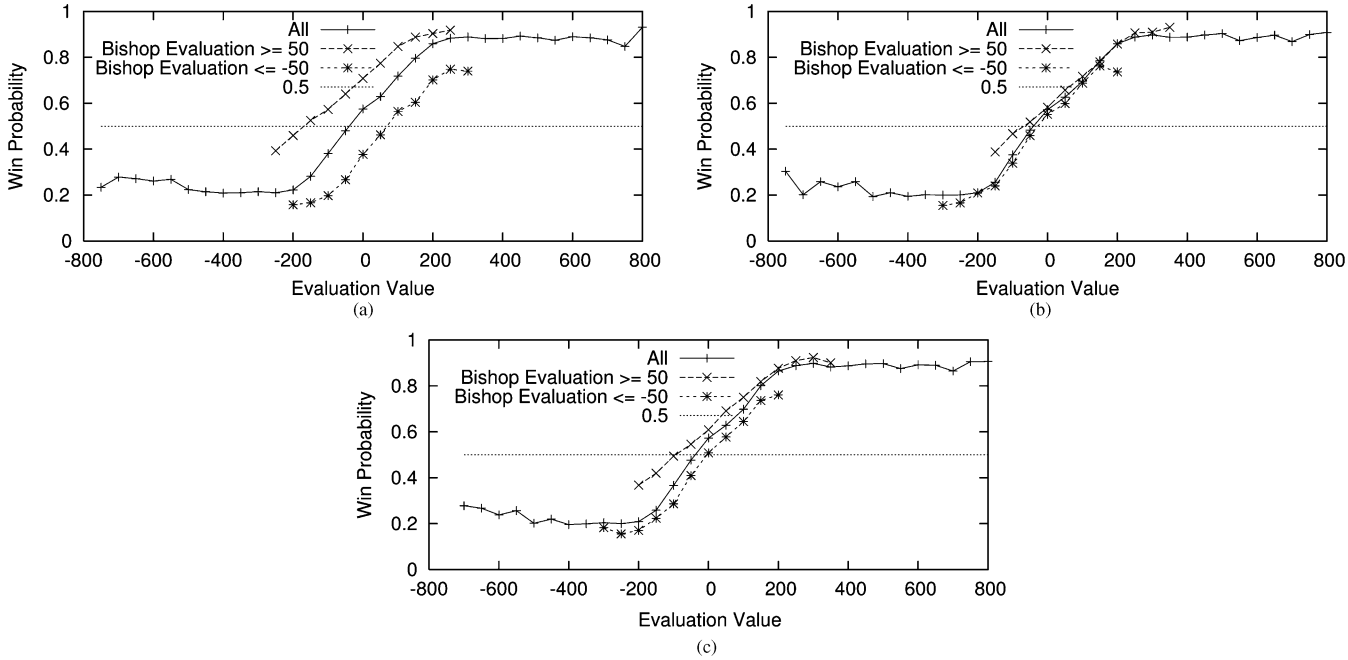

Fig. 13. Evaluation curve in *Chess*: (a) without Bishop Evaluation; (b) adjusted by LS; and (c) original Crafty.

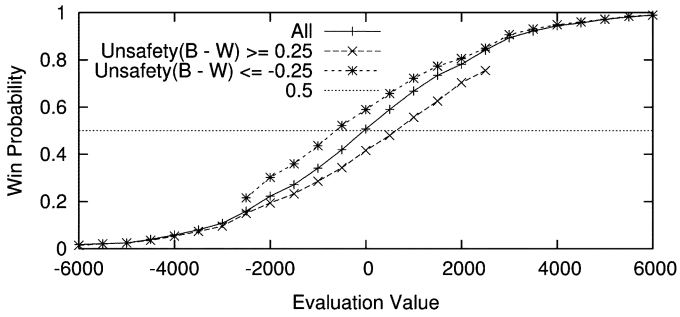

Fig. 14. Evaluation curves: *Shogi* without quiescence search, difference in KUs.

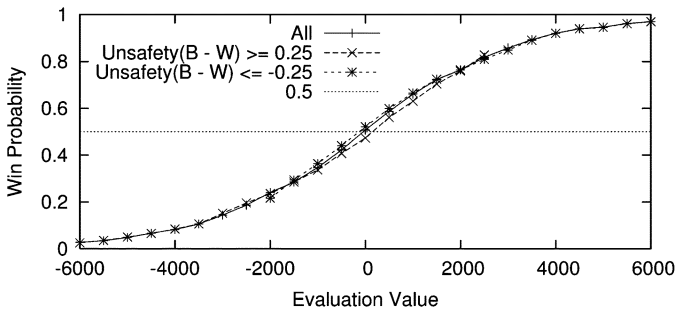

Fig. 15. Evaluation curve in *Shogi* (difference in KUs, adjusted by MLM).

the rate of progress in $e'(p)$ because it is empirically known that such differences are of more importance near the endgame. When weights w_1 and w_2 are 0, (6) reduces to (5).

Table VI compares the weights adjusted by MLM as well as those manually adjusted. We can see that they have similar values. The evaluation curves after adjusting them with MLM are plotted in Fig. 15. (We have omitted manually adjusted curves because they are very similar to those in Fig. 15.) The conditioned curves are much closer to the total curves than those in Fig. 14.

TABLE V
RESULTS FOR SELF-PLAY IN *SHOGI* (WINS–LOSSES–DRAWS)

MLM v.s. Orig. 600 - 392 - 8	MLM v.s. Hand 499 - 493 - 8
Hand v.s. Orig. 566 - 421 - 13	

TABLE VI
RESULTS OF ADJUSTING WEIGHTS IN *SHOGI*

	w_1	w_2		w_1	w_2
MLM	- 115	83	Hand	- 125	50

We conducted 1000 self-plays between programs before adjustment and two programs adjusted by MLM and manually to find whether there were any improvements. We used positions after 30 moves in the professional game records as the initial positions for self-play. Each player was given 10 min per game. The results are summarized in Table V. The program with the new evaluation function (MLM) had more wins against the original program (original), and it was statistically significant with a significance level of 5% in a binomial test. The adjustments based on our method therefore effectively improved evaluation functions. There were no statistically significant differences between adjustments done by MLM and those done manually.

V. APPLYING TO NEW GAMES

As shown in the experimental results, the presented methods worked robustly in many cases in different games and also in different search methods. Thus, we expect that the presented methods will be effective in other games. This section discusses practical issues to be considered when one applies our methods to new games.

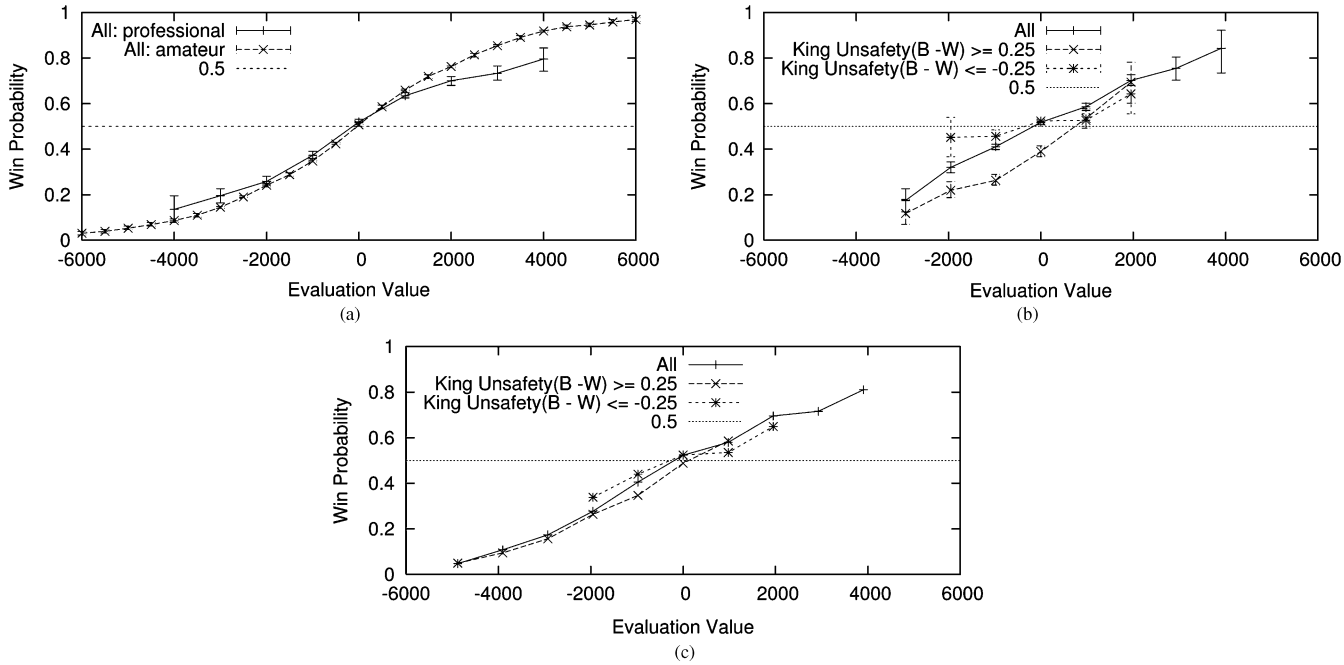

Fig. 16. Evaluation curves in *Shogi*: (a) amateur records versus professional records; (b) before adjustment (professional); and (c) after adjustment (professional).

A. Game Record Selection

Here, we explain how we prepare game records. We explain about player strength and other issues such as rule, handicap, and so on.

1) Player Strength: Since our methods use the wins and losses of game records, we need an assumption that the records were played by players with reasonable strength. Intuitively, if records were played by random players, we cannot extract any information from the number of wins and losses.

In order to see the dependency on the strength of players, we conducted additional experiments with another set of game records. In the additional experiments, the similar results were observed when a different set of game records were used as explained below.

We conducted additional experiments with professional game records. We used 603 records from the 59th Junisen, a professional championship tournament in *Shogi*. Fig. 16(a) plots the total evaluation curves for the professional records, as well as those for amateur records (Shogi Club 24). Because there were an insufficient number of professional records, we used intervals consisting of more than 100 positions and added error bars for the confidence interval of 5%. We can see that the probability of wins for the professional records increases more gradually than that for the amateur records. This suggests that difficult positions appear more often in professional game records for computers.

Fig. 16(b) plots the evaluation curves for the original evaluation function. Although the curves are not as clearly sigmoid due to the limited number of the records, we can see that the conditioned curves differ from the total curve in the professional records, as well as in the amateur records (Fig. 14). Fig. 16(c) plots the evaluation curves for the new evaluation function adjusted by MLM in the previous section. The conditioned curves are much closer to the total curves for the professional records, even though the evaluation function was adjusted using the amateur records. Evaluation functions adjusted by using amateur records are thus also expected to be effective in professional records.

TABLE VII
THE NUMBER OF GAME RECORDS IN 9×9 *Go*

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

We conducted an experiment to see the effect of strength in game records by the ratings of the records for *Go*. Table VII shows 2000 games from *Go* play server KGS for different ratings, komi, rules, and win/loss in 9×9 *Go*. In rate, d (dan) is stronger than k (kyu). Dan and kyu are associated with number. For dan, the larger number means stronger. The number ranges from 1 to 9. For kyu, the smaller number means stronger. The number ranges from 1 to 40. In the following, rate $(d, 6k)$ means that at least one of the players has the strength between dan and 6 kyu. Fig. 17 shows the evaluation curves for games with rate $(d, 6k)$ and those with rate $(9k, 11k)$, and Fig. 18 shows the evaluation curves for games with rate $(d, 3k)$ and those with rate $(9k, 11k)$ for different komi. These evaluation curves are almost similar meaning that the rate of the games does not affect the result much.

At these evaluation curves win probability is lower than $y = x$. This is probably because the player with higher ratings plays the second in KGS.

2) Other Issues: Here, we explain other issues of preparing game records. The issues are rule, komi, both ends in evaluation curves, and draw.

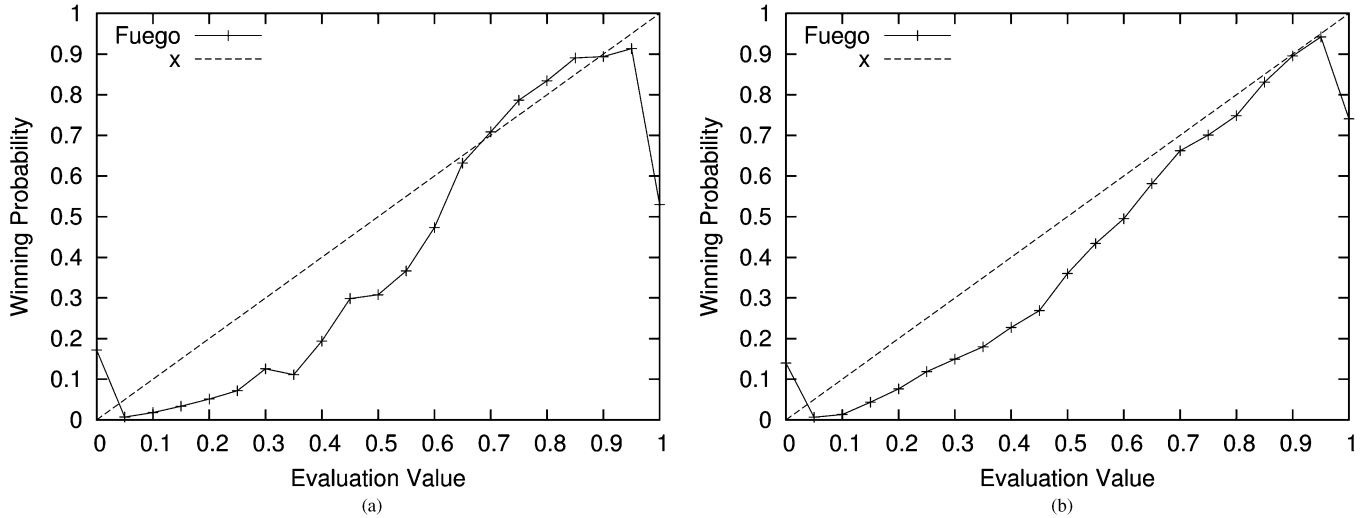

Fig. 17. Evaluation curve of Fuego, komi 6.5: (a) A $(d, 6k)$; (b) B $(9k, 11k)$.

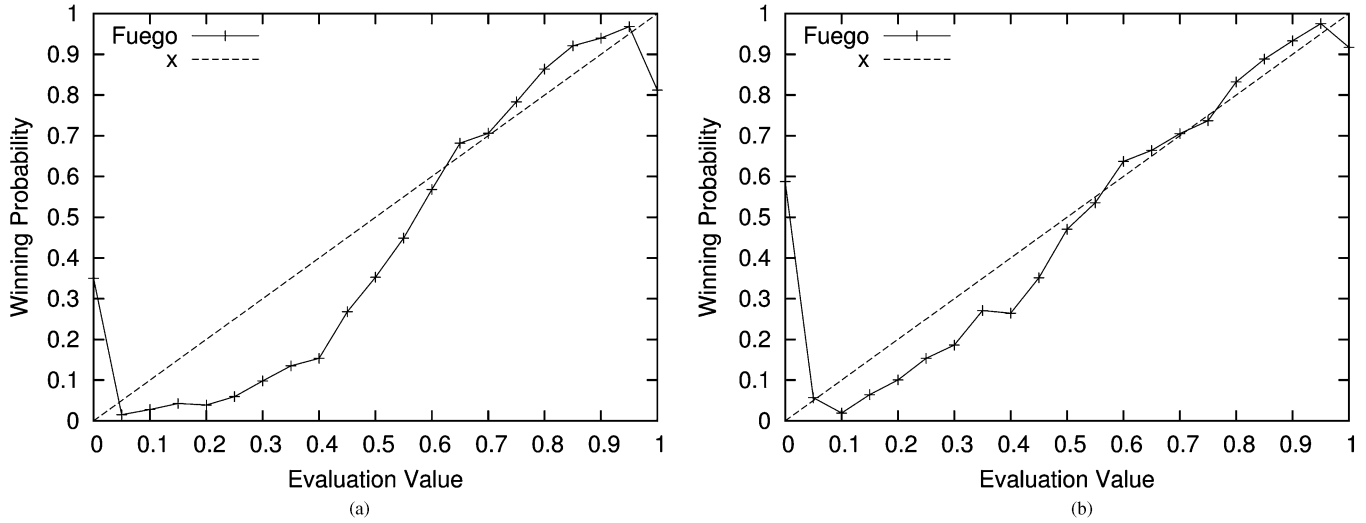

Fig. 18. Evaluation curve of Fuego, komi 0.5: (a) C $(d, 3k)$; (b) D $(9k, 11k)$.

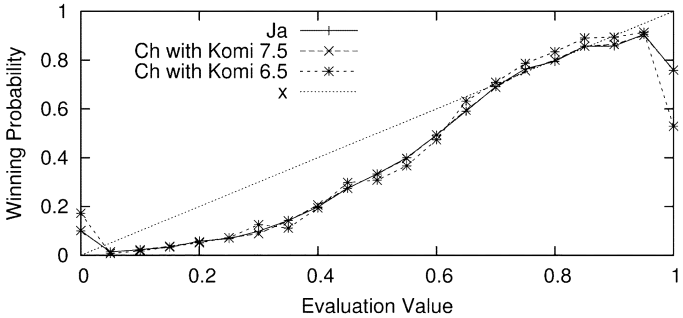

Fig. 19. Evaluation curves of Fuego (KGS, $\geq 6k$, komi 6.5).

For *Go*, there are slight variations in the games rules. There are the Chinese and Japanese rules. Also, there is difference in komi such as 6.5 and 7.5, and sometimes 0.5 for lower ratings. As shown in Figs. 17–19, rule and komi do not affect the result much.

At both ends ($x > 0.95$ or $x < 0.05$) of evaluation curves in *Go* such as Figs. 17–19, the win probabilities fluctuate significantly. This occurs when the winner of the game records and that determined by the program differ. One possible explanation is that the program correctly determines the winner while humans fail to find the win. For UCT and MC (Fig. 2), which are weaker than Fuego, there is not much fluctuation supporting this speculation.

As explained in Section IV-B1, records of draws can be discarded.

B. δ Selection

In order to calculate win probability from game records, we have to determine δ in (4). For smaller δ, the win probability is calculated from a small number of positions and the resolution of the win probability is lower. For larger δ, the resolution of evaluation value is lower. So, to strike a balance is important.

The resolution of the win probability calculated from n positions is estimated as follows. Suppose that there is "true" win probability μ and we estimate it from n samples. The error from μ and estimated $\bar{\mu}$ has the variance of (σ^2/n) and the standard deviation $\bar{\sigma} = (\sigma/\sqrt{n})$. Because each position results in win

$(=1)$ or loss $(=0)$, the variance $\sigma^2 = \mu(1-\mu)^2 + (1-\mu)(0-\mu)^2 = \mu(1-\mu)$. In the experiment in Section IV, we easily collected the game records with 10% error in the win probability and 5% error in evaluation value, which are sufficient for producing useful evaluation curves. For example, in *Chess*, 1 872 381 positions were available. See the evaluation curve labeled "all" in Fig. 11. At evaluation value $=0$ with $\delta = 50$ (2.5%), $\mu = 0.573$ for $n = 806\,912$ resulting in $\bar{\sigma} = 0.0005506$. At evaluation value $= 800$ with $\delta = 50$ (2.5%), $\mu = 0.900$ for $n = 2244$ resulting in $\bar{\sigma} = 0.006341$. In this case, if necessary, we may adaptively reduce δ around evaluation value $=0$ without much worsening the resolution in the win probability because the number of positions in the range is large.

In Section IV-A, we showed that Monte Carlo tree search performs better near the end of games in *Go*. These phenomena are observed in other games also. Thus, the numeric measures tend to be better for the game records that contain fewer early positions. In general, positions satisfying a given condition do not contain early, middle, and late positions uniformly. So, the comparison of numeric measures for positions for different conditions may favor the one with more late positions. In order to avoid this bias, care must be taken to normalize the distribution of early, middle, and late positions.

C. Method Selection

We proposed evaluation methods to use the relationship between the win probabilities and evaluation curves in Section III and they are used to reveal various facts in Section IV. Here, we summarize from their purposes.

Evaluation curve is a useful method to find missing features, such as Ladder (Fig. 8), Ko (Fig. 9), Bishop Evaluation (Fig. 13), and King Unsafety (Fig. 14). However, it is difficult to determine which number of playouts is stronger (Fig. 2).

On the other hand, numeric measures are useful to see the effect of playouts (Tables I and II). However, it is hard to find missing features. For example, ACC for UCT with 5000 playouts is 0.608 at all positions and 0.763 at Ko positions. This result is contrary to the empirical fact that the Monte Carlo tree search methods work more poorly at tactical positions than other positions. This phenomenon was explained in Section V-B.

VI. Concluding Remarks

We presented a means of measuring the performance of methods involving Monte Carlo tree search by using the relationship between evaluation value and win probability. By plotting evaluation curves for Monte Carlo tree search methods, we could see 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, by 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 [27].

Appendix

MLM and LS

Here, we explain MLM and LS for weight adjustment.

Because evaluation curves form a sigmoid as has been confirmed by numerous experiments that were discussed in Section IV, it is acceptable to use logistic regression that maximizes the likelihood of training examples (denoted as MLM). Let $g(p)$ be the win probability of a position approximated by the sigmoid transformation of $e'(p)$

$$g(p) = \frac{1}{(1+\exp(-w_0 \cdot e'(p)))}.$$

The likelihood in MLM for a training position p_i is defined as

$$\text{likelihood}\ (p_i, y_i) = g(p_i)^{y_i}(1-g(p_i))^{(1-y_i)}$$

where y_i denotes the winner of the ith training position whose value is 1 (0) if the winner is the black (white) player. Finally, weights $\hat{\mathbf{w}}$ are determined so that the product of the likelihood of each position is maximized

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \prod_i \text{likelihood}\ (p_i, y_i).$$

As an alternative, weights can be determined with least squares (LS) by minimizing the summation of the squared errors between y_i and $g(p_i)$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_i (y_i - g(p_i))^2.$$

Acknowledgment

The authors would like to thank the referees for their helpful comments.

References

[1] L. Kocsis and C. Szepesvari, "Bandit based Monte-Carlo planning," in *Machine Learning: ECML 2006*. Berlin, Germany: Springer-Verlag, 2006, vol. 4212, pp. 282–293.

[2] M. Müller, "Computer Go," *Artif. Intell.*, 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 *Proc. 23rd AAAI Conf. Artif. Intell.*, 2008, pp. 1537–1540.
[4] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Proc. Comput. Games*, 2006, pp. 72–83.
[5] M. Buro, "Improving heuristic mini-max search by supervised learning," *Artif. Intell.*, vol. 134, no. 1-2, pp. 85–99, Jan. 2002.
[6] J. van Rijswijck, "Learning from perfection: A data mining approach to evaluation function learning in awari," in *Computer and Games*, ser. Lecture Notes in Computer Science, T. A. Marsland and I. Frank, Eds. Berlin, Germany: Springer-Verlag, Oct. 2001, pp. 115–132, no. 2063.
[7] 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.
[8] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Dev.*, vol. 3, no. 3, pp. 210–229, 1959.
[9] M. Buro, "From simple features to sophisticated evaluation functions," in *Proc. 1st Int. Conf. Comput. Games*, Tsukuba, Japan, Nov. 1998, pp. 126–145.
[10] D. Gomboc, T. A. Marsland, and M. Buro, "Evaluation function tuning via ordinal correlation," in *Advances in Computer Games*. Berlin, Germany: Springer–Verlag, 2003, pp. 1–18.
[11] G. Tesauro, "Temporal difference learning and TD-Gammon," *Commun. ACM* vol. 38, no. 3, pp. 58–68, Mar. 1995 [Online]. Available: http://www.research.ibm.com/massdist/tdl.html
[12] J. Baxter, A. Tridgell, and L. Weaver, "Learning to play chess using temporal differences," *Mach. Learn.*, vol. 40, no. 3, pp. 243–263, 2000.
[13] M. L. Ginsberg, "GIB: Steps toward an expert-level bridge-playing program," in *Proc. 16th Int. Joint Conf. Artif. Intell.*, 1999, pp. 584–589.
[14] B. Sheppard, "World-championship-caliber scrabble," *Artif. Intell.*, vol. 134, no. 1-2, pp. 241–275, Jan. 2002.
[15] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron, "The challenge of poker," *Artif. Intell.*, vol. 134, no. 1–2, pp. 201–240, 2002.
[16] B. Abramson, "Expected-outcome: A general model of static evaluation.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 2, pp. 182–193, Feb. 1990.
[17] B. Brügmann, "Monte Carlo Go," Physics Dept., Syracuse Univ., Syracuse, NY, Tech. Rep., 1993.
[18] B. Bouzy and B. Helmstetter, "Monte Carlo Go developments," in *Advances in Computer Games. Many Games, Many Challenges.*. Norwell, MA: Kluwer, 2003, pp. 159–174.
[19] H. Yoshimoto, K. Yoshizoe, T. Kaneko, A. Kishimoto, and K. Taura, "Monte Carlo Go has a way to go," in *Proc. 21st Nat. Conf. Artif. Intell.*, 2006, pp. 1070–1075.
[20] 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. Berlin, Germany: Springer-Verlag, 2006, vol. 4630, pp. 72–83.
[21] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of UCT with patterns in Monte-Carlo Go," INRIA, Tech. Rep. RR-6062, 2006.
[22] G. M. J.-B. Chaslot, M. H. M. Winands, I. Szita, and H. J. van den Herik, "Cross-entropy for Monte-Carlo tree search," *J. Int. Comput. Games Assoc.*, vol. 31, no. 3, pp. 145–156, Sep. 2008.
[23] R. Coulom, "Computing Elo ratings of move patterns in the game of Go," *J. Int. Comput. Games Assoc.*, vol. 30, no. 4, pp. 198–208, Dec. 2007.
[24] D. Silver, R. Sutton, and M. Müller, "Sample-based learning and search with permanent and transient memories," in *Proc. 25th Annu. Int. Conf. Mach. Learn.*, A. McCallum and S. Roweis, Eds., 2008, pp. 968–975.
[25] N. Araki, "Move prediction and strength in Monte-Carlo Go program," M.S. thesis, Grad. School, Univ. Tokyo, Tokyo, Japan, 2008.
[26] S. Takeuchi, T. Kaneko, and K. Yamaguchi, "Evaluation of Monte Carlo tree search and the application to Go," in *Proc. IEEE Symp. Comput. Intell. Games*, 2008, pp. 191–198.
[27] S. Takeuchi, T. Kaneko, K. Yamaguchi, and S. Kawai, "Visualization and adjustment of evaluation functions based on evaluation values and win probability," in *Proc. 22nd Nat. Conf. Artif. Intell.*, 2007, pp. 858–863 [Online]. Available: http://www.graco.c.utokyo.ac.jp/
[28] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms using different performance metrics," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 161–168.
[29] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006.
[30] H. Kume, "Shogi-Club-24-Man-Kyoku-Shu," (in Japanese) Naitai Shuppan Co., 2002, ISBN: 4931538037.

Shogo Takeuchi received the B.S., M.S., and Ph.D. degrees in multidisciplinary sciences from the University of Tokyo, Tokyo, Japan, in 2005, 2007, and 2010, respectively.

Currently, he is the Japan Society for the Promotion of Science (JSPS) Research Fellow at the University of Tokyo. His research interest is in artificial intelligence for computer games, especially in *Shogi*.

Tomoyuki Kaneko received the Ph.D. degree in multidisciplinary sciences from the University of Tokyo, Tokyo, Japan, in 2002.

Currently, he is an Assistant Professor at the University of Tokyo. His research interests include game programming and machine learning. He is also known to be a developer of a computer *Shogi* program, GPSShogi, which was the winner of the World Computer Shogi Championship in 2009.

Kazunori Yamaguchi received the B.S., M.S., and Doctor of Science degrees in information science from the University of Tokyo, Tokyo, Japan, in 1979, 1981, and 1985, respectively.

Currently, he is a Professor at the University of Tokyo. His research interest is in data models for database, artificial intelligence, argumentation, language processing, education, and visualization.