Journal of Artificial Intelligence Research 49 (2014) 527-568 Submitted 10/13; published 03/14

Large-Scale Optimization for Evaluation Functions with Minimax Search

Kunihito Hoki HOKI@CS.UEC.AC.JP
Department of Communication Engineering and Informatics
The University of Electro-Communications

Tomoyuki Kaneko KANEKO@ACM.ORG
Department of Graphics and Computer Sciences
The University of Tokyo

Abstract

This paper presents a new method, *Minimax Tree Optimization* (MMTO), to learn a heuristic evaluation function of a practical alpha-beta search program. The evaluation function may be a linear or non-linear combination of weighted features, and the weights are the parameters to be optimized. To control the search results so that the move decisions agree with the game records of human experts, a well-modeled objective function to be minimized is designed. Moreover, a numerical iterative method is used to find local minima of the objective function, and more than forty million parameters are adjusted by using a small number of hyper parameters. This method was applied to shogi, a major variant of chess in which the evaluation function must handle a larger state space than in chess. Experimental results show that the large-scale optimization of the evaluation function improves the playing strength of shogi programs, and the new method performs significantly better than other methods. Implementation of the new method in our shogi program Bonanza made substantial contributions to the program's first-place finish in the 2013 World Computer Shogi Championship. Additionally, we present preliminary evidence of broader applicability of our method to other two-player games such as chess.

1. Introduction

Heuristic search is a powerful method in artificial intelligence. In 1997, the chess-playing computer Deep Blue defeated the world chess champion Garry Kasparov (Campbell, Hoane, & Hsu, 2002). The computer decided its moves after making a large number of searches of the minimax game tree and using heuristic evaluation functions. In this framework of artificial intelligence, the heuristic evaluation functions, as well as the search methods, are crucial for making strong computer players. Thus, researchers working on various games have made substantial efforts in a quest to create effective evaluation functions by using machine learning techniques (Fürnkranz, 2001). However, fully automated learning of the heuristic evaluation functions remains a challenging goal in chess variants. For example, developers have reported that the majority of the features and weights in Deep Blue were created/tuned by hand (Campbell et al., 2002). It is said that recent top-level chess programs tune some of their parameters automatically, although we have yet to find any publication describing the methods they use. Moreover, reinforcement learning has been applied to chess (Baxter, Tridgell, & Weaver, 2000; Veness, Silver, Uther, & Blair, 2009).

However, to the best of the authors' knowledge, the evaluation functions learned by the methods reported in the literature are still weaker than the best hand-crafted functions in terms of chess-playing strength.

In this paper, we revisit the idea behind earlier research on learning chess evaluation functions (Marsland, 1985; Hsu, Anantharaman, Campbell, & Nowatzyk, 1990; Tesauro, 2001) and reformulate the task as an optimization problem using an alternative learning method, called *Minimax Tree Optimization* (MMTO). The objective here is to optimize a full set of parameters in the evaluation function so that the search results match the desired move decisions, e.g., the recorded moves of grandmaster games. The evaluation functions are learned through iteration of two procedures: (1) a shallow heuristic search for all training positions using the current parameters and (2) a parameter update guided by an approximation of the gradient of the objective function. To achieve scalability and stability, we introduce a new combination of optimization techniques: a simplified loss function, *grid-adjacent* update, equality constraint, and l_1-regularization. One of the resulting merits is that MMTO can ensure the existence of a local minimum within a convenient range of parameters.

This study demonstrates the performance of MMTO in shogi, a variant of chess where evaluation functions need to handle a wider variety of features and positions than in Western chess. Implementation of MMTO in our shogi program Bonanza (described in Section 4.6) made substantial contribution to the program's first-place finish in the 2013 World Computer Shogi Championship. The rules of shogi, as well as a survey of approaches in artificial intelligence, are described in the literature (Iida, Sakuta, & Rollason, 2002). Basic techniques, such as a minimax search guided by heuristic evaluation functions, are as effective in shogi as in chess. However, the "drop rule" that allows a player to reuse captured pieces significantly changes a few properties: (1) the number of legal moves, as well as average game length, is greater than in chess, (2) endgame databases are not available, (3) and the material balance is less important than in chess, especially in the endgame. Thus, the performance of a shogi program is more dependent on the quality of its evaluation function.

Through experiments, we first show that the full set of parameters in the evaluation functions can be optimized with respect to the rate of agreement with the training set. After that, we examine the performance of various learned evaluation functions in terms of their rates of agreement with the test positions and win rates against references. Scalability is demonstrated up to about forty million parameters, which is far too many to tune by hand. The features we used are piece values and extended versions of piece-square tables that are commonly used to learn evaluation functions in chess (Tesauro, 2001; Baxter et al., 2000; Veness et al., 2009). We also briefly examine the performance of MMTO in chess to catch a glimpse of the applicability of MMTO to other games.

The rest of this paper is organized as follows. The next section reviews related research. The third section presents the MMTO method. The fourth section shows our experimental results, where forty million of parameters are adjusted for better performance, and compares the performance of our method with that of existing methods. The last section presents our concluding remarks. This paper incorporates and extends our previous work (Hoki & Kaneko, 2012; Kaneko & Hoki, 2012).

2. Related Work

This section reviews related research on learning evaluation functions. First, we describe supervised learning methods that use the desired moves. Second, we discuss other learning methods, including regression and reinforcement learning. Third, we briefly discuss the difficulty of supervised learning in terms of numerical optimization. Although machine learning of other components besides evaluation functions in game programs would be an interesting research topic (Björnsson & Marsland, 2002; Tsuruoka, Yokoyama, & Chikayama, 2002; Coulom, 2007; Silver & Tesauro, 2009), this review only focuses on research that has been done on learning evaluation functions.

2.1 Learning from Desired Moves in Chess

Grandmaster games are a popular source of information for learning chess. Let us say that we have a set of positions $\mathcal{P}$ and the desired moves for each position in $\mathcal{P}$. Typically, such positions and moves are sampled from grandmaster games. A chess program has an evaluation function $\mathrm{e}(p, \boldsymbol{w})$, where p is a game position and the feature weight vector $\boldsymbol{w}$ contains the parameters to be adjusted.

Let us assume that the evaluation function $\mathrm{e}(p, \boldsymbol{w})$ is partially differentiable with respect to w_i for any i. Here, w_i is the i-th component of $\boldsymbol{w}$. For example, the function could be a linear combination of weighted features, i.e., $\mathrm{e}(p, \boldsymbol{w}) = \sum_i w_i f_i(p)$, where $f_i(p)$ is the i-th feature value of position p. The aim of learning is to find a better weight vector $\boldsymbol{w}$ for strengthening the play of the program. The hypothesis behind this kind of learning is that the more the computer play agrees with the desired moves, the better it plays.

Let us begin with a simple intuitive goal: make the results of a one-ply search agree with the desired moves. For simplicity, let us assume that the maximizing player moves first at the root position p. In a one-ply search, the move with the highest evaluation value is selected. Thus, $\boldsymbol{w}$ should be adjusted so that the desired move has the highest evaluation of all the moves. This goal can formally be written as a mathematical minimization problem with an objective function:

$$J_H^{\mathcal{P}}(\boldsymbol{w}) = \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}'_p} H\left(\mathrm{e}(p.m, \boldsymbol{w}) - \mathrm{e}(p.d_p, \boldsymbol{w})\right). \tag{1}$$

Here, $p.m$ is the position after move m in position p, d_p is the desired move in position p, $\mathcal{M}'_p$ is the set of all legal moves in p excluding d_p, and $H(x)$ is the Heaviside step function, i.e., $H(x)$ equals 1 if $x \geq 0$, and 0 otherwise. Because this objective function counts the number of moves that have an evaluation value greater than or equal to that of the desired move, a better $\boldsymbol{w}$ can be found by minimizing Eq. (1). Although several studies have attempted machine learning on the basis of this framework (Nitsche, 1982; van der Meulen, 1989; Anantharaman, 1997), their numerical procedures were complicated, and the adjustment of a large-scale vector $\boldsymbol{w}$ seemed to present practical difficulties.

Marsland (1985) presented a notable extension wherein a continuous function is used so that conventional optimization techniques can be exploited. Here, a continuous function of difference is substituted for the non-continuous step function in Eq. (1). An interesting modified function is

$$J_2^{\mathcal{P}}(\boldsymbol{w}) = \sum_{p\in\mathcal{P}} \sum_{m\in\mathcal{M}'_p} \left[\max\left\{0, \mathrm{e}(p.m, \boldsymbol{w}) - \mathrm{e}(p.d_p, \boldsymbol{w})\right\}\right]^2 . \tag{2}$$

The meaning of the function value is different from that in Eq. (1); i.e., the function does not count the number of moves that have an evaluation value greater than or equal to that of the desired move. However, the gradient vector $\nabla_{\boldsymbol{w}} J_2^{\mathcal{P}}(\boldsymbol{w})$ helps to reduce the function value numerically. Marsland also introduced inequality constraints in order to keep the evaluation in the right range. However, the literature does not provide any experimental results on practical chess programs.

A second notable extension was proposed early in the development of chess machines *Deep Thought* (Nowatzyk, 2000; Hsu et al., 1990). Here, the positions being compared are not $p.m$, but rather $\pi_{\boldsymbol{w}}^{p.m}$, that is, one of the leaves of the principal variations (PVs), possibly several plies from $p.m$. This extension carries out a least-square fitting of the evaluation values. Therefore, it does not have the max function that $J_2^{\mathcal{P}}(\boldsymbol{w})$ does. Instead, it biases the value of $\mathrm{e}(\pi_{\boldsymbol{w}}^{p.d_p}, \boldsymbol{w})$ before it is used in each least-square fitting, if the evaluation value of the desired move d_p, $\mathrm{e}(\pi_{\boldsymbol{w}}^{p.d_p}, \boldsymbol{w})$ is lower than that of another move m, $\mathrm{e}(\pi_{\boldsymbol{w}}^{p.m}, \boldsymbol{w})$.

A third notable extension is the comparison training proposed by Tesauro (2001). Tesauro modified the objective function to

$$J_{\mathrm{ct}}^{\mathcal{P}}(\boldsymbol{w}) = \sum_{p\in\mathcal{P}} \sum_{m\in\mathcal{M}'_p} T_{\mathrm{ct}}(\mathrm{e}(\pi_{\boldsymbol{w}}^{p.d_p}, \boldsymbol{w}) - \mathrm{e}(\pi_{\boldsymbol{w}}^{p.m}, \boldsymbol{w})),$$

$$T_{\mathrm{ct}}(x) = \left[\sigma(R(x)) - 1\right]^2 , \tag{3}$$

where σ is the standard sigmoid function, and R is a heuristic rescaling factor for positive differences, i.e., $R(x) = x$ when $x \le 0$, and $R(x) = cx$ for a constant $c > 1$ otherwise. Note that $R(x)$ is still a continuous function. The important property of this modified objective function is that the value and the derivative are zero in the limit as the difference x goes to positive infinity, and they are respectively one and zero in the limit as the difference x goes to negative infinity. Therefore, $T_{\mathrm{ct}}(x)$ in Eq. (3) is a continuous approximation of $H(-x)$ in Eq. (1). Note that this property is not explicitly stated by Tesauro, but it is notably distinct from the other work. The number of feature weights adjusted with his method was less than two hundred. Tesauro also mentioned an application for small-bit integers, which was used to adjust some of the weights in Deep Blue. However, he neither clarified its procedure nor mentioned whether the weights were automatically adjusted in that experiment.

Table 1 summarizes the related work. Each of the existing methods possesses at least one of three important properties for optimization, i.e., continuity, minimax searches, and assured local minimum. However, none of them have all three properties. Also, some of the existing methods (Nowatzyk, 2000; Hsu et al., 1990; Tesauro, 2001) do not try to decrease the functions through iteration as much as possible. We will revisit these issues in Section 2.3. On the other hand, our method, MMTO, has scalability in high-dimensional learning. Moreover, we empirically show that a decrease in the objective function value leads to an increase in playing strength. The existing methods have not been shown to have this property.

Method	Continuity	Search	Assured local minimum
(Nitsche, 1982)	No	No	No
(Marsland, 1985)	Yes	No	No
(van der Meulen, 1989)	No	No	No
(Hsu et al., 1990)	Yes*	Yes	No
(Anantharaman, 1997)	No	Yes	Yes
Comparison training	Yes*	Yes	No
MMTO	Yes*	Yes	Yes

Table 1: Summary of learning methods using the desired moves in training positions to adjust the feature weights in the evaluation functions. The first column is the name of the method or piece of literature. The second column describes the continuity of the objective functions with respect to the feature weights. Yes* means that continuity depends on the kind of search method used. The third column indicates whether the objective functions use minimax searches with depths more than 1, instead of comparisons of legal moves at the root position. The fourth column shows whether the hyper parameters of the objective functions can assure a local minimum can be found.

2.2 Other Methods of Learning Evaluation Functions

Many researchers have utilized information sources other than the desired moves. For example, some studies on Othello dating from the 1990s compare the desired moves with other moves (Fawcett, 1993). However, the most practical and famous machine learning method that has yielded strong programs is based on regression of the desired value by using 1.5 million features (Buro, 1995, 2002). In Othello, different evaluation functions are used for game stages determined on the basis of the number of discs in play. Thus, the desired values of the training positions are obtained through a complete endgame search as well as a heuristic search with evaluation functions learned in later game stages. This method has also been successfully applied to card games (Buro, Long, Furtak, & Sturtevant, 2009), but not to chess variants. To the best of the authors' knowledge, learning based on regression with win/loss-labeled data has not yielded decent evaluation functions in chess variants. Except for not using the desired moves, Buro's method has properties that are similar to those listed in Table 1; his objective function has continuity as well as an assured local minimum, and his method is scalable. Gomboc, Buro, and Marsland (2005) proposed to learn from game records annotated by human experts; however, the feature weights that were adjusted in their experiments were only a small part of the full evaluation functions.

Reinforcement learning (Sutton & Barto, 1998), especially temporal difference learning, of which a famous success is Backgammon (Tesauro, 2002), is considered to be promising way to avoid the difficulty in finding the desired values for regression. This approach has been applied to chess and has been shown to improve the strength of programs (Baxter et al., 2000; Levinson & Weber, 2001; Veness et al., 2009). The KnightCap program achieved a rating of about $2,150$ points at the Free Internet Chess Server (FICS[1]) and

1. Free Internet Chess Server, `http://www.freechess.org`, last access: 2013

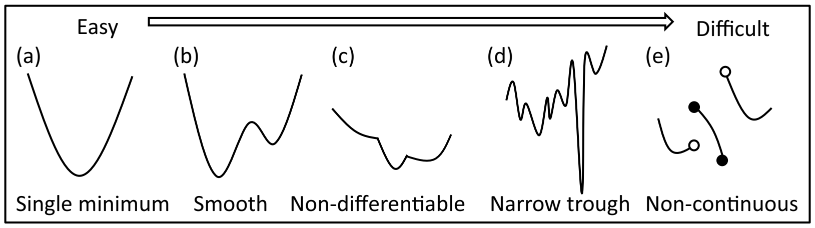

Figure 1: Example illustrating the difficulties facing any minimization procedure.

2, 575 points at its highest peak at the Internet Chess Club (ICC) (Baxter et al., 2000). Another program achieved 2, 338 points at its highest peak at ICC (Veness et al., 2009). However, strong human players have ratings of more than 3, 000 points at ICC, and this difference means these programs have not reached the top level of chess programs; that is, evaluation functions tuned by reinforcement learning have not yet reached the level of the best-handcrafted evaluation functions in chess. Moreover, the number of feature weights to be adjusted is on the order of thousands. In checkers, evaluation functions trained by temporal difference learning are reportedly comparable to the best handcrafted efforts (Schaeffer, Hlynka, & Jussila, 2001). It has also been reported that a player stronger than expert human checker players was created by using neural networks trained with an evolutionary strategy (Chellapilla & Fogel, 1999). Here, no features beyond the piece differentials were given to the neural network a priori.

Many machine learning techniques (Baxter et al., 2000; Veness et al., 2009) have been applied to shogi. However, despite efforts by many programmers and researchers, the adjustment of the full weight vector in the evaluation function remains a challenging goal. The studies published so far have adjusted only piece values or a small part of the feature weights in the evaluation functions (Beal & Smith, 2001; Ugajin & Kotani, 2010).

2.3 Learning and Numerical Optimization

Some learning methods reviewed in Section 2.1 have objective functions to decrease; the learning process can be extended into a numerical optimization using these functions. The performance of numerical optimization is sensitive to the surface of the objective function. Figure 1 shows the properties of particular sorts of functions and their difficulties regarding numerical minimization. The easiest one among them is the convex function (a); if a local minimum exists, then it is a global minimum. Function (b) has multiple local minima; however, it can still be thought of as an easy problem, because various minimization algorithms using gradients and Hessian matrices are effective on it. It would be desirable to design a learning method using, say, linear or logistic regression, which uses one of these two types of objective function (Buro, 2002).

In contrast, non-differentiable functions such as (c) and (e) are often more difficult to minimize than differentiable ones. This is because a quadratic model, such as the Hessian approximation of the conjugated gradient method (Bertsekas & Bertsekas, 2008), is not always appropriate for these functions. Function (d) is also a difficult target, because an important local minimum is hidden inside a deep narrow trough, and it is quite difficult to find it by using numerical iteration methods. The most difficult example is minimization of the non-continuous function (e); even primitive iterative methods such as gradient decent are not capable of finding its minimum. The extreme case would be a function for which an analytical formula for the gradient is unavailable. In that case, a learning method would not be able to use partial derivatives, and the minima would have to be obtained using derivative-free methods, e.g., sampling methods (Björnsson & Marsland, 2002; Coulom, 2012).

Theorems in Appendix A show that the minimax value is continuous but not always partially differentiable. Thus, the existing methods that incorporate a minimax search (Hsu et al., 1990; Tesauro, 2001) and MMTO listed in Table 1 are type (c). Moreover, certain forward pruning techniques may cause discontinuities. Therefore, even these learning methods can be type (e). To overcome this difficulty, MMTO has a well-modeled objective function and updates the feature weights in a careful manner.

3. Minimax Tree Optimization

Minimax Tree Optimization (MMTO) is an extension of comparison training to reach the first intuitive goal embodied in Eq. (1). The purpose of this extension is to overcome the practical difficulties and stabilize the mathematical optimization procedure with a large-scale feature weight vector $\boldsymbol{w}$. Given a set of training positions $\mathcal{P}$ and the desired move d_p for each position p, MMTO optimizes the weight vector $\boldsymbol{w}$ so that the minimax search with $\boldsymbol{w}$ better agrees with the desired moves.

The weight vector $\boldsymbol{w}$ is improved through iteration of sub-procedures (see Figure 2). For each iteration t, the first step consists of tree searches to identify one of the leaves of PVs $\pi_{\boldsymbol{w}(t)}$ for all legal moves in the training positions $\mathcal{P}$. Because PV leaf $\pi_{\boldsymbol{w}(t)}$ depends on the feature weights $\boldsymbol{w}(t)$ in an evaluation function, a new PV will be obtained when $\boldsymbol{w}(t)$ is updated (We discuss this issue in Section 3.5). The second step is the calculation of the approximate partial derivatives, which depends on both PV and the weight vector. The last step is the update of the weight vector. For numerical stability, the difference $|\boldsymbol{w}(t+1) - \boldsymbol{w}(t)|$ must be kept small so that it will not be distorted by drastic changes in the partial derivatives. Section 3.4 shows that a grid-adjacent update ensures this.

3.1 Objective Function to be Minimized

The objective function is

$$J^{\mathcal{P}}_{\text{MMTO}}(\boldsymbol{w}) = J(\mathcal{P}, \boldsymbol{w}) + J_{\text{C}}(\boldsymbol{w}) + J_{\text{R}}(\boldsymbol{w}), \tag{4}$$

where the first term $J(\mathcal{P}, \boldsymbol{w})$ on the right side is the main part. The other terms J_{C} and J_{R} are constraint and regularization terms, respectively, which are defined in Section 3.2.

The first term is

$$J(\mathcal{P}, \boldsymbol{w}) = \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}'_p} T\left(\text{s}(p.d_p, \boldsymbol{w}) - \text{s}(p.m, \boldsymbol{w})\right), \tag{5}$$

where $\text{s}(p, \boldsymbol{w})$ is the minimax value identified by the tree search for position p. $T(x)$ is $1/(1 + \exp(ax))$, which is a horizontally mirrored sigmoid function. The slope of $T(x)$ is controlled by a constant parameter $a > 0$. In the large a limit, $T(x)$ becomes the Heaviside

1. Perform a game-tree search to identify PV leaves $\pi^{p.m}_{\boldsymbol{w}(t)}$ for all child positions $p.m$ of position p in training set $\mathcal{P}$, where $\boldsymbol{w}(t)$ is the weight vector at the t-th iteration and $\boldsymbol{w}(0)$ is the initial guess.
2. Calculate a partial-derivative approximation of the well-modeled objective function defined in Section 3.1 by using both $\pi^{p.m}_{\boldsymbol{w}(t)}$ and $\boldsymbol{w}(t)$. The objective function employs a differentiable approximation of $H(x)$ (see Section 3.1), as well as a constraint and regularization term (see Section 3.2).
3. Obtain a new weight vector $\boldsymbol{w}(t{+}1)$ from $\boldsymbol{w}(t)$ by using a grid-adjacent update guided by the partial derivatives computed in step 2 (see Section 3.4). Go back to step 1, or terminate the optimization when the objective function value converges (see Section 4).

Figure 2: Minimax Tree Optimization: Iteration of searches and update using partial derivatives

step function $H(x)$. Thus, the main differences from the first intuitive objective function $J^{\mathcal{P}}_{H}(\boldsymbol{w})$ in Eq. (1) are the use of $T(x)$ for a smooth approximation of $H(x)$ and the use of the search result $\mathrm{s}(p,\boldsymbol{w})$ instead of the raw evaluation $\mathrm{e}(p,\boldsymbol{w})$. The difference from $J^{\mathcal{P}}_{2}(\boldsymbol{w})$ in Eq. (2) and $J^{\mathcal{P}}_{\mathrm{ct}}(\boldsymbol{w})$ in Eq. (3) is that $J(\mathcal{P},\boldsymbol{w})$ is simpler and closer to the first intuitive one in Eq. (1). Moreover, none of the existing studies incorporate $J_{\mathrm{C}}(\boldsymbol{w})$ or $J_{\mathrm{R}}(\boldsymbol{w})$ in Eq (4).

The minimax value $\mathrm{s}(p,\boldsymbol{w})$ equals the raw evaluation value $\mathrm{e}(\pi^{p}_{\boldsymbol{w}},\boldsymbol{w})$, where $\mathrm{e}(p,\boldsymbol{w})$ is the evaluation of position p and $\pi^{p}_{\boldsymbol{w}}$ is one of the PV leaves identified by the tree search rooted at p with a weight vector $\boldsymbol{w}$. In most cases, the derivatives of $\mathrm{s}(p,\boldsymbol{w})$ equal the derivatives of $\mathrm{e}(\pi^{p}_{\boldsymbol{w}},\boldsymbol{w})$. For these reasons, the PV leaves are identified in step 1 in Figure 2.

3.2 Constraint and Regularization Terms

In the computer programs of chess variants, the evaluation values are typically represented by integers. Signed 16-bit integers are especially preferred because the corresponding transposition tables will be memory efficient. Thus, we will restrict the range of the absolute value of the evaluation function $\mathrm{e}(p,\boldsymbol{w})$. Moreover, because the search results do not change when one uses a scaled weight vector $\alpha\boldsymbol{w}$ with a constant factor $\alpha > 0$, this restriction stabilizes the numerical optimization procedure if the value of α is uncertain.

For this restriction, we introduce a constraint term $J_{\mathrm{C}}(\boldsymbol{w}) = \lambda_0 g(\boldsymbol{w}')$ in Eq. (4), where $\boldsymbol{w}'$ is a subset of $\boldsymbol{w}$, $g(\boldsymbol{w}') = 0$ is an equality constraint, and λ_0 is a Lagrange multiplier. In addition to the constraint term, we also introduce a regularization term $J_{\mathrm{R}}(\boldsymbol{w})$ (see the last term in Eq. (4)). We use l_1-regularization $J_{\mathrm{R}}(\boldsymbol{w}) = \lambda_1|\boldsymbol{w}''|$, where $\lambda_1 > 0$ is a constant variable, and $\boldsymbol{w}''$ is a subset of $\boldsymbol{w}$. l_1-regularization is widely used to deal with high-dimensional parameters, whereas l_2-regularization is used to avoid over-fitting (Tibshirani, 1996).

The constraint and regularization terms ensure that a local minimum of $J^{\mathcal{P}}_{\text{Mmto}}(\boldsymbol{w})$ exists in a finite range of $\boldsymbol{w}$. On the other hand, depending on $\mathcal{P}$ and the distribution of d_p, this property is not always true for $J(\mathcal{P}, \boldsymbol{w})$ itself, for $J^{\mathcal{P}}_2(\boldsymbol{w})$ in Eq. (2), or for $J^{\mathcal{P}}_{\text{ct}}(\boldsymbol{w})$ in Eq. (3).

The constraint and l_1-regularization terms have similar functionalities; i.e., both restrict the range of the absolute value of the evaluation function $\mathrm{e}(p, \boldsymbol{w})$. However, their distinctions are important in practice because l_1-regularization makes the weight vector $\boldsymbol{w}''$ sparse whereas the constraint term does not. Thus, the regularization term is suitable for minor features that are rarely seen, whereas the constraint term is suitable for major features that appear often in the training set. Moreover, both terms are useful for controlling the strength of the restriction. Because major feature values usually change more often than minor feature values, the magnitudes of the partial derivatives with respect to major feature weights are usually greater than those with respect to minor feature weights. We can adjust the strength of l_1-regularization term so that it is weaker than the constraint term.

For example, our experiments used the constraint term for the piece values because their feature values, i.e., the number of pieces owned by black/white, change in most single games of shogi. The many other weights were penalized by l_1-regularization. Each weight was controlled by either the constraint or l_1-regularization term, i.e., $\boldsymbol{w} = (\boldsymbol{w}', \boldsymbol{w}'')$. Because the partial derivatives with respect to the major and minor feature weights differed by several orders of magnitude, it was difficult to stabilize the optimization procedure by means of a single hyper parameter λ_1.

3.3 Partial Derivative Approximation

In each iteration, feature weights are updated on the basis of the partial derivatives of the objective function $J^{\mathcal{P}}_{\text{Mmto}}(\boldsymbol{w})$ defined by Eq. (4). The partial derivative, if it exists, is

$$\frac{\partial}{\partial w_i} J^{\mathcal{P}}_{\text{Mmto}}(\boldsymbol{w}) = \frac{\partial}{\partial w_i} J(\mathcal{P}, \boldsymbol{w}) + \frac{\partial}{\partial w_i} J_{\mathrm{C}}(\boldsymbol{w}) + \frac{\partial}{\partial w_i} J_{\mathrm{R}}(\boldsymbol{w}). \tag{6}$$

The last term $\frac{\partial}{\partial w_i} J_{\mathrm{R}}(\boldsymbol{w})$ on the right side is treated in an intuitive manner; $\mathrm{sgn}(w_i)\lambda_1$ for $w_i \in \boldsymbol{w}''$, and 0 otherwise. Function $\mathrm{sgn}(x)$ is 1 for $x > 0$, 0 for $x = 0$, and -1 for $x < 0$. The partial derivative of the constraint term $\frac{\partial}{\partial w_i} J_{\mathrm{C}}(\boldsymbol{w})$ is 0 for $w_i \notin \boldsymbol{w}'$. The case of $w_i \in \boldsymbol{w}'$ is discussed in Section 3.5.

The partial derivative of $J(\mathcal{P}, \boldsymbol{w})$ does not always exist, because the minimax value $\mathrm{s}(p, \boldsymbol{w})$ is not always differentiable. Instead, we can use an approximation,

$$\begin{aligned}
\frac{\partial}{\partial w_i} J(\mathcal{P}, \boldsymbol{w}) &= \frac{\partial}{\partial w_i} \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}'_p} T\left(\mathrm{s}(p.d_p, \boldsymbol{w}) - \mathrm{s}(p.m, \boldsymbol{w})\right) & (7)\\
&\approx \frac{\partial}{\partial w_i} \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}'_p} T\left(\mathrm{e}(\pi^{p.d_p}_{\boldsymbol{w}}, \boldsymbol{w}) - \mathrm{e}(\pi^{p.m}_{\boldsymbol{w}}, \boldsymbol{w})\right) & (8)\\
&= \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}'_p} T'(\mathrm{e}(\pi^{p.d_p}_{\boldsymbol{w}}, \boldsymbol{w}) - \mathrm{e}(\pi^{p.m}_{\boldsymbol{w}}, \boldsymbol{w})) \cdot \frac{\partial}{\partial w_i} \left(\mathrm{e}(\pi^{p.d_p}_{\boldsymbol{w}}, \boldsymbol{w}) - \mathrm{e}(\pi^{p.m}_{\boldsymbol{w}}, \boldsymbol{w})\right),
\end{aligned}$$

where $T'(x) = \frac{\mathrm{d}}{\mathrm{d}\,x} T(x)$. The approximation of Eq. (7) by Eq. (8) makes the computation tractable, because we identify the PV leaves in step 1 in Figure 2. As stated in Appendix A, minimax value $\mathrm{s}(p, \boldsymbol{w})$ found by the $\alpha\beta$ search is continuous, and therefore, the function $J(\mathcal{P}, \boldsymbol{w})$ is also continuous. Moreover, the approximate value is equivalent to the partial derivative when a unique PV exists for each position. Appendix A also discusses the conditions under which $\frac{\partial}{\partial w_i}\,\mathrm{s}(p, \boldsymbol{w})$ exists. Note that we have found that the errors caused by this approximation are sufficiently small for the shogi application (Kaneko & Hoki, 2012). Previous studies (Baxter et al., 2000; Tesauro, 2001) use this approximation as well.

3.4 Grid-Adjacent Update

For numerical stability, the grid-adjacent update in step 3 (see Figure 2) is used to get $\boldsymbol{w}(t+1)$ from $\boldsymbol{w}(t)$. Consider a simple n-dimensional grid in which the distance between two adjacent points is h. Suppose that h is an integer, e.g., $h = 1$. In the grid-adjacent update, the feature vector $\boldsymbol{w}(t)$ is always one of the points of the grid, and the i-th component $w_i(t+1)$ is adjacent to $w_i(t)$:

$$w_i(t+1) = w_i(t) - h\,\mathrm{sgn}\left(\frac{\partial J^{\mathcal{P}}_{\mathrm{MMTO}}(\boldsymbol{w}(t))}{\partial w_i}\right).$$

Thus, $|w_i(t+1) - w_i(t)| = |\Delta w_i| = h$ or 0 for all i. This update should decrease the objective function $J^{\mathcal{P}}_{\mathrm{MMTO}}(\boldsymbol{w})$ because $\Delta\boldsymbol{w} \cdot \nabla_{\boldsymbol{w}} J^{\mathcal{P}}_{\mathrm{MMTO}}(\boldsymbol{w}) \leq 0$ and the errors in the approximation (see Eq. (8)) are negligible. Moreover, h must be small enough so that the update does not change the PV, i.e., $\pi^p_{\boldsymbol{w}(t)} = \pi^p_{\boldsymbol{w}(t+1)}$ for the majority of positions p searched in step 1.

Although MMTO focuses on optimization of weight vectors represented by integers, it should be noted that the gradient descent update is not suitable even when one uses floating-point feature weights. Our preliminary experiments indicate that the partial derivatives of $J(\mathcal{P}, \boldsymbol{w})$ with respect to major and minor feature weights differ by more than seven orders of magnitude. Thus, an update vector $\Delta\boldsymbol{w}$ that is proportional to the gradient vector may not be appropriate for updating the minor feature weights with a small step. Thus, the step size of each component in a weight vector should be fixed as in the grid-adjacent update, or it might be able to be controlled in other ways (see, e.g., Duchi, Hazan, & Singer, 2011).

3.5 Combination of Techniques and Practical Issues

MMTO is a combination of the above-described techniques. This subsection discusses the practical issues of this combination and its alternatives; some relate to external constraints on learning (e.g., how many weeks we can wait for results), and some depend on the properties of the domain to which MMTO is applied.

3.5.1 Lagrange Multiplier in Grid-Adjacent Update

For numerical stability, MMTO explores a restricted parameter space where the constraint is satisfied, i.e., $J_{\mathrm{C}}(\boldsymbol{w}) = 0$. To do this, the Lagrange multiplier λ_0 in $J_{\mathrm{C}}(\boldsymbol{w})$ is set to the median of the partial derivatives $\{\frac{\partial J(\mathcal{P},\boldsymbol{w})}{\partial w_i} \,|\, w_i \in \boldsymbol{w}'\}$ in order to maintain the constraint $g(\boldsymbol{w}') = 0$ in each iteration. As a result, $\Delta w'_i$ is h for n feature weights, $-h$ for n feature weights, and 0 in one feature weight, where the number of feature weights in $\boldsymbol{w}'$ is $2n+1$. On the other hand, λ_1 in the regularization term $J_{\mathrm{R}}(\boldsymbol{w})$ is constant in all iterations.

3.5.2 Search Depth

The game tree searches in step 1 in Figure 2 are the most time-consuming step in MMTO. Tesauro (2001) has shown that the use of a quiescence search yields better evaluation functions. Thus, it is expected that deeper searches in MMTO will yield better evaluation functions. On the other hand, we must handle a large amount of training positions, and the search time tends to grow exponentially when we increase the search depth. Therefore, most of our experiments use a 1-ply standard search together with a quiescence search. Here, the quiescence search is called at every frontier node of the standard search. We observed that evaluation functions learned with shallow searches are still effective for playing games with deep searches (see Section 4.4). Similar results were reported by Tesauro.

3.5.3 Reuse of PV for Efficiency of Learning

Because step 1 in Figure 2 is the most time-consuming part, it is worth considering omitting it by assuming $\pi^p_{\boldsymbol{w}(t)} = \pi^p_{\boldsymbol{w}(t-1)}$ with a certain frequency. In our experiments, steps 2 and 3 were repeated 32 times without running step 1. We counted the number of iterations in the run of step 1. That is, each iteration ran a single step 1 and 32 pairs of steps 2 and 3. The number 32 would be domain dependent and should be set small enough so that the update does not change the PV for most positions.

3.5.4 Pruning of Trees

Pruning techniques can dramatically reduce the number of searched nodes and hence speed up learning. Fortunately, $\alpha\beta$ pruning does not introduce any discontinuities in the objective function. On the other hand, other pruning methods, including futility pruning (Schaeffer, 1986), may introduce discontinuities (see Appendix A.4). Therefore, the robustness of the whole learning procedure should be examined when such pruning techniques are used. As far as the authors' experience goes, the objective function with futility pruning seems to be continuous (see Section 4.5).

3.5.5 Convergence and Performance Measurement

The termination criteria is usually difficult to determine in iterative computations. In the case of learning the shogi evaluation function, the convergence of the objective function in MMTO seems to be a significant criteria, because the rate of agreement with the test set and the Elo rating of the learned evaluation function also converge when it converges. Note that the rate of agreement should be measured on a separate test set from the training set in order to detect overfitting (see Section 4.3).

3.5.6 Duplication in Positions and Alternative Moves

Game records usually have duplications in the positions and desired moves in the opening phase. Although the ideal distributions of these positions and desired moves are unknown, we decided to remove the duplications from the training and test sets for simplicity. That is, we use each pair of $\langle \textit{position}, \textit{move} \rangle$ at most once in each iteration. These duplications are detected by Zobrist hashing (1990). Note that two or more different moves may be suggested for the same position in the training and test sets, and the objective function becomes smaller if the tree search rooted at the position matches one of these moves. As a result, conflicting goals such as "move a should be better than move b and vice versa" are independently augmented to the objective function and cancel each other when both moves can be played in the same position. In our experience, this adaptation seems to work reasonably well in shogi, but the best solution may depend on the target game.

4. Experiments

We evaluated the effectiveness of MMTO in experiments in which the number of feature weights in the evaluation function was varied from thirteen to about forty million. We found that MMTO works better than comparison training and its intuitive modifications in terms of the rate of agreement, speed of convergence, and game-playing strength. We also observed that the constraint term $J_{\mathrm{C}}(\boldsymbol{w})$ and regularization term $J_{\mathrm{R}}(\boldsymbol{w})$ help to increase the performance of the evaluation functions in terms of the rate of agreement with the test set. To see numerical convergence, we investigated the surfaces of the objective function in MMTO with a limited number of feature weights and experimentally found that MMTO finds local minima in a reasonable range of feature weights. Finally, we carried out preliminary experiments on chess as well as experiments on data quality dependence.

4.1 Setup: Evaluation Functions, Features, and Game Records

Most of the experiments described in this section used Bonanza, whose source code is available online (Hoki & Muramatsu, 2012). The performance of Bonanza in major tournaments is discussed in Section 4.6. Bonanza uses techniques such as MMTO, PVS (Pearl, 1980; Marsland & Campbell, 1982; Reinefeld, 1983), a capture search at frontier nodes as a quiescence search, transposition tables (Zobrist, 1990; Russell & Norvig, 2002), static exchange evaluation (Reul, 2010), killer and history heuristics (Akl & Newborn, 1977; Schaeffer, 1989), null move pruning (Adelson-Velskiy, Arlazarov, & Donskoy, 1975; Heinz, 1999), futility pruning (Schaeffer, 1986; Heinz, 1998), and late move reductions (Romstad, 2010). It also uses an opening-book database from which we randomly chose the opening lines of self-play experiments. The game records in the training and test sets were exclusively chosen from games played in famous tournaments[2]. There were $48,566$ game records in total. More than $30,000$ of the games were played by professional players using standard time controls, i.e., from one to ten hours for a side with a "byoyomi" period (once the time

2. The abbreviated tournament name, number of games we used, and date range of the games are: Juni, 12827, 1946–2010; Kisei, 3286, 1962–2010; Ryuo, 3279, 1987–2010; Osho, 2286, 1950–2010; Oui, 2017, 1959–2010; Ouza, 1849, 1952–2010; NHK-cup, 1745, 1951–2010; Ginga, 1735, 1991–2010; Kio, 1620, 1973–2010; Shinjino, 1332, 1969–2010; Zen-nihon-proshogi, 1160, 1982–2001; Hayazashi-shogi-senshuken, 945, 1972–2003; Judan, 764, 1962–1987; Meisho, 752, 1973–1989; Joryu-meijin, 608, 1974–2010; Meijin, 551, 1935–2010; All-star-kachinuki, 545, 1978–2003; Rating-senshuken, 476, 1987–2007; Asahi-open, 429, 2001–2007; Heisei-saikyo, 412, 1992–2007; Teno, 351, 1984–1992; Joryu-osho, 351, 1979–2010; Kurashiki-touka, 314, 1993–2010; Nihon-series, 304, 1981–2010; 3-dan-league, 283, 1963–2009; Ladies-open, 255, 1987–2007; Joryu-oui, 253, 1990–2010; Shoureikai, 217, 1941–2008; Gakusei-osho, 212, 1972–2006; Hayazashi-shinei, 206, 1982–2002; Gakusei-ouza, 191, 2001–2006; Asahi-amashogi, 187, 1980–2009; Wakajishi, 183, 1953–1991; Kudan, 182, 1947–1961; Gakusei-meijin, 177, 1972–2006; Shogi-Renmei-cup, 172, 1967–1984; Tatsujin, 160, 1993–2010; Kinsho-cup, 156, 2002–2005; Amateur-meijin, 146, 1948–2009; Kashima-cup, 119, 1996–2006; Grand-champion, 111, 1981–2008; Saikyosya-kettei, 101, 1954–1973; Miscellaneous, 5317, 1607–2010.

evaluation function		dimension
e^{A}	$\sum_{i=1}^{13} \left[f_i^{\mathrm{A}}(p) - f_i^{\mathrm{A}}(\tilde{p})\right] w_i^{\mathrm{A}}$	13
e^{B}	$\sum_{k,j} \left[f_{kj}^{\mathrm{B}}(p) - f_{kj}^{\mathrm{B}}(\tilde{p})\right] w_{kj}^{\mathrm{B}}$	$60,876$
e^{C}	$\sum_{k,k',l} \left[f_{kk'l}^{\mathrm{C}}(p) - f_{kk'l}^{\mathrm{C}}(\tilde{p})\right] w_{kk'l}^{\mathrm{C}}$	$2,425,950$
e^{D}	$\sum_{k,j\leq j'} \left[f_{kjj'}^{\mathrm{D}}(p) - f_{kjj'}^{\mathrm{D}}(\tilde{p})\right] w_{kjj'}^{\mathrm{D}}$	$44,222,454$

Table 2: Dimensions of evaluation functions. Each evaluation function is a linear combination of weighted features. e^{A} evaluates the material balance, and the others evaluate a variety of positional scores by using extended piece-square tables.

had expired, a player had to move within sixty seconds). Some tournaments employed rapid time controls such as 30 seconds per move and had top-level amateur players as participants.

Table 2 shows the four basic evaluation functions, where e^{A} is for material balance and the others are for positional scores. Our experiments used the sum of these functions, i.e., e^{A}, $\mathrm{e}^{\mathrm{AB}} = \mathrm{e}^{\mathrm{A}} + \mathrm{e}^{\mathrm{B}}$, $\mathrm{e}^{\mathrm{ABC}} = \mathrm{e}^{\mathrm{AB}} + \mathrm{e}^{\mathrm{C}}$, and $\mathrm{e}^{\mathrm{ABCD}} = \mathrm{e}^{\mathrm{ABC}} + \mathrm{e}^{\mathrm{D}}$. All evaluation functions are anti-symmetric with respect to the exchange of black and white: $\mathrm{e}(p, \boldsymbol{w}) = -\,\mathrm{e}(\tilde{p}, \boldsymbol{w})$. Here, $\tilde{p}$ is a complete reversal of the black and white sides at position p; that is, black plays for white and white plays for black[3]. After this reversal, the pieces owned by black and white in p are regarded as white and black pieces in $\tilde{p}$, respectively. Also, all evaluation functions are symmetric with respect to right-and-left mirroring of a position: $\mathrm{e}(p, \boldsymbol{w}) = \mathrm{e}(\hat{p}, \boldsymbol{w})$, where $\hat{p}$ is the mirror image of p along file `e`.

The function $\mathrm{e}^{\mathrm{A}}(p, \boldsymbol{w}^{\mathrm{A}})$ was used to evaluate the material balance. There are 13 types of pieces in shogi (Iida et al., 2002). Each feature $f_i^{\mathrm{A}}(p)$ represents the number of the i-th type owned by black in position p, and w_i^{A} is the relative value of the i-th type of piece. The partial derivative of the evaluation function with respect to w_i^{A} is $\partial\,\mathrm{e}^{\mathrm{A}}(p, \boldsymbol{w}^{\mathrm{A}})/\partial w_i^{\mathrm{A}} = f_i^{\mathrm{A}}(p) - f_i^{\mathrm{A}}(\tilde{p})$.

The function $\mathrm{e}^{\mathrm{B}}(p, \boldsymbol{w}^{\mathrm{B}})$ is a linear combination of weighted two-piece-square features. These are natural extensions of one-piece-square features that were employed in recent machine learning studies of chess evaluations (Baxter et al., 2000; Tesauro, 2001; Veness et al., 2009). These two-piece-square features were used to evaluate all conditions of the king and another piece. Each feature $f_{kj}^{\mathrm{B}}(p)$ is an indicator function that returns one if both of the conditions k and j exist in position p. Otherwise, it returns zero. Condition k represents the location of the black king (there are 81 squares), and j represents the type, owner (black or white), and location of the other piece. There were $1,476$ different conditions for j after some of the minor conditions were merged. Thus, the total number of the king–piece conditions was $81 \cdot 1,476 = 119,556$ before the mirror symmetric conditions were merged.

3. Following shogi notation, black and white refer to the players who plays first and second, respectively.

Similarly, the functions $\mathrm{e}^{\mathrm{C}}(p, \boldsymbol{w}^{\mathrm{C}})$ and $\mathrm{e}^{\mathrm{D}}(p, \boldsymbol{w}^{\mathrm{D}})$ were used to evaluate the king–king–piece features and king–piece–piece features, respectively. The indicator function $f^{\mathrm{C}}_{kk'l}(p)$ represents the location of the two kings (k, k') and the condition (type and location) of a black piece l. The indicator function $f^{\mathrm{D}}_{kjj'}(p)$ represents the location of black king k and the conditions of the other two black or white pieces (j, j').

Game tree searches are required to identify PV leaf positions for MMTO and to obtain best moves to measure the rate of agreement. For these purposes, a nominal depth 1 search was used together with the quiescence search. To normalize the objective function values, the objective function values were divided by the total number of move pairs, $Z^{\mathcal{P}} = \sum_{p\in\mathcal{P}} |\mathcal{M}'_p|$. The constraint function was set to

$$g(\boldsymbol{w}^{\mathrm{A}}) = \left(\sum_{i=1}^{13} w_i^{\mathrm{A}}\right) - 6,500. \tag{9}$$

Also, in accordance with the magnitude of the constraint, a in the horizontally mirrored sigmoid function $T(x) = 1/(1 + \exp(ax))$ was set to 0.0273 so that $T(x)$ would vary significantly if x changed by a hundred. The regularization term was $J_{\mathrm{R}}(\boldsymbol{w}) = 0.00625 \cdot (|\boldsymbol{w}^{\mathrm{B}}| + |\boldsymbol{w}^{\mathrm{C}}| + |\boldsymbol{w}^{\mathrm{D}}|)$. An intuitive explanation of the penalty strength is that the absolute value of w_i can be increased to 160 if doing so improves the relationship between the evaluation values of a desired move and another legal move. The sums in e^{B}, e^{C}, and e^{D} were computed using 32-bit integers, and they were divided by 2^5 in order to fit the evaluation value into a 16-bit integer. Step h in the grid-adjacent update was set to the smallest integer value 1.

4.2 Learning the Piece Values

First, the feature weights $\boldsymbol{w} = \boldsymbol{w}^{\mathrm{A}}$ of the evaluation function e^{A} were adjusted by MMTO or by comparison training, starting from the same initial value $w_i^{\mathrm{A}} = 500$ for all i. Tesauro (2001) used floating-point feature weights and the conventional gradient descent method. That is, the weight vector $\boldsymbol{w}$ was updated by

$$\boldsymbol{w}(t) = \boldsymbol{w}(t) - r\nabla_{\boldsymbol{w}} J^{\mathcal{P}}_{\mathrm{ct}}(\boldsymbol{w}), \tag{10}$$

where r is a constant training rate hand-tuned to 0.5. The components in $\boldsymbol{w}$ used in the tree search were rounded to the nearest integer values. The rescaling factor R in Eq. (3) was set to 0.0025 in accordance with the range of the difference $|\,\mathrm{e}(\pi_{\boldsymbol{w}}^{p.d_p}, \boldsymbol{w}) - \mathrm{e}(\pi_{\boldsymbol{w}}^{p.m}, \boldsymbol{w})|$ from 50 to $5,000$. Because this experiment had only 13 piece values to be adjusted in $\boldsymbol{w}^{\mathrm{A}}$, only $1,000$ game records were used to compose the training set $\mathcal{P}$. The set had 101,898 desired moves and $Z^{\mathcal{P}} = 7,936,180$ move pairs after removing duplications and handicapped games.

One problem observed in the comparison training was slow learning: as shown in Figure 3, phase I of the iterative procedure (from iteration 1 to 10) is mainly for adjusting the pawn value, because the partial differential value of Eq. (3) for pawns is the largest in this phase. After a good pawn value is found, phase II (from iteration 10 to 100) is mainly for adjusting the promoted rook and promoted bishop values. These values should be the highest and second highest for reasonable game play. The long period of time taken by phase II indicates that $Jct^{\mathcal{P}}(\boldsymbol{w})$ in Eq. (3) scales poorly. This is a general problem in gradient descent methods with multiple degrees of freedom (Nocedal & Wright, 2006), and

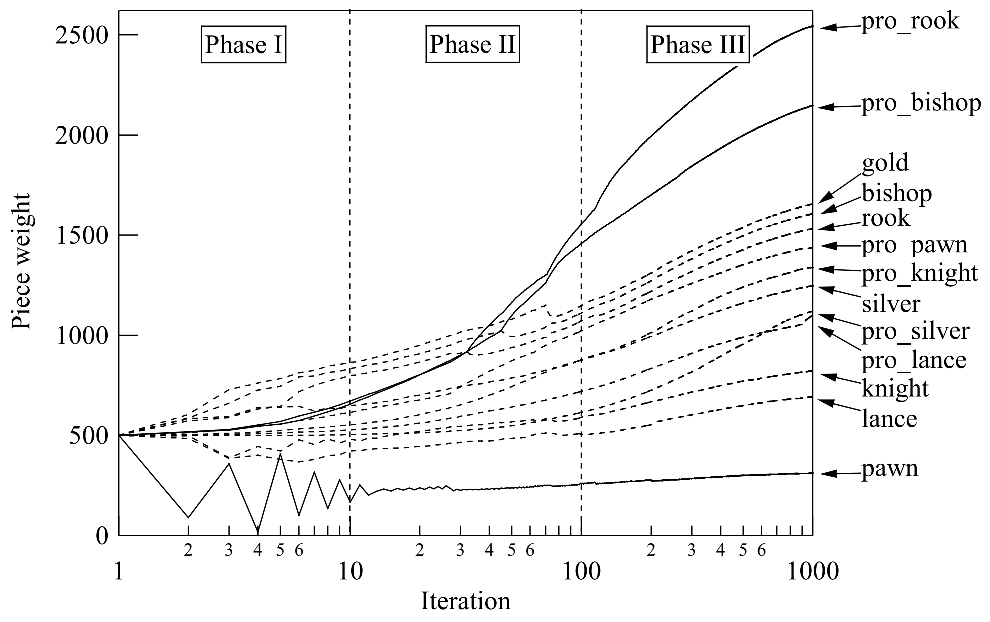

Figure 3: Results of comparison training of the piece weights in shogi. The horizontal axis plots the number of iterations on a logarithmic scale.

to cope with it, the learning rate r cannot be greater than 0.5 in accordance with the largest partial derivative in these experiments.

The second problem was about convergence: in phase III (after iteration 100) of Figure 3, all piece values keep increasing without changing the ratio of piece values, even though the relative ratios of the piece values have room for improvement. This problem is inherent to the objective function of comparison training, because Eq. (3) has no explicit term to avoid it. In an extreme case where the training data satisfy the inequality condition $\mathrm{e}(\pi_{\boldsymbol{w}}^{p.d_p}, \boldsymbol{w}) \geq \mathrm{e}(\pi_{\boldsymbol{w}}^{p.m}, \boldsymbol{w})$ for all moves m in any position p, all piece values diverge to infinity when the value $Jct^{\mathcal{P}}(\boldsymbol{w})$ is minimized. In fact, it was found that the training data in this experiment satisfied the condition for 94% of the pairs of the best and another legal move. Moreover, in the other extreme case, where the training does not satisfy the inequality condition for any move m in any position p in Eq. (3), all piece values shrink to zero.

MMTO deals with these problems by making grid-adjacent updates and keeping the magnitudes constant through its constraint term $J_{\mathrm{C}}(\boldsymbol{w})$. The weighted vector $\boldsymbol{w}^{\mathrm{A}}$ converged in 40 iterations (see Figure 4); the value of the promoted rook was 945 and that of the pawn was 122. Note that the number of iterations was counted as the number of step (1)s throughout the experiments.

4.3 Scalability in Learning Practical Evaluation Functions

After learning the piece values, we adjusted the weight vectors for positional scores. This time, a large number of training records were used to cope with high-dimensional weight vectors. The main training set $\mathcal{P}$ had $4,467,116$ desired moves and $Z^{\mathcal{P}} = 368,313,024$

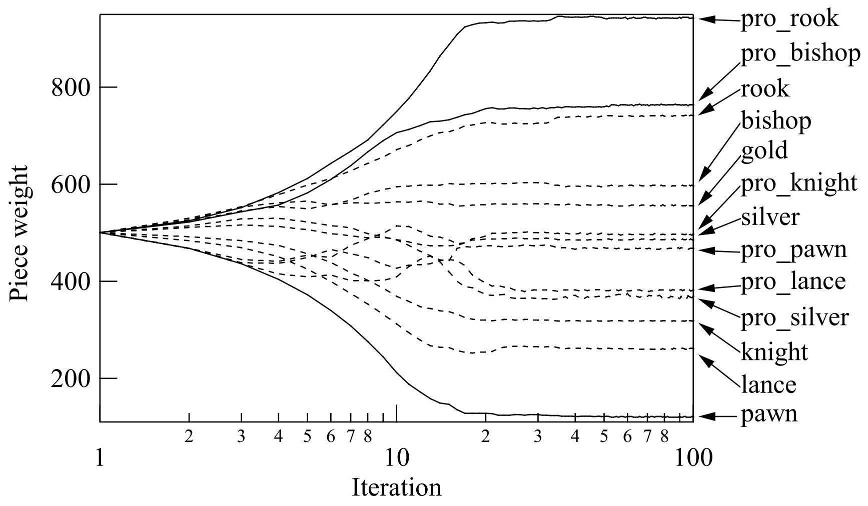

Figure 4: Results of MMTO for the piece weights.

move pairs after removing duplications and handicap games from the $47,566$ game records. The test set had $103,105$ desired moves after removing duplications and handicap games from another $1,000$ game records. The feature weights for e^{AB} were adjusted with MMTO and comparison training and its intuitive modifications. The same initial feature weights $(\boldsymbol{w}^{\mathrm{A}}(0), \boldsymbol{w}^{\mathrm{B}}(0))$ were used in all three methods; $\boldsymbol{w}^{\mathrm{A}}(0)$ was optimized by MMTO from the previous experiment, and $\boldsymbol{w}^{\mathrm{B}}(0) = 0$. After that, to show scalability, the feature weights for $\mathrm{e}^{\mathrm{ABC}}$ and $\mathrm{e}^{\mathrm{ABCD}}$ were optimized by MMTO in this order. To adjust the feature weights for $\mathrm{e}^{\mathrm{ABC}}$, the optimized feature weights of e^{AB} were used for the initial feature weights $\boldsymbol{w}^{\mathrm{A}}(0)$ and $\boldsymbol{w}^{\mathrm{B}}(0)$, and 0 was used for $\boldsymbol{w}^{\mathrm{C}}(0)$. Similarly, in $\mathrm{e}^{\mathrm{ABCD}}$, the optimized feature weights in $\mathrm{e}^{\mathrm{ABC}}$ were used for the initial feature weights $\boldsymbol{w}^{\mathrm{A}}(0)$, $\boldsymbol{w}^{\mathrm{B}}(0)$, and $\boldsymbol{w}^{\mathrm{C}}(0)$, and 0 was used for $\boldsymbol{w}^{\mathrm{D}}(0)$.

Comparison training with $\mathrm{e}^{\mathrm{ABC}}$ and $\mathrm{e}^{\mathrm{ABCD}}$ was not tested because learning e^{AB} yielded only small improvements. The rate r in Eq. (10) was hand tuned to 0.031. As an example of the intuitive modifications to stabilize the iterative procedure, a constant-step update was also tested for learning e^{AB}. In this case, the training rate $r'(t)$ was substituted for r and

$$r'(t) = r_{\mathrm{c}} / |\nabla_{\boldsymbol{w}(t)} J^{\mathcal{P}}_{\mathrm{ct}}(\boldsymbol{w}(t))|,$$

where this constant-step modification conservatively updated $\boldsymbol{w}$ by using a constant step r_{c} that was hand tuned to $1,000$. Each value of r and r_{c} was the best of five trials. Another intuitive modification was the reuse of PV, as explained in Section 3.5, where the same PVs were used 32 times and the rate r in Eq. (10) was 0.01. The rescaling factor R in Eq. (3) was set to 0.0025, because this value was satisfactory in the previous experiment shown in Figure 3. Although the three methods are different, their iterations consumed almost the same amount of time. This is because the most time-consuming step of these experiments was the game tree search to identify the PV leaf positions.

The rate of agreement with the test set is shown in Figure 5. Here, agreement means that the legal move that obtained the highest value in the tree search is the desired move. The

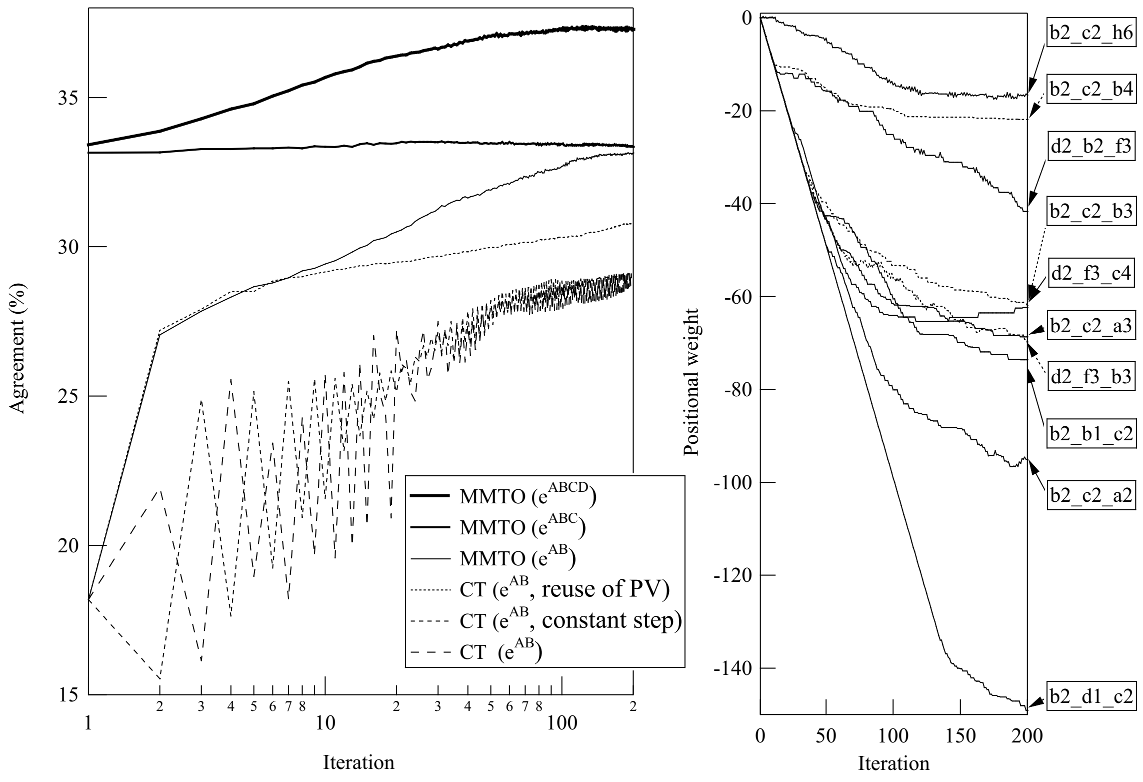

Figure 5: (Left panel) Improvement in rate of agreement with the test set for MMTO and comparison training (CT). (Right panel) Improvement in feature weights for positional features in e^D. Feature weight b2_c2_b4 indicates the black king is at `b2` and two gold generals are at `c2` and `b4`. Similarly, feature weights b2_c2_h6, b2_c2_b3, b2_c2_a3, b2_b1_c2, b2_c2_a2, and b2_d1_c2 indicate two gold generals with the king at `b2`. Feature weight d2_b2_f3 indicates the black king is at `d2` and the opponent's two gold generals are at `b2` and `f3`. Similarly, feature weights d2_f3_b3 and d2_f3_c4 indicate the opponent's two gold generals with the king at `d2`. Here, each value has been divided by 2^5.

rate is calculated by excluding positions in which there is only one legal move, or positions in which there is an easy checkmate sequence that can be identified in the shallow-depth search. Tied values are not counted, either.

The performances of MMTO, comparison training, and its variations were compared in the case of learning e^{AB}. We can see from the figure that the agreement rates of comparison training and constant-step modification are unstable and substantially lower than that of MMTO. We can also see that the reuse-of-PV modification increases stability because it reduces the step length from 0.031 to 0.01 and reduces the computation time of learning by almost 32 times because it reduces the number of time-consuming PV updates.

MMTO with the full evaluation function e^{ABCD} had the highest rate (37%). The large-scale optimization of the weight vector w^D increased the level of agreement in 200 iterations.

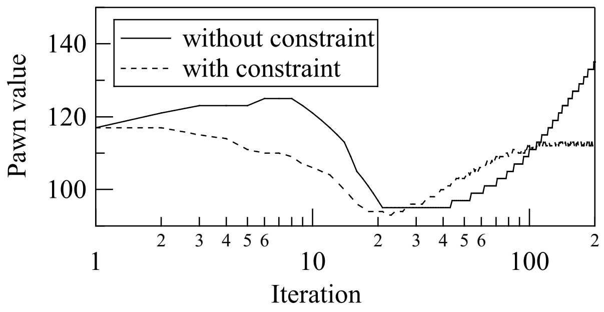

Figure 6: Effect of the constraint term in MMTO (e^{AB}).

This computation took about a week using an Intel X5690 workstation. The agreement ratio with the test set converged in 100 iterations. However, the feature weights did not converge.

Figure 6 shows how the constraint $J_C(\boldsymbol{w})$ in Eq. (9) improves the stability of MMTO in response to pawn-value changes during the e^{AB} learning. We can see that the value keeps on increasing when $J_C(\boldsymbol{w})$ is turned off and that it converges in 100 iterations with the constraint turned on. One of the feature weights overflowed in the comparison training of e^{ABC}, and this is another reason why the results of e^{ABC} are not shown for comparison training. As the regularization term $J_R(\boldsymbol{w})$ has little effect on the learning of e^{AB}, the improvement in the agreement rates of MMTO is mainly due to the use of the constraint $J_C(\boldsymbol{w})$ and grid-adjacent updates.

The regularization term $J_R(\boldsymbol{w})$ is important for optimizing larger weight vectors. Figure 7 shows how $J_R(\boldsymbol{w})$ in Eq. (4) improves the weight vector in the enlarged evaluation function e^{ABCD}. Without a regularization term, the objective function value and the rate of agreement with the training set increase with the number of iterations. However, there is also a linear increment in the absolute value of the weight vectors, and it distorts the rate of agreement with the test set after the 50-th iteration. After the 200-th iteration, only 0.2% of the components in $\boldsymbol{w}$ are zero. On the other hand, 96.3% of the components in $\boldsymbol{w}$ are zero with the regularization term. These results indicate that MMTO without the regularization suffers from overfitting of the training set when a large-scale weight vector is used. A similar effect of regularization also occurs when MMTO is used in the e^{ABC} learning, though the effect is smaller than that of e^{ABCD}.

4.4 Improvements in Strength

To analyze the relationship between the agreement rate and the strength, we had the programs learned by MMTO and by comparison training (Figure 5) play games against a reference shogi program many times. The reference program was a version of GPS Shogi released in 2008 (Kaneko, 2009). It is an open source program and was a finalist in past world computer shogi championships. It has a completely different evaluation function in which the majority of the parameters have been hand tuned. This version of GPS Shogi serves as a reference program on a popular game server for shogi programs[4]. The matches were as follows: the reference program ($4 \cdot 10^5$ nodes/move) vs. all four learned programs,

4. `http://wdoor.c.u-tokyo.ac.jp/shogi/`, last access: 2013 (in Japanese).

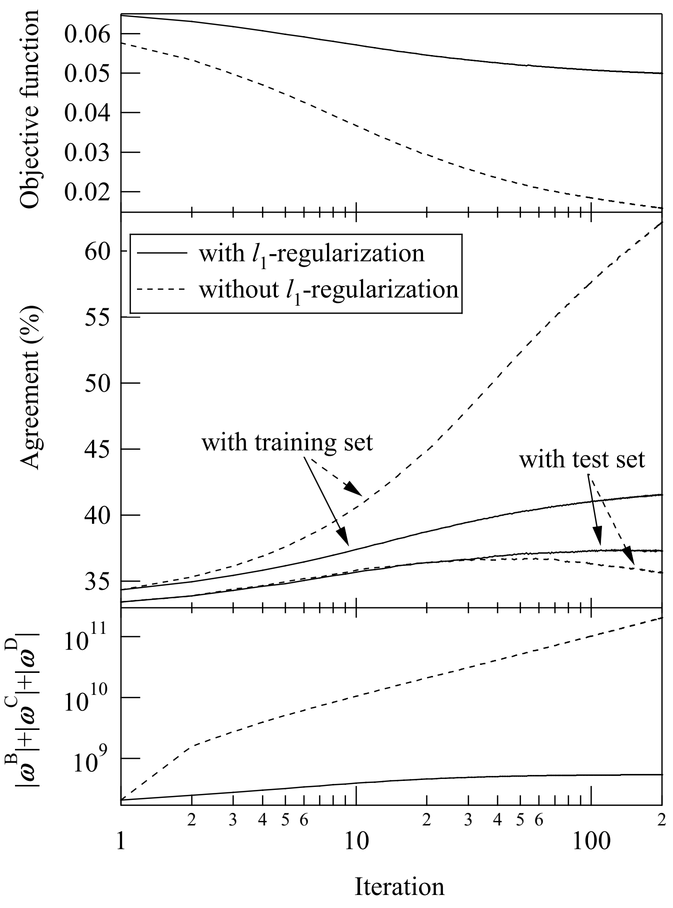

Figure 7: Effect of the regularization term in MMTO (e^{ABCD}).

the reference program ($2 \cdot 10^5$ nodes/move) vs. the two learned programs with the evaluation function e^A and e^{AB}, and the reference program ($8 \cdot 10^5$ nodes/move) vs. the two learned programs with the evaluation functions e^{ABC} and e^{ABCD}. 500 games were played in each match. The weight vectors obtained after $1, 2, 4, 8, 16, 32, 48, 64, 80, 96, 112, 128,$ and 144 iterations were tested for each learning configuration. Thus, a total of $8 \cdot 13 \cdot 500 = 52,000$ games were played. The learned programs searched $4 \cdot 10^5$ nodes per move. All programs ran on a single thread and searched similar numbers of nodes in a second.

We measured the playing strength in terms of the Elo rating, which is a popular way to represent relative strength in two-player games. The winning probability between two players is estimated as $1/(1 + 10^{d/400})$, where d is the difference between their ratings. For example, if the rating of player A is higher than that of player B by 150, the winning percentage of player A is about 70%. Here, the ratings were determined by using maximum likelihood estimation on all the games.

Figure 8 shows the Elo rating of each player. We can see that MMTO with e^{AB} significantly outperformed comparison training with the same initial feature weights. When MMTO used e^{AB}, the winning percentage against the reference (400k/move) stably increased from 11.2% (1,800 Elo points) to 59.4% (2,210 Elo points). In contrast, comparison

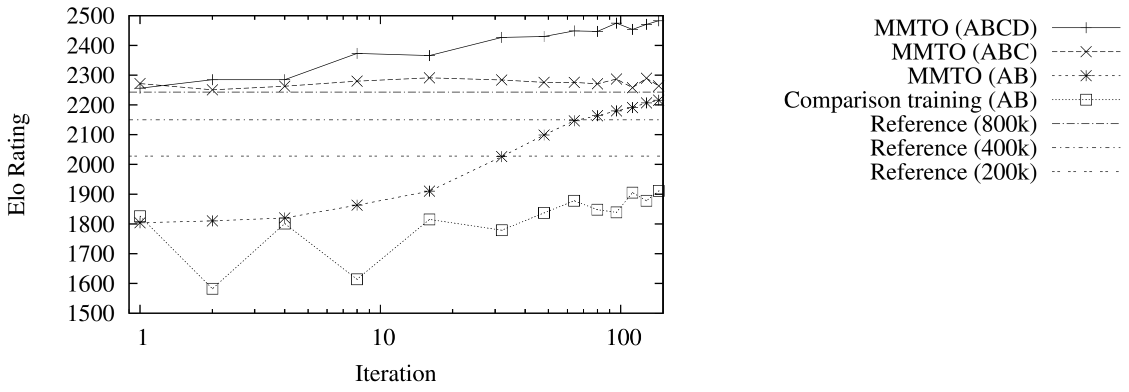

Figure 8: Improvements in strength (Elo rating) achieved by MMTO and comparison training.

Opponent	Depth 2	Depth 3	Depth 4	Depth 5	Depth 6	Depth 7	Depth 8
Player 1	$94 \pm 2\%$	$89 \pm 3\%$	$82 \pm 3\%$	$85 \pm 3\%$	$78 \pm 3\%$	$81 \pm 3\%$	$78 \pm 3\%$
Player 2	$77 \pm 3\%$	$75 \pm 3\%$	$77 \pm 3\%$	$82 \pm 3\%$	$81 \pm 3\%$	$85 \pm 3\%$	$84 \pm 3\%$

Table 3: Winning percentages of program learned with game tree search having various depths. Opponent player 1 is the same program but the search depth is reduced by 1, and opponent player 2 is also the same program but it uses the weight vector before the learning.

training won at most 19.5% (1,910 Elo points) of its games. The results shown in Figs. 5 and 8 indicate that MMTO outperforms comparison training.

The large number of features also contributed to the playing strength of the programs learned by MMTO. Although e^{ABC} showed a small improvement in terms of the agreement rate and Elo rating, e^{ABCD} consistently yielded significant improvements in these two criteria. Thus, we concluded that MMTO scales well to forty million features. Note that the computational cost of e^{ABCD} was reasonably small for practical game play. This is because the number of features that appear in a position is only 2,800 or less even when the total number of features is about forty million. Also, the summations in Table 2 can be maintained in an incremental manner when the program makes or unmakes a move. This sort of feature design is similar to that of a famous Othello program (Buro, 2002). As a result, Bonanza using e^{ABCD} searched about $3 \cdot 10^6$ nodes/sec on an Intel Xeon X5680 with 12 threads. The speed itself is slower than that of many chess programs, but about average for strong shogi programs. In addition, we found that Bonanza using e^{ABCD} trained by MMTO played better than or was comparable to any of the top shogi programs in actual tournaments. The details are discussed in Section 4.6.

Two additional fixed-depth self-play experiments were conducted to see if evaluation functions trained by using shallow searches (depth 1 with quiescence search) are effective on deep searches. Table 3 shows the winning percentages of the learned program with various search depths of game play. The learned program had the $\mathrm{e}^{\mathrm{ABCD}}$ evaluation function yielded after the 200-th iteration (Figure 5). The winning percentages against the same program (player 1) with the search depth reduced by 1 were around 80%. Thus, we see that the deeper the learned program searched, the stronger the program was. Tesauro (2001) also reported similar results by using comparison training. In addition, the winning percentage was about 80% against a program (player 2) that searched to the same depth but used $\mathrm{e}^{\mathrm{ABC}}$ after the 200-th iteration. Thus, the use of $\mathrm{e}^{\mathrm{ABCD}}$ trained by 200 iterations was effective even when the program searched deeper. Here, the winning percentages were computed for a thousand games (Seventy-six games or less ending in draws or exceeding 300 moves were not counted). Fifty megabytes of memory were assigned to the transposition table of each program. The uncertainties indicated as ± 3 was estimated by conducting a two-sided test at a significance level of 5% on the one-thousand games.

4.5 Numerical Stability and Convergence

We investigated the continuity and partial differentiability of the objective function and convergence of the feature weights in an empirical manner. While forward pruning techniques in game tree searches can speed up MMTO in practical applications, such methods do not always maintain continuous search values, as is shown in Appendix A.4. Moreover, the objective function contains a large number of search values. This means it is difficult to estimate its properties in a theoretical manner.

To make the empirical investigation manageable, we used only the smallest evaluation function e^{A} that deals with thirteen shogi piece values. Moreover, we reduced the number of game records to $1,000$; the game records had $98,224$ desired moves and $Z^{\mathcal{P}} = 7,900,993$ move pairs after removing duplications and handicapped games.

4.5.1 Surface of the Objective Function

We investigated the function surface of the main part of the objective function $J(\mathcal{P}, \boldsymbol{w}^{\mathrm{A}})$ of MMTO in Eq. (4) by generating contour maps from millions of sampling vectors $\boldsymbol{w}$. Note that a contour line (isovalue surface) is a curve along which the functions take the same value. The contour lines have certain properties: the gradient of the function is perpendicular to the lines, and the magnitude of the gradient is large when two lines are close together. In addition, a closed-loop contour line indicates the location of the local minimum or maximum.

Two of the thirteen piece values in the weight vector $\boldsymbol{w}^{\mathrm{A}}$ were sampled in order to draw the contour maps of two-dimensional functions at an interval of 5 for each piece value. The remaining eleven pieces were assigned reasonable values; 118 (pawn), 273 (lance), 318 (knight), 477 (silver general), 562 (gold general), 620 (bishop), 734 (rook), 485 (promoted pawn), 387 (promoted lance), 445 (promoted knight), 343 (promoted silver), 781 (promoted bishop), and 957 (promoted rook). The constraint term $J_{\mathrm{C}}(\boldsymbol{w}^{\mathrm{A}})$ was ignored so that $\boldsymbol{w}^{\mathrm{A}}$ could be freely changed and the regularization term $J_{\mathrm{R}}(\boldsymbol{w})$ was turned off for piece values. A nominal depth 1 search, together with the quiescence search, was used.

We analyzed two pairs: $\langle$gold, bishop$\rangle$, and $\langle$pawn, promoted_lance$\rangle$. Figure 9 shows an enlargement of the contour map of $J(\mathcal{P}, w^{\mathrm{gold}}, w^{\mathrm{bishop}})$. The contour interval is $5 \cdot 10^{-4}$. The map was computed with the ranges of $[200, 1100]$ for the bishop and $[100, 930]$ for

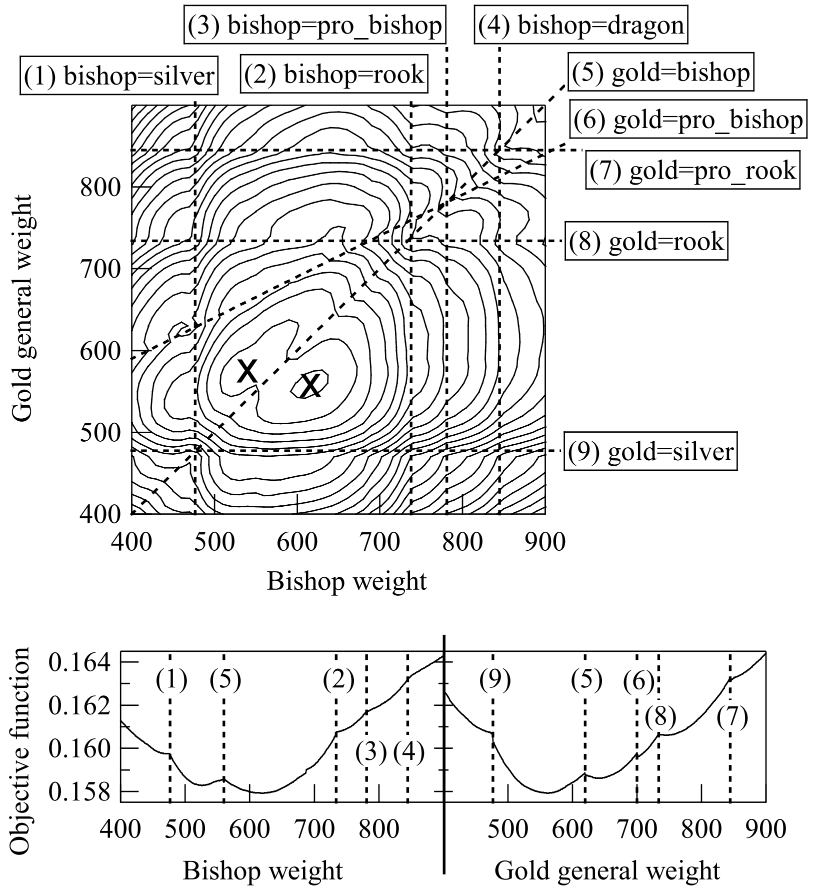

Figure 9: (Upper panel) Enlarged contour map of $J(\mathcal{P}, w^{\text{gold}}, w^{\text{bishop}})$. The dashed lines indicate the critical boundaries at which the two-dimensional function is not partially differentiable. The two minima are indicated by x. (Bottom panel) Cross sections of the contour map. The left one shows the intersection of the map with the line $w^{\text{gold}} = 560$, and the other shows that of $w^{\text{bishop}} = 620$.

the gold general. Note that the function simply increases and there are no interesting structures outside of the enlarged map. Figure 10 shows an enlargement of the contour map of $J(\mathcal{P}, w^{\text{pawn}}, w^{\text{pro\_lance}})$. The contour interval is $1 \cdot 10^{-3}$. The map was computed with the ranges of $[10, 500]$ for a pawn and $[200, 700]$ for a promoted lance.

We can see from these maps that there are local minima within reasonable ranges and no sudden changes in the function values. Although the function depends on a large number of empirical search values, $\mathrm{s}(p, \boldsymbol{w})$, it is approximately continuous and amenable to optimization on the basis of gradients approximated by MMTO.

On the other hand, the maps illustrate three difficulties. The first difficulty is the clear edges of the contour lines. They indicate that the function is not partially differentiable at the points on these edges. The dashed lines in these maps are critical boundaries at which the profit and loss ratio of material exchanges inverts itself. For example, a silver is usually

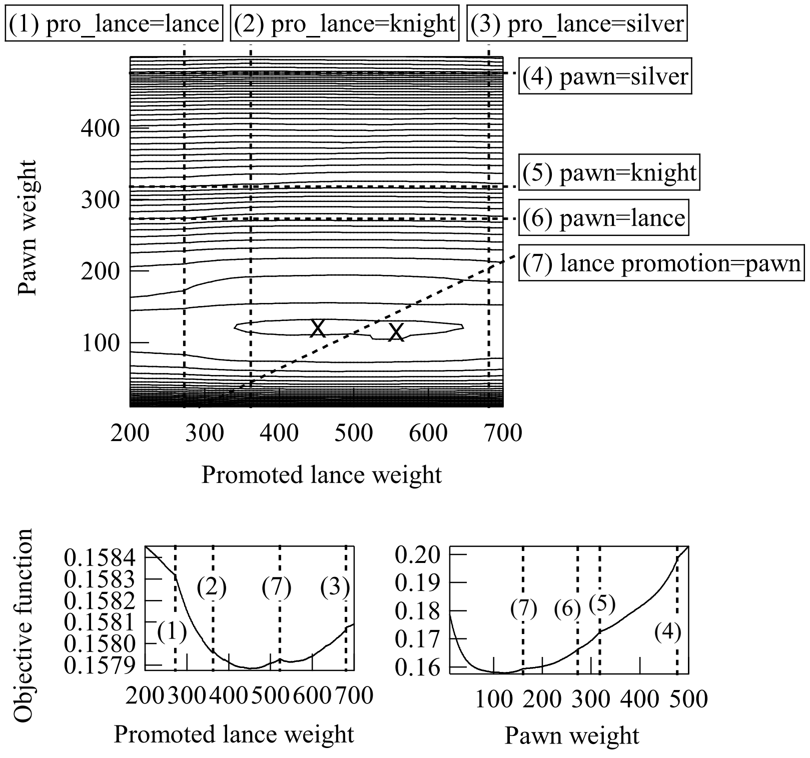

Figure 10: (Upper panel) Enlarged contour map of $J(\mathcal{P}, w^{\text{pawn}}, w^{\text{pro\_lance}})$. The dashed lines indicate critical boundaries at which the two-dimensional function is not partially differentiable. The two minima are indicated by x. (Bottom panel) Cross sections of the contour map. The left one shows the intersection of the map with the line of $w^{\text{pawn}} = 125$, and the other shows that of $w^{\text{pro\_lance}} = 450$.

less valuable than a bishop, but capturing a silver becomes more profitable than capturing a bishop when the bishop value is smaller than 477. This boundary is labeled "bishop=silver" in Figure 9. As discussed in Appendix A, the function is not always partially differentiable at these critical boundaries, where multiple moves share the same best value. Note that there can be more boundaries in theory, e.g., a "bishop=promoted knight" boundary. Whether a boundary is visible or not depends on the training set and evaluation features. In addition, the boundaries become winding curves when a non-linear evaluation function is used instead of a linear weighted sum.

The second difficulty, the scaling problem, is illustrated in Figure 10. In this map, we can see that the scales of the two piece values differ by two orders of magnitude. That is, a pawn-value variation of five hundred changes the function value by 0.04, whereas a promoted-lance-value variation of five hundred changes the function value by only $4 \cdot 10^{-4}$. Because of the difference in scaling, the surface along a promoted lance is almost flat. This property explains why the pawn value is optimized earlier than those of the other pieces in comparison training, as shown in Figure 3. This property of ill-scaling is disadvantageous when it comes to optimizing the promoted-lance value using a naive gradient decent method.

Methods based on second-order partial derivatives or approximations of the Hessian matrix can resolve this problem; however, they behave poorly at non-partially differentiable points on many boundaries. These two difficulties point to why the grid-adjacent update in MMTO is effective.

The third difficulty is that there are multiple local minima in the two maps. This means the results of MMTO depend on the initial values and there is a chance of ending up with a local rather than global minimum. We will investigate this problem in the next subsection 4.5.2.

4.5.2 Empirical Convergence and Local-Minima Properties

In the previous subsection, we examined two-dimensional cross sections of the function $J(\mathcal{P}, \boldsymbol{w}^{\mathrm{A}})$. In this subsection, we loosen the restriction from two to thirteen dimensions, which is sufficiently large to express all piece values in shogi. The aim of this experiment is to catch a glimpse of the global map and numerical convergences for arbitrary initial guesses about the values of all of the pieces.

For this purpose, a Monte Carlo sampling of the initial guess, $\boldsymbol{w}^{\mathrm{A}}(0)$, was carried out to enumerate the local minima and analyze the optimized vectors. We ran 444 MMTO with randomized initial values. Here, a uniformly distributed integer in the range of $[0, 32767]$ was assigned to each vector component, and the resulting vector was scaled to satisfy the equality condition $g(\boldsymbol{w}^{\mathrm{A}}) = 0$.

Figure 11 shows the cosine similarity and objective-function value of a hundred of the 444 runs. Here, the cosine similarity of a weight vector is measured relative to the best vector whose objective function is the smallest among those of 444 vectors after 100 iterations. In the majority of the runs, we can see that function values and weight vectors converged numerically in 50 iterations. Here, we regard the iteration procedure to have converged when the function values and similarities oscillate and show neither steady increase nor decrease from the 50-th to 100-th iteration. Although convergence is almost assured for MMTO with thirteen piece values, it would be difficult to achieve if more feature weights were to be optimized. For example, Figure 5 shows there was no convergence after two-thousand iterations using $\mathrm{e}^{\mathrm{ABCD}}$. Because 200 iterations took about a week on an Intel X5690 workstation, we could not afford to investigate the convergence of $\mathrm{e}^{\mathrm{ABCD}}$ with the current hardware. However, 200 iterations nonetheless achieved a significant improvement in strength, as shown in Figure 8.

We can also see that these trials of MMTO ended up with multiple local minima. Although a multiplicity of minima is generally undesirable in an optimization, there were other, more favorable properties. The first property is that each run of MMTO changed the weight-vector components by a sufficient amount. That is, the cosine similarity of the 444 optimized vectors was localized in the range of $[0.925, 1]$, while that of the random initial vectors were widely spread (see the top panel of Figure 12). The second property is that there was a weak correlation between the cosine similarities of the initial and optimized vectors. This means that starting from a better initial vector in terms of the cosine similarity should be beneficial (see the top panel of Figure 12). However, starting from a better initial vector in terms of the objective function value is not beneficial (see the middle panel of Figure 12). The third is that the distribution of local minima formed structures (see the

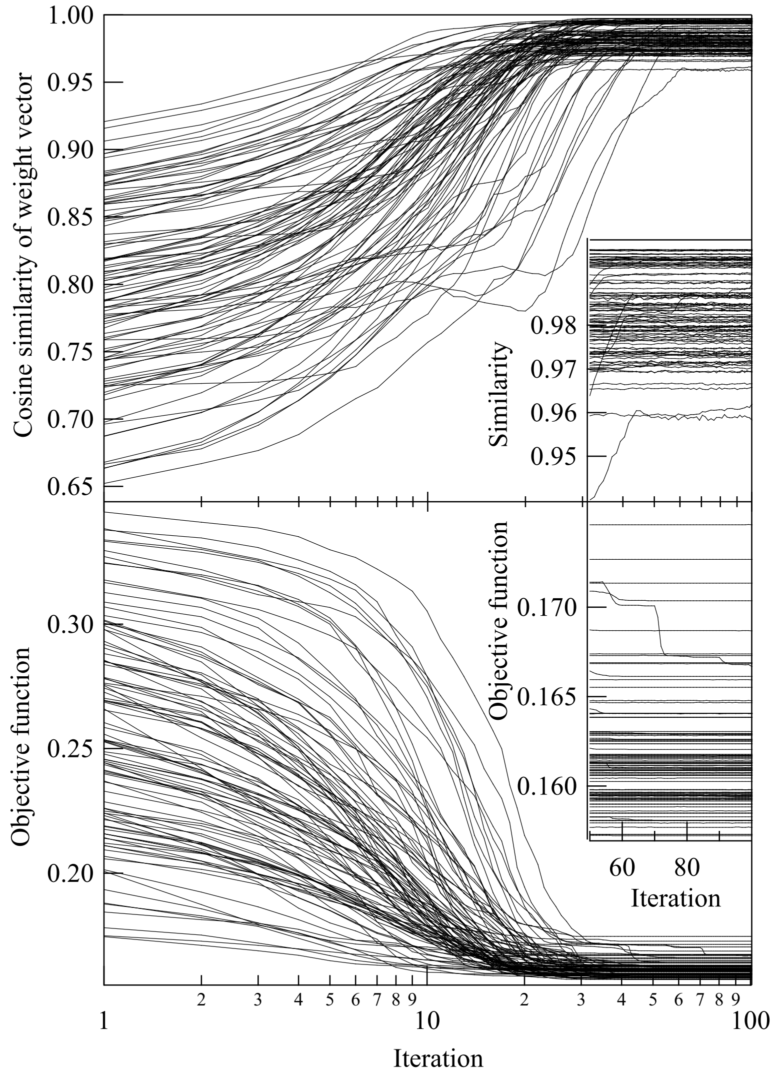

Figure 11: A hundred runs of MMTO for a weight vector $\boldsymbol{w}^{\mathrm{A}}$ consisting of thirteen piece values. The initial vectors were set using pseudo-random numbers. The inset is an enlargement showing the appearance of the numerical convergences. The top panel shows the cosine similarities relative to the best weight vector. The bottom panel shows the values of the objective function.

bottom panel of Figure 12). That is, the lower the local minimum is, the more similar it becomes to the best vector. Moreover, the number of local minima decreases as the weight vector gets farther away from the best.

We also investigated the dependence of the performance on the nominal search depth of step (1) shown in Figure 2. Similar results in terms of convergence and the distribution of local minima were obtained using a deeper search with a nominal depth of 2. Because MMTO with a depth of 2 consumes more time than MMTO with a depth of 1, the number

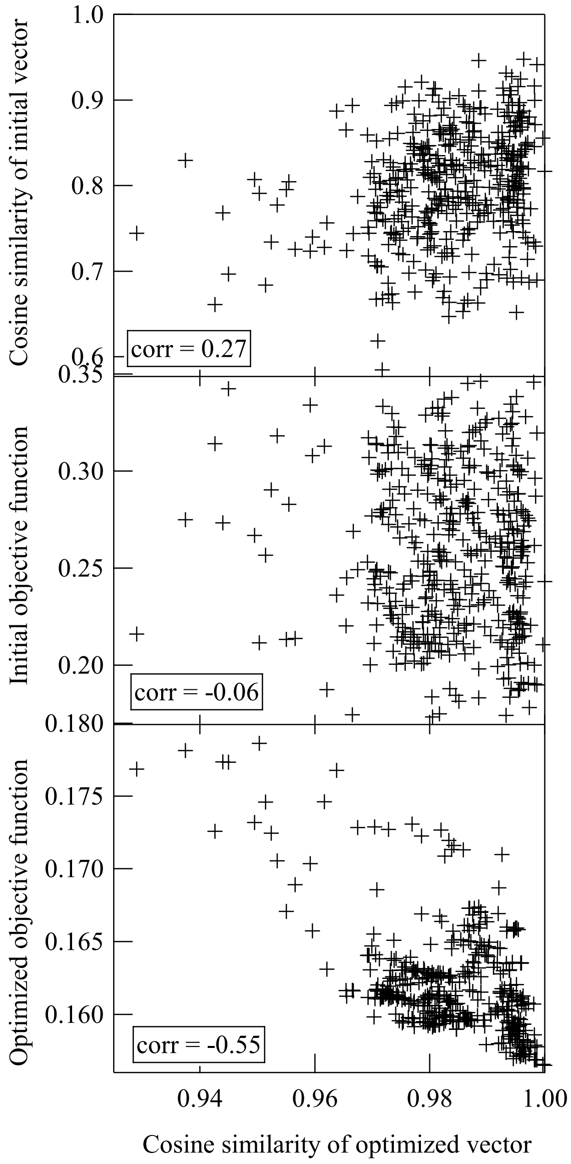

Figure 12: Scatter plots for 444 trials of thirteen-dimensional weight vectors. The vector expresses thirteen piece values. The cosine similarity of the vector is measured relative to the best vector. The initial vector consists of uniform pseudo-random numbers, and the optimized one is the 100-th vector of the MMTO iterations starting from the initial one. The inset shows the correlation coefficient of each scatter plot.

of random initial vectors was reduced to 78, and the number of iterations was reduced to sixty for the sake of speed. In the majority of runs, the function values and weight vectors converged in 50 iterations. Figure 13 shows the strength (Elo rating) and objective-

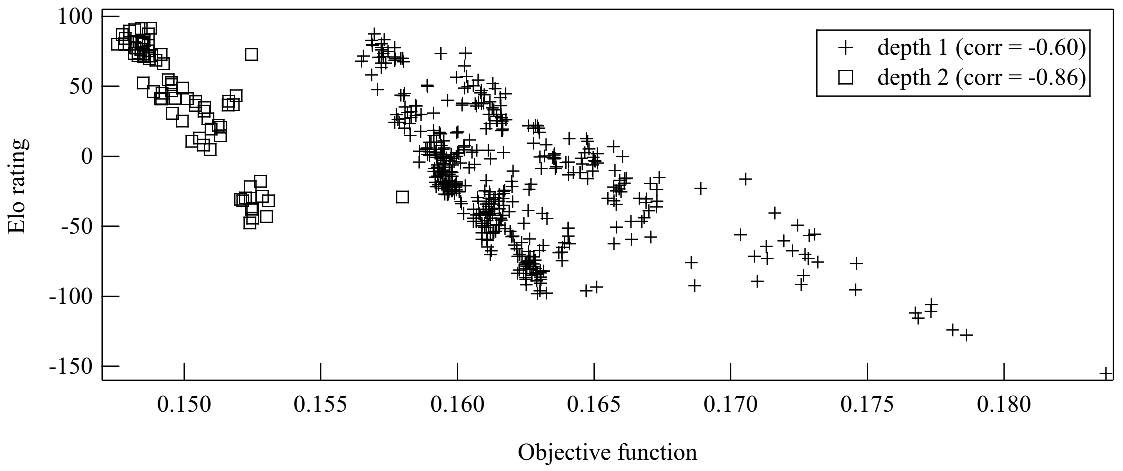

Figure 13: Scatter plots for thirteen-dimensional weight vectors. The 444 vectors indicated by crosses were learned with the nominal depth 1 search of step (1), and the 78 vectors indicated by squares were learned with a depth 2 search.

function value of the 78 runs with depth 2 (squares) and 444 runs with depth 1 (crosses). Here, the Elo ratings were identified by using maximum likelihood estimation on 894, 244 random-pairing games ($5 \cdot 10^4$ nodes/move). The Elo rating with depth 1 was -17 on average and that with depth 2 was 41 on average. Also, the correlation coefficient between the Elo rating and objective function value with depth 2 was -0.86 and that with depth 1 was -0.60. Moreover, we compared the performance of two best vectors that gave the smallest objective function values. Here, we computed the winning probability between the best results of depth 1 and 2. Each player was allowed to use one second for each move, and one core of an Intel Xeon X5680 and fifty megabytes of memory were assigned to the transposition table. After excluding two drawn games and two games exceeding a thousand moves, we obtained a 43.6% winning rate against the program using the best results of depth 2. These results indicate that MMTO is better with depth 2 than with depth 1.

4.6 Performance of MMTO under Tournament Conditions

MMTO was invented by the developer of Bonanza and made it one of the best programs in shogi. Moreover, the ideas behind earlier versions of MMTO published in Japanese (Hoki, 2006) have been adopted by many developers and have dramatically changed shogi programs.

One of the authors started developing Bonanza in 2004, published program files on the web in 2005, and published source codes on the web in 2009 (Hoki, 2013). This paper gives detailed descriptions of the evaluation-function learning, whereas the literature (Hoki & Muramatsu, 2012) gives detailed descriptions of the game-tree pruning of Bonanza. In addition to the learning method MMTO, Bonanza uses the evaluation function e^{ABCD} shown in Table 2. The earlier versions until 2009 used a subset of e^{ABCD} with a modified

	2006 May	2007 May	2008 May	2009 May
1	Bonanza**	YSS	Gekisashi	GPS Shogi**
2	YSS	Tanase Shogi*	Tanase Shogi*	Otsuki Shogi*
3	KCC Shogi	Gekisashi	Bonanza**	Monju***
4	TACOS	Bonanza**	YSS	KCC Shogi*
5	Gekisashi	Bingo Shogi	Bingo Shogi	Bonanza***

	2010 May	2011 May	2012 May	2013 May
1	Gekisashi*	Bonkras**	GPS Shogi**	Bonanza***
2	Shueso*	Bonanza***	Puella α**	ponanza**
3	GPS Shogi**	Shueso*	Tsutsukana*	GPS Shogi**
4	Bonkras**	Gekisashi*	ponanza**	Gekisashi*
5	Bonanza Feliz***	ponanza**	Shueso*	NineDayFever**

Table 4: Program names and results of the recent World Computer Shogi Championship. ***MMTO, **an earlier version or a variant of MMTO, or *a learning method influenced by MMTO is used.

l2-regularization (Hoki, 2006). Subsequent versions fully evaluate e^{ABCD} learned with l1-regularization.

Table 4 shows the results of the World Computer Shogi Championships. Since 2006, the performance of Bonanza has been examined in several computer shogi tournaments, where each participant connects to a server program and plays shogi under a time control of 25 minutes a side. Bonanza received the first prize twice, second prize once, and third prize once. Moreover, players entitled Bonanza Feliz and Monju used the same evaluation functions as obtained by MMTO. Thus, we claim that when Bonanza uses MMTO, it plays better than or is comparable to any of the top programs in shogi, including commercial ones. This method clearly plays at the level of handcrafted shogi programs. Moreover, descriptions of the learning shogi evaluation functions and the earlier version of MMTO were published by Hoki (2006) in Japanese and were quickly recognized as significant advances. In fact, no shogi program with conventional handcrafted evaluation functions has broken into the top five in during the last five years of tournaments. One interesting case is the results of GPS Shogi (Kaneko, 2009), the winner of the 2009 and 2012 tournaments, and source codes are available online (Tanaka and Kaneko, 2013). From 2003 to 2008, this program uses a handcrafted evaluation function but in 2009 it used a variant of MMTO and its results dramatically improved. The variants of MMTO used in each program differ in accordance with the content and policy of each program. For example, Tanase Shogi, the runner-up program in 2008, used a learning method based on MMTO and handcrafted evaluation functions. Bonkras, ponanza, Puella α, and NineDayFever also used variants of MMTO. These excellent results make it clear that MMTO outperforms conventional programs that use handcrafted evaluation functions and has played extremely well in recent shogi tournaments.

It should be noted that some versions of Bonanza add a small amount of randomness to the grid-adjacent updates. However, we omitted any discussion of using randomness in this paper because it is not clear whether the added randomness improved the quality of the evaluation function or not. The source codes of various versions of Bonanza are available online (Hoki, 2013) and the source code of MMTO are in two files, learn1.c and learn2.c.

4.7 Preliminary Experiments on Chess

So far, we have discussed the performance of MMTO in shogi. We expect that MMTO would be effective in other two-player perfect information games provided that certain conditions are met: (1) a sufficient number of game records are available, (2) minimax searches guided by heuristic evaluations are effective, and (3) the analytic partial derivatives of the evaluation function with respect to the variables are available. For example, MMTO would not yield interesting results were it to be applied to a game that has been solved by other means (e.g., van den Herik, Uiterwijk, & van Rijswijck, 2002). Also, it would not yield interesting results in the game of Go because Monte-Carlo tree searches are more effective than minimax searches guided by a heuristic evaluation function (Kocsis & Szepesvari, 2006; Gelly & Silver, 2011; Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, & Colton, 2012; Gelly, Kocsis, Schoenauer Sebag, Silver, Szepesvári, & Tẽytaud, 2012). Moreover, a simpler learning method (e.g., a regression method in Othello, Buro, 2002) would be preferable to MMTO, if it is sufficiently effective.

We conducted preliminary experiments on chess to catch a glimpse of the applicability of MMTO to other games. Note that there already are evaluation functions in chess that can outplay grandmasters, whereas there are none in shogi. Thus, it might be difficult to improve well-crafted chess evaluation functions. For this experiment, we chose an open-source program (Crafty) as a fair implementation of a chess program (Hyatt, 2013). The original evaluation function had been tightly tuned and is not a simple multivariable function. Thus, for the sake of simplicity, we did not modify it in any way except to add a new linear combination of weighted two-pieces-square features. The features were used to evaluate all conditions of the king and another piece, such as in e^{B} in Section 4.1. The mirror symmetric property described in Section 4.1 was not applied and features in which a pawn exists at the eighth rank were not counted. As a results, the total number of added weights $\boldsymbol{w}^{\mathrm{B'}}$ was $39,312$. Because a chess position possesses only thirty or fewer two-pieces-square features, the additional computational time due to the above modification became almost negligible with the help of a pawn hash table and the lazy evaluation technique that had come with the original.

The training and test sets were composed by using game records at the Free Internet Chess Server (FICS). These games were played using the standard time control of the server by two players with ratings of $2,600$ or more. The training set $\mathcal{P}$ had $1,267,032$ desired moves and $Z^{\mathcal{P}} = 33,619,904$ move pairs after removing duplications from the $13,440$ game records, whereas the test set $\mathcal{P}$ had $101,982$ desired moves and $Z^{\mathcal{P}} = 2,755,217$ move pairs after removing duplications from the $1,000$ game records.

Figure 14 shows the rate of agreement with the test set and the number of correct answers of chess problems through iteration. Here, α in the sigmoid function was set to 0.00341, the equality constraint was not used, and the regularization term was $J_{\mathrm{R}}(\boldsymbol{w}^{\mathrm{B'}}) = 0.156|\boldsymbol{w}^{\mathrm{B'}}|$.

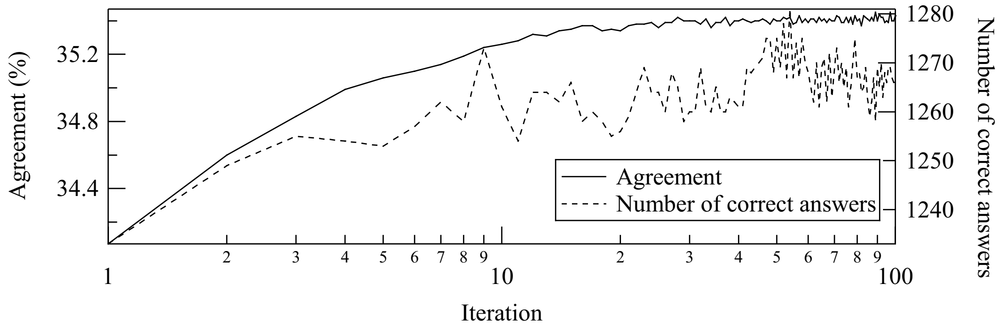

Figure 14: Improvement in rate of agreement with the test set (solid line) and the number of correct answers of 2, 180 problems (dashed line) in chess. The two-piece-square weights $\boldsymbol{w}^{\mathrm{B'}}$ were adjusted using MMTO.

Rating	1010–1279	1280–1489	1490–1769	1770–2049	2050–
Win	$33 \pm 3\%$	$35 \pm 3\%$	$39 \pm 4\%$	$43 \pm 4\%$	$42 \pm 4\%$

Table 5: Dependence of the strength (winning percentages) of learned programs on the quality (ratings of players) of the training set. The uncertainty indicated as ± 3 was estimated by conducting a two-sided test at a significance level of 5% on 1, 000 games.

A total of 2, 180 chess problems from the Encyclopedia of Chess Middlegames (the second section of the 879 problems), Win at Chess (300 problems), and Winning Chess Sacrifices (1, 001 problems) were used (Krogius, Livsic, Parma, & Taimanov, 1980; Reinfeld, 2001, 1969). The learned program searched $5 \cdot 10^4$ nodes per problem and eight megabytes of memory were assigned to the transposition table. We see that the agreement rate as well as the number of correct answers tends to improve as the number of iterations grows, though the differences are moderate. It means that MMTO found room for improvement in a well-implemented chess program. These results indicate that MMTO can be a useful way to learn heuristic evaluation functions in chess, especially when one can design evaluation features suitable for learning.

4.8 Data Quality Dependence

To assess the importance of the quality of the game records, we conducted additional experiments using game records of players with various levels of experience in shogi. Here, $\mathrm{e}^{\mathrm{ABCD}}$ was learned by using the results of $\mathrm{e}^{\mathrm{ABC}}$ in Figure 5 as the initial value. The results are summarized in Table 5. Each training set was composed from the records of 47, 566 rapid time control (30 seconds per move) games played by amateurs on a popular Internet shogi site, Shogi Club 24[5]. The first line in the table shows the ratings of the amateur players. The second line shows the winning percentages of the learned evaluation function

5. Shogi Club 24, `http://www.shogidojo.com`, last access: 2013.

against the evaluation function trained with grandmaster-game records. Here, each evaluation function was learned in 200 iterations. The winning percentages were computed by averaging the results of a thousand games (About 15 drawn games and games exceeding 300 moves were not counted). Each player was allowed to use one second on one core of an Intel Xeon X5680 for each move, and fifty megabytes of memory were assigned to the transposition table. Table 5 shows the significance of the quality of the training set; the use of game records of stronger players made the program stronger.

5. Conclusion

We presented a method, *Minimax Tree Optimization* (MMTO), that uses game records to adjust a full set of feature weights of the evaluation function in a two-player game. The learning of MMTO has been designed so that the search results match the desired moves, e.g., the recorded moves of grandmaster games. MMTO consists of two procedures: (1) a shallow heuristic search for all training positions using the current feature weights and (2) an update guided by an approximation of the gradient of the objective function. A new combination of a simple smooth approximation of the step function and grid-adjacent updates with standard techniques, i.e., gradient guided optimization, constraints, and regularization, contributed to the scalability and stability of MMTO and led to it showing substantial improvements over existing methods.

The performance of MMTO was demonstrated in experiments on shogi, a variant of chess that has a larger number of legal moves. MMTO clearly outperformed the existing methods. In addition, the experimental results on the rate of agreement and playing strength indicate that MMTO can adjust forty million parameters. Possible future work would be automated adjustment of the step length and a theoretical convergence analysis.

Acknowledgments

We are grateful to Dr. Masakazu Muramatsu for his support of this work.

Appendix A. Notes on the Continuity and Partial Differentiability of the Minimax Value

We saw in Section 4.5 that the objective function of MMTO has a piecewise smooth surface. In this Appendix, we theoretically discuss the continuity and partial differentiability of the minimax value $v_p(\boldsymbol{w})$ with respect to $\boldsymbol{w} \in \mathbb{R}^N$, where $\boldsymbol{w}$ is the vector of parameters in the evaluation function $\mathrm{e}(p, \boldsymbol{w})$ and p is the position. The continuity of the minimax value ensures the continuity of the main part of objective function of MMTO defined in Eq. (5). The partial differentiability analysis gives conditions under which the approximation inside MMTO described in Section 3.3 is valid. We first analyze a single minimax tree, assuming that the tree is known and fixed. Then, we extend our discussion to game-tree-search methods that possibly explore different trees for different $\boldsymbol{w}$.

Definition 1. The *evaluation function* $\mathrm{e}(\cdot,\cdot)$ is a $(\mathbb{P}, \mathbb{R}^N) \mapsto \mathbb{R}$ function, where $\mathbb{P}$ is the set of all positions in a target game, $\mathbb{R}$ is the set of real numbers, and $\mathbb{R}^N$ is an N-dimensional

Euclidean space. The evaluation function $\mathrm{e}(p, \boldsymbol{w})$ is continuous with respect to the parameters $\boldsymbol{w}$ for any position $p \in \mathbb{P}$ and for any $\boldsymbol{w} \in \mathbb{R}^N$. Moreover, the evaluation function $\mathrm{e}(p, \boldsymbol{w})$ is partially differentiable with respect to any component of $\boldsymbol{w}$ at any $\boldsymbol{w} \in \mathbb{R}^N$.

The continuity and partial differentiability of the evaluation function are feasible assumptions. Note that an evaluation based on an ordinary piece-square table has these properties, and all recent machine learning of evaluation functions have them (Baxter et al., 2000; Veness et al., 2009; Buro, 2002).

Definition 2. The theoretical *game graph* $\mathcal{G}$ is a finite, directed acyclic, connected graph representing all possible transitions of states in the target game, where a node (resp. edge) represents a position (resp. move). The set of nodes in $\mathcal{G}$ corresponds to $\mathbb{P}$; $V(\mathcal{G}) = \mathbb{P}$. A minimax graph $\mathcal{T}$ is a finite connected sub-graph of $\mathcal{G}$. By convention, we use the term *minimax tree* for the minimax graph even when it is not a tree. We denote the set of minimax trees in $\mathcal{G}$ by $\mathbb{T}$. A node is called a maximizing (resp. minimizing) node if the corresponding position is the maximizing (resp. minimizing) player to move. The destination of an edge is a maximizing (resp. minimizing) node if and only if the source of the edge is a minimizing (resp. maximizing) node. We can clearly assume any node n to be a single position p, and we will denote the evaluation function as $\mathrm{e}(n, \boldsymbol{w})$.

Let $\mathcal{L}_{r,\mathcal{T}}$ be the set of leaf nodes of the entire sub-tree $\mathcal{T}_r$ of $\mathcal{T}$ and $\mathcal{T}_r$ is rooted at node r. We will omit tree $\mathcal{T}$ and use $\mathcal{L}_r$ if it is obvious. We denote the set of immediate successors (or children) at node n in tree $\mathcal{T}$ by $\mathcal{C}_{n,\mathcal{T}}$ or by $\mathcal{C}_n$. Note that $\mathcal{C}_n = \emptyset$ if n is a leaf. In the standard notation, a node (or vertex) in a graph $\mathcal{T}$ is denoted by $n \in V(\mathcal{T})$. However, in this Appendix, we will omit $V(.)$ and write $n \in \mathcal{T}$ because it is obvious.

Definition 3. A *minimax value* $v_{n,\mathcal{T}}(\boldsymbol{w})$ is a value associated with each node n in a minimax tree $\mathcal{T} \in \mathbb{T}$ and it is defined recursively by a tree structure and by a static evaluation function $\mathrm{e}(n, \boldsymbol{w})$, as follows:

$$v_{n,\mathcal{T}}(\boldsymbol{w}) = \begin{cases} \mathrm{e}(n, \boldsymbol{w}) & \text{if } n \text{ is a leaf,} \\ \max_{c \in \mathcal{C}_{n,\mathcal{T}}} v_c(\boldsymbol{w}) & \text{if } n \text{ is a non-leaf maximizing node,} \\ \min_{c \in \mathcal{C}_{n,\mathcal{T}}} v_c(\boldsymbol{w}) & \text{if } n \text{ is a non-leaf minimizing node.} \end{cases} \tag{11}$$

We will omit tree $\mathcal{T}$ and use $v_n(\boldsymbol{w})$ if it is obvious. For two minimax values a and b of a maximizing (resp. minimizing) node, we say a is better than b if $a > b$ (resp. $b < a$).

A.1 Continuity of Minimax Value

The continuity of the minimax value follows from the continuity of the evaluation function.

Theorem 4. *The minimax value $v_{n,\mathcal{T}}(\boldsymbol{w})$ is continuous with respect to $\boldsymbol{w}$ for any minimax tree $\mathcal{T} \in \mathbb{T}$ and for any $\boldsymbol{w} \in \mathbb{R}^N$. That is, $\lim_{\boldsymbol{w} \to \boldsymbol{w}^0} v_{n,\mathcal{T}}(\boldsymbol{w}) = v_{n,\mathcal{T}}(\boldsymbol{w}^0)$, or equivalently, for any $\boldsymbol{w}^0 \in \mathbb{R}^N$ and for any $\varepsilon > 0$, there exists $\delta > 0$ such that $|\boldsymbol{w} - \boldsymbol{w}^0| < \delta$ logically implies $|v_{n,\mathcal{T}}(\boldsymbol{w}) - v_{n,\mathcal{T}}(\boldsymbol{w}^0)| < \varepsilon$.*

The following assertion about the ordinary properties of the basic functions max and min and is common sense in analysis. It is rather difficult, however, to find a suitable reference containing it. We therefore give a proof that will be useful in the subsequent discussion.

Proposition 5. *Let k be a natural number and each $f_1(\boldsymbol{x}), ..., f_k(\boldsymbol{x})$ be a continuous function $\mathbb{R}^N \mapsto \mathbb{R}$. Then,* $\max_i(f_i(\boldsymbol{x}))$ *is a continuous function on $\mathbb{R}^N$. Similarly,* $\min_i(f_i(\boldsymbol{x}))$ *is a continuous function on $\mathbb{R}^N$.*

Proof. Because each $f_i(\boldsymbol{x})$ is continuous, for any $\boldsymbol{x}^0 \in \mathbb{R}^N$ and for any $\varepsilon > 0$, there exists $\delta_i > 0$ such that $|\boldsymbol{x} - \boldsymbol{x}^0| < \delta_i$ implies $|f_i(\boldsymbol{x}) - f_i(\boldsymbol{x}^0)| < \varepsilon$. Hence, if we choose $\delta = \min_i \delta_i$, then $|\boldsymbol{x} - \boldsymbol{x}^0| < \delta$ implies $|f_i(\boldsymbol{x}) - f_i(\boldsymbol{x}^0)| < \varepsilon$ for any $i = 1, \ldots, k$; that is,

$$f_i(\boldsymbol{x}^0) - \varepsilon < f_i(\boldsymbol{x}) < f_i(\boldsymbol{x}^0) + \varepsilon, \qquad \text{for any } i = 1, \ldots, k.$$

Note that $a_i < b_i$ for any $i = 1, \ldots, k$ obviously implies $\max_i a_i < \max_i b_i$. Thus, from the above inequalities we obtain

$$\max_i f_i(\boldsymbol{x}^0) - \varepsilon < \max_i f_i(\boldsymbol{x}) < \max_i f_i(\boldsymbol{x}^0) + \varepsilon, \quad \text{that is,} \quad |\max_i f_i(\boldsymbol{x}) - \max_i f_i(\boldsymbol{x}^0)| < \varepsilon.$$

This implies the continuity of $\max_i f_i(\boldsymbol{x})$. The proof is similar for $\min_i f_i(\boldsymbol{x})$. □

Let r be the root of a given tree $\mathcal{T}$. Now, we prove Theorem 4 on the basis of mathematical induction from the leaf nodes $\mathcal{L}_{r,\mathcal{T}}$ to root r. That is, at any leaf node $n \in \mathcal{L}_{r,\mathcal{T}}$, the minimax value is continuous because of the continuity of the evaluation function; $v_{n,\mathcal{T}}(\boldsymbol{w}) = \mathrm{e}(n, \boldsymbol{w})$. For an internal node n, we assume that continuity holds for any child c in $\mathcal{C}_{n,\mathcal{T}}$. This induction hypothesis and Proposition 5 ensure the continuity of $v_{n,\mathcal{T}}(\boldsymbol{w})$.

A.2 Stability of Principal Variations

In the above subsection, we showed the continuity of minimax values through the continuity of min and max functions. Here, we show that the best moves and principal variations are stable when the changes in the leaves are small enough. We analyze the stability in order to discuss partial differentiability.

Definition 6. The symbol $\mathcal{C}^+_{n,\mathcal{T}}(\boldsymbol{w})$, hereafter called the *best children*, denotes the set of such children at node n in tree $\mathcal{T}$ that have the same minimax value as that of n:

$$\mathcal{C}^+_{n,\mathcal{T}}(\boldsymbol{w}) = \{c \in \mathcal{C}_{n,\mathcal{T}} | v_c(\boldsymbol{w}) = v_n(\boldsymbol{w})\}.$$

We denote the rest of the children as $\mathcal{C}^-_{n,\mathcal{T}}(\boldsymbol{w})$; that is, $\mathcal{C}_{n,\mathcal{T}} \setminus \mathcal{C}^+_{n,\mathcal{T}}(\boldsymbol{w})$. Here, $\mathcal{A} \setminus \mathcal{B}$ denotes the set difference, i.e., $\{e | e \in \mathcal{A} \wedge e \notin \mathcal{B}\}$.

A child is considered to be the best choice in its parent node if the minimax value of the child is the same as that of the parent node. When no two children share the same value, $\mathcal{C}^+_n(\boldsymbol{w})$ contains only one child. Otherwise, the number of nodes in $\mathcal{C}^+_n(\boldsymbol{w})$ can be greater than one.

Definition 7. Let r be the root of a tree $\mathcal{T} \in \mathbb{T}$. The *principal variation* (abbreviated PV for short) $\mathcal{T}^*(\boldsymbol{w})$ of tree $\mathcal{T}$ is the sub-tree of $\mathcal{T}$ obtained as the closure of the best children from the root:

$$\begin{aligned} \mathcal{T}^0(\boldsymbol{w}) &= \{r\}, \\ \mathcal{T}^i(\boldsymbol{w}) &= \{c \in \mathcal{C}^+_{n,\mathcal{T}}(\boldsymbol{w}) \mid n \in \mathcal{T}^{i-1}(\boldsymbol{w})\} \text{ for } i > 0, \\ \mathcal{T}^*(\boldsymbol{w}) &= \bigcup_{i=0}^{\infty} \mathcal{T}^i(\boldsymbol{w}). \end{aligned}$$

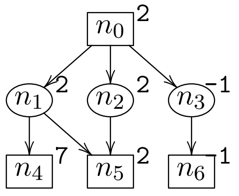

Figure 15: Example of a minimax tree (graph) with a transposition at n_5

Note that $\mathcal{C}^+_{n,\mathcal{T}^*}(\boldsymbol{w}) = \mathcal{C}^+_{n,\mathcal{T}}(\boldsymbol{w})$ and $\mathcal{C}^-_{n,\mathcal{T}^*}(\boldsymbol{w}) = \emptyset$ for any $n \in \mathcal{T}^*(\boldsymbol{w})$. Also, we denote leaves in $\mathcal{T}^*(\boldsymbol{w})$ by $\mathcal{L}^*(\boldsymbol{w})$, that is, $\mathcal{T}^*(\boldsymbol{w}) \cap \mathcal{L}_{r,\mathcal{T}}$.

Example 8. Figure 15 shows a small minimax tree $\mathcal{T}$ that has two best children at root n_0; the maximizing and minimizing nodes are denoted by boxes and circles, respectively. Here, $\mathcal{C}^+_{n_0} = \{n_1, n_2\}$ and $\mathcal{C}^+_{n_1} = \{n_5\}$. The principal variation $\mathcal{T}^*$ of this tree is $\{n_0, n_1, n_2, n_5\}$.

Lemma 9. *For any internal node n in any tree $\mathcal{T} \in \mathbb{T}$ and for any $\boldsymbol{w}^0 \in \mathbb{R}^N$, there exists a positive number δ_n such that for any $\boldsymbol{w}^1$ satisfying $|\boldsymbol{w}^1 - \boldsymbol{w}^0| < \delta_n$, the set of the best children at node n for $\boldsymbol{w}^1$ is a subset of the one for $\boldsymbol{w}^0$:*

$$\mathcal{C}^+_{n,\mathcal{T}}(\boldsymbol{w}^1) \subseteq \mathcal{C}^+_{n,\mathcal{T}}(\boldsymbol{w}^0), \text{ for any } \boldsymbol{w}^1 \text{ s.t. } |\boldsymbol{w}^1 - \boldsymbol{w}^0| < \delta_n.$$

Proof. When all child values are the same, i.e., $\mathcal{C}^-_n(\boldsymbol{w}^0)$ is empty, the assertion is trivial. Otherwise, let ε_0 be the minimum absolute difference between the best value and any of the other values, i.e., $\varepsilon_0 = \min_{c \in \mathcal{C}^-_n(\boldsymbol{w}^0)} |v_n(\boldsymbol{w}^0) - v_c(\boldsymbol{w}^0)| > 0$. The continuity of the minimax values ensures the existence of δ_n such that for any $\boldsymbol{w}^1$ satisfying $|\boldsymbol{w}^1 - \boldsymbol{w}^0| < \delta_n$, we have $\max_{c \in \mathcal{C}_n} |v_c(\boldsymbol{w}^1) - v_c(\boldsymbol{w}^0)| < \varepsilon_0/2$ and also $|v_n(\boldsymbol{w}^1) - v_n(\boldsymbol{w}^0)| < \varepsilon_0/2$. From the definition of δ_n and triangle inequalities, any $c \in \mathcal{C}^-_{n,\mathcal{T}}(\boldsymbol{w}^0)$ satisfies

$$\begin{aligned}
\varepsilon_0 &\le |v_c(\boldsymbol{w}^0) - v_n(\boldsymbol{w}^0)| \\
&\le |v_c(\boldsymbol{w}^0) - v_c(\boldsymbol{w}^1)| + |v_c(\boldsymbol{w}^1) - v_n(\boldsymbol{w}^0)| \\
&\le |v_c(\boldsymbol{w}^0) - v_c(\boldsymbol{w}^1)| + |v_c(\boldsymbol{w}^1) - v_n(\boldsymbol{w}^1)| + |v_n(\boldsymbol{w}^1) - v_n(\boldsymbol{w}^0)| \\
&< |v_c(\boldsymbol{w}^1) - v_n(\boldsymbol{w}^1)| + \frac{\varepsilon_0}{2} + \frac{\varepsilon_0}{2} \\
&= |v_c(\boldsymbol{w}^1) - v_n(\boldsymbol{w}^1)| + \varepsilon_0.
\end{aligned}$$

Thus, $|v_c(\boldsymbol{w}^1) - v_n(\boldsymbol{w}^1)| > \varepsilon_0 - \varepsilon_0 = 0$, namely, $v_c(\boldsymbol{w}^1) \neq v_n(\boldsymbol{w}^1)$. This implies by definition (irrespective of whether n is a max or min node) that $c \notin \mathcal{C}^+_{n,\mathcal{T}}(\boldsymbol{w}^1)$. □

Definition 10. The *tree stability* $\delta_\mathcal{T}$ of a tree $\mathcal{T}$ is the minimum value of δ_n among all the nodes $n \in \mathcal{T}$, where δ_n is a positive number satisfying Lemma 9. Note that the minimum value $\delta_\mathcal{T} > 0$ exists because $\mathcal{T}$ is finite.

Example 11. In reference to Figure 15, suppose that each leaf value changes by at most 0.1. Then, it will be proven that $|v_n(\boldsymbol{w}^1) - v_n(\boldsymbol{w}^0)| \le 0.1$ for each internal node n of heights 1, 2, and 3 in order: it is obvious for n_4, n_5, and n_6, and it can be proven for n_1, n_2 and n_3, and finally for n_0. We can see that neither n_4 nor n_6 can become a new best node as a result of this change.

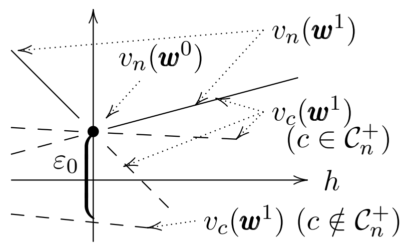

Figure 16: Sketch of $v_n(\boldsymbol{w}^1)$ at maximizing-node n, where $\boldsymbol{w}^1$ changes along the i-th component $w_i^0 + h$ from $\boldsymbol{w}^0$. Here, $v_n(\boldsymbol{w}^1)$ at n equals $v_c(\boldsymbol{w}^1)$ at one of the old best children $c \in \mathcal{C}_n^+(\boldsymbol{w}^0)$.

A.3 Partial Differentiability

We show that the partial differentiability, as well as the partial derivative, of the minimax value at a node in tree $\mathcal{T}$ depends only on its principal variations. We denote the right and left partial derivatives of a function $\mathbb{R}^N \mapsto \mathbb{R}$ at point $\boldsymbol{x}^0$ as

$$\frac{\partial f}{\partial x_i^+}(\boldsymbol{x}^0) = \lim_{h \to +0} \frac{f(x_1^0, \ldots, x_i^0 + h, \ldots, x_N^0) - f(x_1^0, \ldots, x_N^0)}{h}, \quad (12)$$

$$\frac{\partial f}{\partial x_i^-}(\boldsymbol{x}^0) = \lim_{h \to -0} \frac{f(x_1^0, \ldots, x_i^0 + h, \ldots, x_N^0) - f(x_1^0, \ldots, x_N^0)}{h}. \quad (13)$$

Let us pay attention to the single parameter x_i that changes by h under these limit operations. Hereafter, the other parameters held constant will often be omitted as $\frac{\partial f}{\partial x^+}(x^0)$, where x is the one-dimensional parameter of interest. We use the symbol ∂ because of its analogy to the partial derivative in order not to forget that the other parameters have been omitted.

Theorem 12. *For any node n in the principal variation $\mathcal{T}^*(\boldsymbol{w})$ of tree $\mathcal{T} \in \mathbb{T}$ and for any $\boldsymbol{w} \in \mathbb{R}^N$, there exists such a leaf $l_a \in \mathcal{L}^*(\boldsymbol{w})$ at which the partial derivative of the evaluation function equals the right partial derivative of $v_n(\boldsymbol{w})$: $\frac{\partial v_n}{\partial w_i^+}(\boldsymbol{w}) = \frac{\partial}{\partial w_i}\,\mathrm{e}(l_a, \boldsymbol{w})$. Similarly, there exists such a leaf $l_b \in \mathcal{L}^*(\boldsymbol{w})$ at which the partial derivative of the evaluation function equals the left partial derivative of $v_n(\boldsymbol{w})$, $\frac{\partial v_n}{\partial w_i^-}(\boldsymbol{w}) = \frac{\partial}{\partial w_i}\,\mathrm{e}(l_b, \boldsymbol{w})$.*

The proof of the theorem, given at the end of this subsection, is based on the stability of the best moves. We assume that $\boldsymbol{w}^1 = (w_1^0, \ldots, w_i^0 + h, \ldots, w_N^0)$ and $|\boldsymbol{w}^1 - \boldsymbol{w}^0| = |h|$ are sufficiently small in Appendix A.3. Consequently, we have $|h| < \delta_{\mathcal{T}}$, and for any node n in tree $\mathcal{T}$ and for any $\boldsymbol{w}^0 \in \mathbb{R}^N$,

$$v_n(\boldsymbol{w}^1) = \begin{cases} \mathrm{e}(n, \boldsymbol{w}^1) & (n\text{:leaf}) \\ \max_{c \in \mathcal{C}_{n,\mathcal{T}}^+(\boldsymbol{w}^0)} v_c(\boldsymbol{w}^1) & (n\text{:maximizing node}) \\ \min_{c \in \mathcal{C}_{n,\mathcal{T}}^+(\boldsymbol{w}^0)} v_c(\boldsymbol{w}^1) & (n\text{:minimizing node}). \end{cases} \quad (14)$$

Example 13. Figure 16 sketches an example of $v_n(\boldsymbol{w}^1)$ changing with h, where n is a maximizing node. There are three best children with value $v_n(\boldsymbol{w}^0)$ when $h = 0$. Each value continuously (not always linearly) changes with h. While the best child depends on the sign of h, it is always one of $c \in \mathcal{C}_n^+(\boldsymbol{w}^0)$ when h is less than $\delta_{\mathcal{T}}$. This is because the minimax values of the other children $c \in \mathcal{C}_n^-(\boldsymbol{w}^0)$ are sufficiently less (by at least ε_0) than $v_n(\boldsymbol{w}^0)$ at $h = 0$.

The next goal is to show that the right and left partial derivatives of $v_n(\boldsymbol{w})$ are given by the right and left partial derivatives at one of the best children $\mathcal{C}_n^+(\boldsymbol{w})$, respectively. The following propositions describe the ordinary properties of the right and left limits and the basic functions max and min. Similar arguments can be found in a comprehensive textbook of calculus. We will give a detailed proof here, however, because it is rather difficult to find the precisely same assertion in a textbook.

Proposition 14. *Let k be a natural number and any of $f_1(x), ..., f_k(x)$ be a continuous function $\mathbb{R} \mapsto \mathbb{R}$. Suppose that these functions have the same value at point x^0, i.e., $\max_i f_i(x^0) = \min_i f_i(x^0)$, and all of them have a right partial derivative $\frac{\partial f_i}{\partial x^+}(x^0)$. Then, the right partial derivative of the minimum or maximum of $f_i(x)$ at point x^0 exists and is equal to the minimum or maximum of the right partial derivatives of $f_i(x^0)$, respectively.*

$$\frac{\partial \max_i f_i}{\partial x^+}(x^0) = \max_i \frac{\partial f_i}{\partial x^+}(x^0), \quad \frac{\partial \min_i f_i}{\partial x^+}(x^0) = \min_i \frac{\partial f_i}{\partial x^+}(x^0).$$

Proof. Let $o(h)$ be Landau's symbol, and let us use it to denote residual terms converging to 0 faster than h, i.e., $\lim_{h\to+0} \frac{o(h)}{h} = 0$. Recall that $f_i(x^0) = f_1(x^0)$ for any $i = 1, \ldots, k$. For positive h, we have

$$\begin{aligned}
\max_i f_i(x^0 + h) - \max_i f_i(x^0) &= \max_i \left(f_i(x^0) + h\frac{\partial f_i}{\partial x^+}(x^0) + o(h) \right) - \max_i f_i(x^0) \\
&= \max_i \left(f_1(x^0) + h\frac{\partial f_i}{\partial x^+}(x^0) + o(h) \right) - f_1(x^0) \\
&= h \left(\max_i \frac{\partial f_i}{\partial x^+}(x^0) \right) + o(h)
\end{aligned}$$

From Eq. (12), the function $\max_i f_i(x)$ at point x^0 has a right partial derivative $\max_i \frac{\partial f_i}{\partial x^+}(x^0)$. The same argument applies to the right partial derivative of $\min_i f_i(x)$. □

Proposition 15. *Suppose that functions have the same value at point x^0 and all of these functions have a left derivative $\frac{\partial f_i}{\partial x^-}(x^0)$. Then, the left partial derivative of the minimum or maximum of $f_i(x)$ at point x^0 is equal to the maximum or minimum of the left partial derivatives of $f_i(x^0)$:*

$$\frac{\partial \max_i f_i}{\partial x^-}(x^0) = \min_i \frac{\partial f_i}{\partial x^-}(x^0), \quad \frac{\partial \min_i f_i}{\partial x^-}(x^0) = \max_i \frac{\partial f_i}{\partial x^-}(x^0).$$

Proof. Using similar algebra as in the proof of Proposition 14, we find for negative h,

$$\max_i f_i(x^0+h) - \max_i f_i(x^0) = h\left(\min_i \frac{\partial f_i}{\partial x^-}(x^0)\right) + o(h).$$

From Eq. (13), the function $\max_i f_i(x)$ at point x^0 has the left partial derivative $\min_i \frac{\partial f_i}{\partial x^-}(x^0)$. Note that min and max are switched in the algebra above because of the negativity of h. The same argument applies to the left partial derivative of $\min_i f_i(x)$. □

Lemma 16. *Let $g_i^+(n,\boldsymbol{w}) = \frac{\partial v_n}{\partial w_i^+}(\boldsymbol{w})$ (resp. $g_i^-(n,\boldsymbol{w}) = \frac{\partial v_n}{\partial w_i^-}(\boldsymbol{w})$) be the right (resp. left) partial derivative of the minimax value $v_n(\boldsymbol{w})$. For any $\boldsymbol{w} \in \mathbb{R}^N$ and for any internal node n in principal variation $\mathcal{T}^*(\boldsymbol{w})$ of tree $\mathcal{T} \in \mathbb{T}$, there exist right and left partial derivatives $g_i^+(n,\boldsymbol{w})$ and $g_i^-(n,\boldsymbol{w})$ of $v_n(\boldsymbol{w})$ with respect to any $i = 1,\ldots,N$. The right and left partial derivatives are:*

$$g_i^+(n,\boldsymbol{w}) = \begin{cases} \max_{c\in\mathcal{C}^+_{n,\mathcal{T}^*(\boldsymbol{w})}} g_i^+(c,\boldsymbol{w}) & (n\text{: maximizing node}) \\ \min_{c\in\mathcal{C}^+_{n,\mathcal{T}^*(\boldsymbol{w})}} g_i^+(c,\boldsymbol{w}) & (n\text{: minimizing node}) \end{cases}$$

$$g_i^-(n,\boldsymbol{w}) = \begin{cases} \min_{c\in\mathcal{C}^+_{n,\mathcal{T}^*(\boldsymbol{w})}} g_i^-(c,\boldsymbol{w}) & (n\text{: maximizing node}) \\ \max_{c\in\mathcal{C}^+_{n,\mathcal{T}^*(\boldsymbol{w})}} g_i^-(c,\boldsymbol{w}) & (n\text{: minimizing node}). \end{cases}$$

Proof. We prove these equalities on the basis of mathematical induction from the leaf nodes $\mathcal{L}_{r,\mathcal{T}}$ to the root r. For each leaf n in $\mathcal{L}^*(\boldsymbol{w})$, by the definition of the evaluation function, the minimax value $v_n(\boldsymbol{w})$ is clearly continuous and partially differentiable with respect to any component in $\boldsymbol{w} \in \mathbb{R}^N$. For any internal node n, we assume, as an induction hypothesis, that the right partial derivative $g_i^+(c,\boldsymbol{w})$ and left partial derivative $g_i^-(c,\boldsymbol{w})$ exist for any child $c \in \mathcal{C}_{n,\mathcal{T}}$. Recall that $\mathcal{C}_n^+(\boldsymbol{w}^1) \subseteq \mathcal{C}_n^+(\boldsymbol{w}^0)$ for any $|h| < \delta_{\mathcal{T}}$ and Eq. (14). From the induction hypothesis with Proposition 14, we have

$$\frac{\partial \max_{c\in\mathcal{C}_n^+(\boldsymbol{w})} v_c}{\partial w_i^+}(\boldsymbol{w}) = \max_{c\in\mathcal{C}_n^+(\boldsymbol{w})} \frac{\partial v_c}{\partial w_i^+}(\boldsymbol{w}), \quad \frac{\partial \min_{c\in\mathcal{C}_n^+(\boldsymbol{w})} v_c}{\partial w_i^+}(\boldsymbol{w}) = \min_{c\in\mathcal{C}_n^+(\boldsymbol{w})} \frac{\partial v_c}{\partial w_i^+}(\boldsymbol{w}).$$

Similarly, from Proposition 15, we have

$$\frac{\partial \max_{c\in\mathcal{C}_n^+(\boldsymbol{w})} v_c}{\partial w_i^-}(\boldsymbol{w}) = \min_{c\in\mathcal{C}_n^+(\boldsymbol{w})} \frac{\partial v_c}{\partial w_i^-}(\boldsymbol{w}), \quad \frac{\partial \min_{c\in\mathcal{C}_n^+(\boldsymbol{w})} v_c}{\partial w_i^-}(\boldsymbol{w}) = \max_{c\in\mathcal{C}_n^+(\boldsymbol{w})} \frac{\partial v_c}{\partial w_i^-}(\boldsymbol{w}).$$

□

Now, we prove Theorem 12. For any leaf $n \in \mathcal{L}^*(\boldsymbol{w})$, it is obvious that $g_i^+(n,\boldsymbol{w}) = g_i^-(n,\boldsymbol{w}) = \frac{\partial}{\partial w_i}\,\mathrm{e}(n,\boldsymbol{w})$. For any internal node $n \in \mathcal{T}^*(\boldsymbol{w})$, Lemma 16 ensures that the left and right partial derivatives $g_i^+(n,\boldsymbol{w})$ and $g_i^-(n,\boldsymbol{w})$ are given by one of the best children. Thus, for root r, there always exist leaves l_a and $l_b \in \mathcal{L}^*(\boldsymbol{w})$ such that

$$g_i^+(r,\boldsymbol{w}) = \frac{\partial}{\partial w_i}\,\mathrm{e}(l_a,\boldsymbol{w}), \qquad g_i^-(r,\boldsymbol{w}) = \frac{\partial}{\partial w_i}\,\mathrm{e}(l_b,\boldsymbol{w}). \tag{15}$$

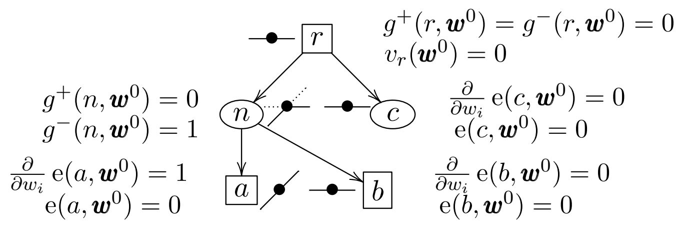
$g^+(r, \boldsymbol{w}^0) = g^-(r, \boldsymbol{w}^0) = 0$
$v_r(\boldsymbol{w}^0) = 0$
r
$g^+(n, \boldsymbol{w}^0) = 0$
$g^-(n, \boldsymbol{w}^0) = 1$
n
c
$\frac{\partial}{\partial w_i} \mathrm{e}(c, \boldsymbol{w}^0) = 0$
$\mathrm{e}(c, \boldsymbol{w}^0) = 0$
$\frac{\partial}{\partial w_i} \mathrm{e}(a, \boldsymbol{w}^0) = 1$
$\mathrm{e}(a, \boldsymbol{w}^0) = 0$
a
b
$\frac{\partial}{\partial w_i} \mathrm{e}(b, \boldsymbol{w}^0) = 0$
$\mathrm{e}(b, \boldsymbol{w}^0) = 0$

Figure 17: Although the partial derivative of $v_r(\boldsymbol{w})$ exists at $\boldsymbol{w}^0$, it is not equal to the partial derivative at a PV leaf $\frac{\partial}{\partial w_i} \mathrm{e}(a, \boldsymbol{w}^0)$.

Remark 17. By definition, if $g_i^+(n, \boldsymbol{w}^0) = g_i^-(n, \boldsymbol{w}^0)$, the partial derivative of $v_n(\boldsymbol{w})$ with respect to w_i exists at the point $\boldsymbol{w}^0$ and there is a leaf $l \in \mathcal{L}^*(\boldsymbol{w}^0)$ satisfying

$$\frac{\partial}{\partial w_i} v_n(\boldsymbol{w}^0) = \frac{\partial}{\partial w_i} \mathrm{e}(l, \boldsymbol{w}^0). \tag{16}$$

Remark 18. For any $i = 1, \dots, N$, the partial derivative of minimax value $v_n(\boldsymbol{w})$ with respect to w_i exists at $\boldsymbol{w}^0$ and equals $\frac{\partial}{\partial w_i} \mathrm{e}(l, \boldsymbol{w}^0)$, if l is the unique element of $\mathcal{L}^*(\boldsymbol{w}^0)$.

Remark 19. There exists a tree $\mathcal{T}_r$ for which the minimax value $v_n(\boldsymbol{w})$ has a partial derivative with respect to w_i at $\boldsymbol{w}^0$, even when the leaves l in PV are not unique ($|\mathcal{L}^*(\boldsymbol{w}^0)| > 1$) and give different partial derivatives $\frac{\partial}{\partial w_i} \mathrm{e}(l, \boldsymbol{w}^0)$. An example is sketched in Figure 17, where the partial derivative is 1 for a and 0 for b and c.

A.4 Game-Tree Search and Pruning Techniques

Consider a game tree search S be a function that takes the root position r and the evaluation-function parameters $\boldsymbol{w}$ as inputs, and yields a minimax tree $\mathcal{T}_r^S(\boldsymbol{w})$ with minimax values $v_{n,\mathcal{T}_r(\boldsymbol{w})}(\boldsymbol{w})$ for all $n \in \mathcal{T}_r^S(\boldsymbol{w})$. We call a game-tree search S *static*, provided that it yields a constant tree with respect to $\boldsymbol{w}$, i.e., $V(\mathcal{T}_r^S(\boldsymbol{w})) = V(\mathcal{T}_r^S(\boldsymbol{w}^0))$, for any root r. Then, theorems 4 and 12 apply to the minimax value $v_{r,\mathcal{T}_r^S}(\boldsymbol{w})$ yielded by such a static game-tree search. For example, a fixed-depth minimax search or a minimax search considering limited types of moves (e.g., capture and promotion) is a static game-tree search. A minimax search with "stand pat" used in the quiescence search (Beal, 1990) is static, too. Note that "stand pat" at node n is equivalent to a virtual move adding an evaluation function $\mathrm{e}(n, \boldsymbol{w})$ as a candidate of the node value $v_n(\boldsymbol{w})$ in Eq. (11), even when n is not a leaf node.

When pruning techniques are incorporated, part of the tree is pruned and not explored. Consider a static search S, that with a pruning S', and tree $\mathcal{T}_r^{S'}(\boldsymbol{w}) \subseteq \mathcal{T}_r^S(\boldsymbol{w})$ yielded by S'. We call a pruning *conservative*, provided that it yields the same minimax value at any root r for any $\boldsymbol{w} \in \mathbb{R}^N$: $v_{r,\mathcal{T}_r^S}(\boldsymbol{w}) = v_{r,\mathcal{T}_r^{S'}(\boldsymbol{w})}(\boldsymbol{w})$. Theorem 4 applies to the minimax value at the root r, $v_{r,\mathcal{T}_r^{S'}(\boldsymbol{w})}(\boldsymbol{w})$, yielded by such a static game-tree search with conservative pruning. Standard $\alpha\beta$ pruning (Knuth & Moore, 1975) is a conservative pruning. However, many pruning techniques, e.g., static exchange evaluation (Reul, 2010), (extended) futility pruning (Heinz, 1998), null move pruning (Adelson-Velskiy et al., 1975), and late move reductions (Romstad, 2010), can prune a sub-tree without having to prove that the sub-tree is irrelevant to the minimax value at the root. Thus, these pruning techniques are generally not conservative.

A.5 Summary

The minimax value of the root of the tree explored by a game-tree search with well-configured pruning techniques is continuous. This result suggests the continuity of the objective function of MMTO in Eq. (4), as was empirically observed in Section 4.5. As for partial differentiability, Theorem 12 suggest that it is feasible to consider the leaves of the principal variations in a search tree. When there is only one principal variation, as stated in Remark 18, the use of the partial derivative at the unique leaf introduced in Section 3.3 is correct. Otherwise, i.e., when there are multiple principal variations, the partial derivative may not exist or be different from the partial derivative at one of the leaves, as stated in Remark 19. Although the frequency of such cases depends on the target game and on the evaluation features, it is almost negligible in the experiments discussed in our previous work (Kaneko & Hoki, 2012).

References

Adelson-Velskiy, G. M., Arlazarov, V. L., & Donskoy, M. V. (1975). Some methods of controlling the tree search in chess programs. *Artificial Intelligence*, *6*(4), 361 – 371.

Akl, S. G., & Newborn, M. M. (1977). The principal continuation and the killer heuristic. In *Proceedings of the 1977 Annual Conference*, ACM '77, pp. 466–473, New York, NY, USA. ACM.

Anantharaman, T. (1997). Evaluation tuning for computer chess: Linear discriminant methods. *ICCA Journal*, *20*(4), 224–242.

Baxter, J., Tridgell, A., & Weaver, L. (2000). Learning to play chess using temporal-differences. *Machine Learning*, *40*(3), 242–263.

Beal, D. F. (1990). A generalised quiescence search algorithm. *Artificial Intelligence*, *43*, 85–98.

Beal, D. F., & Smith, M. C. (2001). Temporal difference learning applied to game playing and the results of application to shogi. *Theoretical Computer Science*, *252*(1-2), 105–119.

Bertsekas, D. P., & Bertsekas, D. P. (2008). *Nonlinear Programming* (2nd edition). Athena Scientific.

Björnsson, Y., & Marsland, T. A. (2002). Learning control of search extensions. In Caulfield, H. J., Chen, S.-H., Cheng, H.-D., Duro, R. J., Honavar, V., Kerre, E. E., Lu, M., Romay, M. G., Shih, T. K., Ventura, D., Wang, P. P., & Yang, Y. (Eds.), *JCIS*, pp. 446–449. JCIS / Association for Intelligent Machinery, Inc.

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, *4*(1), 1–43.

Buro, M. (2002). Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, *134*(1–2), 85–99.

Buro, M., Long, J. R., Furtak, T., & Sturtevant, N. R. (2009). Improving state evaluation, inference, and search in trick-based card games. In *IJCAI*, pp. 1407–1413.

Buro, M. (1995). Statistical feature combination for the evaluation of game positions. *Journal of Artificial Intelligence Research*, *3*, 373–382.

Campbell, M., Hoane, Jr., A. J., & Hsu, F.-h. (2002). Deep Blue. *Artificial Intelligence*, *134*(1–2), 57–83.

Chellapilla, K., & Fogel, D. (1999). Evolving neural networks to play checkers without relying on expert knowledge. *Neural Networks, IEEE Transactions on*, *10*(6), 1382 –1391.

Coulom, R. (2007). Computing "Elo Ratings" of move patterns in the game of go. *ICGA Journal*, *30*(4), 198–208.

Coulom, R. (2012). Clop: Confident local optimization for noisy black-box parameter tuning. In Herik, H., & Plaat, A. (Eds.), *Advances in Computer Games 13*, No. 7168 in LNCS, pp. 146–157. Springer-Verlag.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*, 2121–2159.

Fawcett, T. E. (1993). *Feature Discovery for Problem Solving Systems*. Ph.D. thesis, Department of Computer Science, University of Massachusetts, Amherst.

Fürnkranz, J. (2001). Machine learning in games: a survey. In *Machines that learn to play games*, pp. 11–59. Nova Science Publishers, Commack, NY, USA.

Gelly, S., Kocsis, L., Schoenauer M., Sebag, M., Silver, D., Szepesvári, C., & Tẽytaud, O. (2012). The grand challenge of computer go: Monte carlo tree search and extensions. *Commun. ACM*, *55*(3), 106–113.

Gelly, S., & Silver, D. (2011). Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, *175*(11), 1856–1875.

Gomboc, D., Buro, M., & Marsland, T. A. (2005). Tuning evaluation functions by maximizing concordance. *Theoretical Computer Science*, *349*(2), 202–229.

Heinz, E. A. (1998). Extended futility pruning. *ICCA Journal*, *21*(2), 75–83.

Heinz, E. A. (1999). Adaptive null-move pruning. *ICCA Journal*, *22*(3), 123–132.

Hoki, K. Bonanza – the computer shogi program.. `http://www.geocities.jp/bonanza_shogi/` Last access: 2013. In Japanese.

Hoki, K. (2006). Optimal control of minimax search results to learn positional evaluation. In *The 11th Game Programming Workshop (GPW2006)*, pp. 78–83, Kanagawa, Japan. In Japanese.

Hoki, K., & Kaneko, T. (2012). The global landscape of objective functions for the optimization of shogi piece values with game-tree search. In van den Herik, H. J., & Plaat, A. (Eds.), *Advances in Computer Games 13*, No. 7168 in LNCS, pp. 184–195. Springer-Verlag.

Hoki, K., & Muramatsu, M. (2012). Efficiency of three forward-pruning techniques in shogi: Futility pruning, null-move pruning, and late move reduction (LMR). *Entertainment Computing*, *3*(3), 51–57.

Hsu, F.-h., Anantharaman, T. S., Campbell, M. S., & Nowatzyk, A. (1990). Deep Thought. In Marsland, T. A., & Schaeffer, J. (Eds.), *Computers, Chess, and Cognition*, pp. 55–78. Springer-Verlag.

Iida, H., Sakuta, M., & Rollason, J. (2002). Computer shogi. *Artificial Intelligence*, *134*(1–2), 121–144.

Kaneko, T. (2009). Recent improvements on computer shogi and GPS-Shogi. *IPSJ Magazine*, *50*(9), 878–886. In Japanese.

Kaneko, T., & Hoki, K. (2012). Analysis of evaluation-function learning by comparison of sibling nodes. In van den Herik, H. J., & Plaat, A. (Eds.), *Advances in Computer Games 13*, No. 7168 in LNCS, pp. 158–169. Springer-Verlag.

Knuth, D. E., & Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, *6*(4), 293–326.

Kocsis, L., & Szepesvari, C. (2006). Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, Vol. 4212, pp. 282–293. Springer.

Krogius, N., Livsic, A., Parma, B., & Taimanov, M. (1980). *Encyclopedia of Chess Middlegames: Combinations*. Chess Informant.

Levinson, R., & Weber, R. (2001). Chess neighborhoods, function combination, and reinforcement learning. In Marsland, T. A., & Frank, I. (Eds.), *Computer and Games*, No. 2063 in LNCS, pp. 133–150. Springer-Verlag.

Marsland, T. A. (1985). Evaluation function factors. *ICCA Journal*, *8*(2), 47–57.

Marsland, T. A., & Campbell, M. (1982). Parallel search of strongly ordered game trees. *ACM Computing Surveys*, *14*(4), 533–551.

Nitsche, T. (1982). A learning chess program. In *Advances in Computer Chess 3*, pp. 113–120. Pergamon Press.

Nocedal, J., & Wright, S. (2006). *Numerical Optimization*. Springer-Verlag.

Nowatzyk, A. (2000). `http://tim-mann.org/DT_eval_tune.txt`.

Pearl, J. (1980). Scout: A simple game-searching algorithm with proven optimal properties. In *In Proceedings of the First Annual National Conference on Artificial Intelligence*, pp. 143–145.

Reinefeld, A. (1983). An improvement to the scout tree search algorithm. *ICCA Journal*, *6*(4), 4–14.

Reinfeld, F. (1969). *1001 Winning Chess Sacrifices and Combinations*. Wilshire Book Company.

Reinfeld, F. (2001). *Win at Chess (Dover Books on Chess)*. Dover Publications.

Reul, F. (2010). Static exchange evaluation with $\alpha\beta$-approach. *ICGA Journal*, *33*(1), 3–17.

Romstad, T. An Introduction to Late Move Reductions. `http://www.glaurungchess.com/lmr.html`, Last access: 2010.

Russell, S. J., & Norvig, P. (2002). *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall.

Schaeffer, J. (1986). Experiments in search and knowledge. Ph.D. Thesis, Department of Computing Science, University of Waterloo, Canada.

Schaeffer, J. (1989). The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-11*(1), 1203–1212.

Schaeffer, J., Hlynka, M., & Jussila, V. (2001). Temporal difference learning applied to a high-performance game-playing program. In *IJCAI'01: Proceedings of the 17th international joint conference on Artificial intelligence*, pp. 529–534, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Silver, D., & Tesauro, G. (2009). Monte-carlo simulation balancing. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 945–952. ACM.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press.

Tanaka, T., & Kaneko, T. GPS Shogi.. `http://gps.tanaka.ecc.u-tokyo.ac.jp/gpsshogi/` Last access: 2013. In Japanese.

Tesauro, G. (2001). Comparison training of chess evaluation functions. In *Machines that Learn to Play Games*, pp. 117–130. Nova Science Publishers.

Tesauro, G. (2002). Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, *134*(1–2), 181–199.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *J. Royal. Statist. Soc B*, *58*(1), 267–288.

Tsuruoka, Y., Yokoyama, D., & Chikayama, T. (2002). Game-tree search algorithm based on realization probability. *ICGA Journal*, *25*(3), 145–152.

Ugajin, T., & Kotani, Y. (2010). Learning evaluation function based on tree strap in shogi. In *The 15th Game Programming Workshop*, pp. 114–118. In Japanese.

van den Herik, H. J., Uiterwijk, J. W. H. M., & van Rijswijck, J. (2002). Games solved: now and in the future. *Artif. Intell.*, *134*(1-2), 277–311.

van der Meulen, M. (1989). Weight assessment in evaluation functions. In Beal, D. (Ed.), *Advances in. Computer Chess 5*, pp. 81–89.

Veness, J., Silver, D., Uther, W., & Blair, A. (2009). Bootstrapping from game tree search. In *Advances in Neural Information Processing Systems 22*, pp. 1937–1945.

Zobrist, A. L. (1990). A new hashing method with application for game playing. *ICCA Journal*, *13*(2), 69–73.