A SHOGI PROGRAM BASED ON MONTE-CARLO TREE SEARCH

Yoshikuni Sato[1], *Daisuke Takahashi*[2] *and Reijer Grimbergen*[3]

Tsukuba and Tokyo, Japan

ABSTRACT

Recently, Monte-Carlo Tree Search (MCTS) has been attracting a lot of attention in game programming research. This method has been very successful in computer Go, but results in other games have not been so impressive. In this paper, we present an implementation of MCTS in shogi which combines techniques used in computer Go with a number of shogi-specific enhancements. We tested this implementation on a standard test set of tactical positions. The number of correct answers indicates that the strength of the Monte-Carlo based shogi program is about 1-dan amateur. These results did not carry over to actual playing strength as the match results against a conventional shogi program with a strength of about 1-dan showed. Therefore, it seems unlikely that a pure MCTS-based shogi program will surpass the level of the best conventional shogi programs. However, we also observed that our MCTS program could solve certain opening and endgame positions that are considered hard to solve with current methods. Therefore, we believe that MCTS can be a useful tool to improve the overall performance of a shogi program.

1. INTRODUCTION

Traditionally, strong game programs (like for chess and shogi) use fast search algorithms combined with a game-specific evaluation function. However, recently Monte-Carlo Tree Search (MCTS) has been attracting attention as a new paradigm in game programming (Coulom, 2006). The distinctive feature of MCTS is that no evaluation function is needed. Therefore, this method is especially suited for games like Go where it has been difficult to design efficient evaluation functions.

The success in Go has led to the application of MCTS in a number of other games like Chinese Checkers (Sturtevant, 2008), Amazons (Lorentz, 2008), Hex (Cazenave and Saffidine, 2009), Lines of Action (Winands and Björnsson, 2009), and Kriegspiel (Ciancarini and Favini, 2009). Despite the growing amount of research into MCTS, there has been little work comparing the performance of MCTS in a chess-like game with conventional search using a strong evaluation function.

In this paper, we present an implementation of MCTS for the game of shogi (Japanese chess). Due to recent developments like automatic tuning of evaluation functions, the strength of the best shogi programs is close to that of professional players[4]. Despite this, shogi programs have some serious weaknesses that are difficult to address with the current methods of search and evaluation. In the opening and early middle game, long-term strategy plays a vital role in shogi. This has traditionally been a problem for search algorithms and most shogi programs just try to imitate human moves based on the game records of strong human players. A second problem is that some complicated endgame positions require directed, deep search to reach a conclusion about which side will mate the king of the opponent first. Current search methods are not well-suited for this particular type of search.

[1]Graduate School of Systems and Information Engineering, University of Tsukuba. Mail: ysato@hpcs.cs.tsukuba.ac.jp

[2]Graduate School of Systems and Information Engineering, University of Tsukuba.

[3]School of Computer Science, Tokyo University of Technology

[4]In Japan, the best shogi players receive a basic salary from the Japan Shogi Association and play in different tournaments from amateur players. About 150 players have this professional status. A shogi program close to professional strength can therefore be expected to beat all but the top 150 players.

MCTS could be a solution to these problems, and therefore has recently been investigated for shogi as well. However, thus far no promising results have been reported. The most important problem of using MCTS in shogi is how to satisfy the game-ending condition of checkmate which is the basis for the simulations.

In this paper, we present a shogi program based on UCT (Upper Confidence bounds applied to Trees) (Kocsis and Szepesvári, 2006), the common MCTS implementation for games. We will show that by adding some shogi-specific enhancements to the techniques used successfully in Go, the problem of satisfying the game-ending condition can be solved, leading to a reasonably strong MCTS shogi program. Furthermore, we will show that MCTS could be a tool to improve the weak points of current shogi programs.

This paper is built up as follows. In Section 2 we will describe the general UCT algorithm. In Section 3 we will describe the enhancements needed to make UCT work in shogi. In Section 4 we present the experimental results and we end with conclusions and comments on future work in Section 5.

2. MONTE-CARLO TREE SEARCH

Monte-Carlo Tree Search is a best-first search algorithm based on random simulations of games, called playouts. In playouts, moves are selected randomly from the each position until a position is reached where the game-ending conditions are being satisfied. In the MCTS tree, the selection of the move in a node is based on the win rate of that move among its playouts, where a higher win rate is considered better.

In the original MCTS algorithm, selecting moves in the playouts is done randomly, but the performance of the algorithm can be improved significantly by using game-specific heuristics to select promising moves more often. Because MCTS does not need an evaluation function, it is well-suited for games like Go where the design of an efficient evaluation function has proved difficult.

UCT (Upper Confidence bounds applied to Trees) is the MCTS algorithm that is most commonly used in game programs. It is an adaptation to minimax tree search of the UCB1 algorithm (Auer, Cesa-Bianchi, and Fischer, 2002) for multi-armed bandit problems. UCT is a decision algorithm to select the most effective node to expand next and thus guides the playouts. Effectiveness is evaluated by UCB1 based on the win rate of the node and the number of times the node was visited.

UCB1 selects node $I \in \{1, ..., K\}$ as follows:

$$I = \operatorname*{argmax}_{i \in \{1,\ldots,K\}} \{X_i + c\sqrt{\frac{2\log n}{n_i}}\} \qquad (1)$$

where K is number of moves, X_i is the value (win rate) of node i, n_i is the visiting count of node i, and n is the visiting count of the parent node of i. The parameter c is usually set to 1.0, but may need to be tuned based on the implementation. Larger c assigns more playouts to nodes with lower win rate. The move that will be played next in the actual game is then the move whose associated child node has the highest visit count.

3. A SHOGI PROGRAM BASED ON MONTE-CARLO TREE SEARCH

A shogi program using simple MCTS based on the algorithm used in the Go program CRAZY STONE has been implemented before (Hashimoto, Hashimoto, and Nagashima, 2006). In this implementation, to avoid the problem of doing playouts until the end of the game, the length of simulations was set at about 10 moves. The decision about the simulation ending in a win or a loss was then made using an evaluation function. The number of correct answers to a standard set of tactical shogi positions was only about 3%, leading to the conclusion that this type of implementation of MCTS was unsuitable for shogi.

In this paper, we present the results of a shogi program based on UCT, which has a number of additional techniques that were recently shown to be effective in Go, particularly progressive widening, killer moves, history heuristics and simple parallelism. Additionally, the selection of moves during playouts was improved by using Elo ratings for moves. As a result of these modifications, the quality of the simulations was improved significantly and the problem of being unable to do playouts until the end of the game was solved, so no evaluation function combined with a strong limit on the length of the playouts was needed. We will now explain each of the enhancements in detail.

3.1 Improving the Quality of the Playouts

Originally, the move selection in playouts is performed completely at random. However, in most MCTS algorithms there are heuristics for selecting promising moves more often than other moves. As a result, the reliability of the outcome of the playouts is improved. Such heuristics are also expected to help the playouts reach the end of the game, which has been a problem in earlier implementations of MCTS in shogi.

One of the heuristics we used to select moves in playouts is the method of assigning an Elo rating to moves as proposed by Coulom (2007). This method estimates the probability that a move will be selected based on a combination of features.

3.1.1 Bradley-Terry Model and Elo Ratings

The method of assigning an Elo rating to moves is based on the Bradley-Terry model, which predicts the outcome of competitions between individuals and groups based on the relative strength of the individual participants.

The prediction for individuals is as follows. When the strength of individual i is expressed as a positive value γ_i, the probability that i wins a competition between n individuals can be calculated using the following formula:

$$P(i \text{ wins}) = \frac{\gamma_i}{\sum_{j=1}^{n} \gamma_j}$$

The expected outcome of competitions between teams (a group of individuals) can be also calculated. The strength of a team is estimated as the product of the strength of its members. For instance, the probability that a team with two individual players 4 and 5 wins against a team with three players 1, 2 and 3 and a team with a single player 6 is calculated as follows:

$$P(\text{4-5 wins}) = \frac{\gamma_4\gamma_5}{\gamma_1\gamma_2\gamma_3 + \gamma_4\gamma_5 + \gamma_6}$$

In general, the Elo rating of individual i is defined by $400 \log_{10} \gamma_i$. In this paper, we just use γ_i as the rating of move feature i.

3.1.2 Application to Shogi

In shogi, each move can have a number of different features. For example, capture, attacking the king, giving check, escape from an attacking piece, etc. A move can also have multiple features, e.g. capturing a piece while giving check. Therefore, in the Bradley-Terry model, features and moves can be considered as individuals and teams, respectively. The probability that a move is selected in a certain position corresponds to the probability that a certain team of move features wins.

The rating γ_i of each move feature is estimated by the method given by Coulom (2007). This method uses the game records of strong players as learning data. Each position in the game record corresponds to a single competition. If a move is selected by a strong player, this means that the team of features of the selected move wins the competition while the other teams of move features lost. The ratings of the features are then adjusted in such a way that the rating of the selected move is increased and the rating of the features of the moves that were not selected is decreased.

In shogi, we have used the following features. For clarity, we have only given the major categories here. Most of the features have sub-features. For example, escaping with a major piece like a rook will have a different rating than escaping with a small piece like a pawn. The total number of features in our implementation is about 200.

- Material balance

  This is calculated by SEE (Static Exchange Evaluation) using the piece values of Table 1. Calculating SEE is more expensive than the other move features, but it is particularly important in shogi.

Table 1: Values of Pieces (+ indicates a promoted piece)

歩 (Pawn)	香 (Lance)	桂 (Knight)	銀 (Silver)	金 (Gold)	角 (Bishop)	飛 (Rook)
100	280	300	420	530	620	700
と (+Pawn)	成香 (+Lance)	成桂 (+Knight)	成銀 (+Silver)	—	馬 (+Bishop)	龍 (+Rook)
270	320	250	430	—	710	850

- Capture, recapture

  Even bad captures should be considered in shogi, because in shogi the captured pieces change sides. Therefore, a minor piece that can be used for mating the opponent king next can be more important than a major piece. Not taking the piece with the highest value or sacrificing a piece of higher value against a piece of lower value is therefore quite common in shogi.

- Promotion

  Unlike in chess, in shogi most pieces can promote. Promoted pieces are more valuable than unpromoted pieces, so a promotion is in general a good move.

- Check

- Escape of a piece attacked by pieces of lesser value

- Change of values based on piece value tables

  In most strong shogi programs, piece value tables are used to estimate the value of a piece based on its relative position to either king. We have used the piece value tables of the shogi program KIRI[5]. These values have been tuned by using a method for tuning evaluation function features developed by Hoki (2006). This method is used in most of the strong shogi programs, which is a strong indication that piece value tables can be learned very accurately by this method. Figure 1 shows the values of a golden general when one's own king is on the square 8h (indicated by the X). The values close to the king, especially those corresponding to the position of the gold in common castle formations like the *Yagura* and *Mino* have higher values than the values corresponding to squares further away from the king. Like most strong shogi programs, our program has a piece value table for each piece, each possible square of one's own king and each possible square of the opponent king.

−62	−108	−64	−70	−88	−98	−88	−124	−98
−116	−112	−4	4	−22	−28	−6	−74	−94
−110	−46	−50	−26	−22	−2	−28	−44	−76
−64	−40	0	−16	10	−18	−8	−4	−50
−98	−52	−26	−18	−22	−14	−20	−26	−60
−78	−38	−20	−26	−20	−18	−14	−44	−66
−104	−20	2	18	−48	−22	−52	−64	−100
−120	X	46	−22	−2	−34	−52	−80	−162
−116	6	−58	8	−82	−30	−86	−74	−160

Figure 1: Bonus values of the gold when one's own king is on the square 8h.

The feature of the move according to this category is decided by the difference between the values before and after moving. This gives a large range of possible values, so we have split this up into the following

[5]KIRI is a strong shogi program developed by the first author. It finished 9th in the 18th World Computer Shogi Championship.

categories to have a limited number of move features: < -100, -100 to -11, -10 to -1, 0, 1 to 10, 11 to 100 and > 100.

- Change of king danger if the king moves

  When the number of attacks on a square adjacent to the king is large, the king is in danger. Therefore, moves that increase the number of attacks by the opponent cannot be considered good moves. King danger can change dramatically if the king moves. Here, we use the change of the number of attacks on the squares adjacent to the king prior to the king move and after moving as a feature. Specifically, there are three categories: the king move leads to having more opponent attacks on adjacent squares, fewer opponent attacks on adjacent squares or no change in the number of attacks on adjacent squares. Furthermore, a difference is being made between a king move when the king is in check and not in check.

- Distance to the destination square of the previous move (or the move before the previous move)

  Because most pieces have only limited movement in shogi, when a piece moves to a certain square it is likely that on the next move an opponent piece close to this destination square will move. The further away pieces are from this destination square, the less likely it is that these pieces will move. The reason for this is that in shogi most pieces have only limited mobility. The goal of moving pieces with limited mobility is therefore often close to where the piece moved. We used the move played immediately before the current position as well as the move that was played just before that as a feature. Distance is divided into three categories: 1, 2 and ≥ 3.

- Move previous piece

  Because of the strategic nature of the opening and middle game in shogi, pieces often move consecutively (an example is advancing the pawn in front of the rook, which increases the mobility of the rook with each move forward).

- Return to previous square

  In shogi, a number of pieces can return to their original square after moving. However, these types of moves are in general not considered to be good moves. Not only is this type of back-and-forth movement a waste of time, it can have an adverse effect on the quality of the simulations because it can cause cycles in the search by returning to a previous position (this is a difference between shogi and Go, where such a problem does not occur). Therefore, unless there is a specific reason (for example, returning by capturing a piece), we want to select these types of moves as few times as possible (i.e. have a low Elo rating).

3.2 Improvements of UCT

Our implementation not only uses the basic UCT algorithm, but also a number of improvements that were proved successful in Go, namely progressive widening, killer moves, the history heuristic, and simple parallelization.

3.2.1 Progressive Widening

By using the Elo rating for moves, it is possible to add progressive widening to UCT. When new nodes are expanded in UCT, not all child nodes are generated at once, but the range of moves being searched is widened progressively. Progressive widening can be realized using the following formula for selecting nodes in UCT:

$$U_i = \begin{cases} dR_i + c'\sqrt{\frac{2\log n}{e}} & (n_i = 0) \\ X_i + c\sqrt{\frac{2\log n}{n_i}} & (n_i \neq 0) \end{cases}$$

$$I = \underset{i \in \{1,\ldots,K\}}{\operatorname{argmax}} \{U_i\}$$

Constant parameters c', d and e were set to 0.5, 4.0 and 10.0 as optimal results of a number of experiments. R_i is the value normalized from the rating of move i by the following formula:

$$R_i = \frac{\gamma_i}{\underset{j \in \{1,\ldots,K\}}{\text{argmax}} \{\gamma_j\}}$$

By using this method, the UCT algorithm will have a preference for expanding moves with a high rating such as recapture and check. If there are moves for which the win rate X_i is high, the range of moves that is being searched is only widened slowly. On the other hand, if the win rate is low, the range of moves is widened quickly.

Note that our method is a modification of the method proposed by Coulom (2007). Coulom's method aims at aggressively pruning the moves deeper in the tree. However, in shogi it is often the case that in order to understand the effects of a tactical move, deeper search is needed. Therefore, we felt that in shogi a method that prunes less aggressively was needed.

3.2.2 Killer Moves and the History Heuristic

We have also used killer moves (Akl and Newborn, 1977) and the history heuristic (Schaeffer, 1983) for move selection. Most traditional game programs based on minimax search use killer moves (the best moves of sibling nodes) and the history heuristic (using the number of times a move was selected as the best move during the search). In Go, RAVE (Rapid Action Value Estimate) (Gelly and Silver, 2007) or AMAF (All Moves As First) are successful methods based on ideas like the killer move and history heuristic.

The difference between RAVE (AMAF) and our approach is that we make a difference between killer moves and the history heuristic, especially emphasizing killer moves. In shogi, many sibling nodes have the same best moves, therefore killer moves are especially effective. In our program, we count all the best moves of the sibling nodes and use the move with the highest frequency as killer move for the current node. In conventional shogi programs, killer moves are very important, so extending UCT with killer moves was expected to have a significant impact. We changed the parameter c in equation (1) based on the killer and history heuristic. If the move is a killer move the parameter c is set to 2.0. The history heuristic is used for every move and the parameter c is set as follows:

$$c = 1.0 + \max(0, H_i - 0.6)$$

where H_i is the ratio of the move i being the best move during the search. As a result of these settings, even if the visiting count is small, killer moves and moves with a high history value will be preferred.

3.2.3 Parallelization and Other Improvements

The number of playouts is strongly related to the overall performance. For games, more playouts mean stronger play. Also, playouts are unrelated so a natural way to improve the performance of UCT is to use parallelization. We have used the simple parallelization scheme of assigning each playout in the shared UCT tree to a parallel thread (Gelly *et al.*, 2006). Although the search behavior is different than when a single thread is used, it has been shown that this method increases the performance if the number of threads that is being used is small (Gelly *et al.*, 2006).

For Lines of Action it has been shown that using information about checkmate is an important improvement of UCT (Winands and Björnsson, 2009). If a checkmate is found in the tree, the score (win or loss) can be determined without doing any playouts. This information about checkmate is also conveyed to the parent node and this blocks unprofitable playouts in certain nodes. More than 99% of all shogi games played between human players ends in checkmate, so this modification is expected to improve the performance, especially in the endgame.

3.3 Opening Book Selection Using Monte-Carlo Tree Search

Another area where MCTS could improve a shogi program is in the opening. Thus far, no effective algorithm for playing the opening in shogi has been found. Most existing shogi programs select moves based on the game records of strong human players. For example, a common method is to use the following formula to calculate the probability that a move i is selected:

$$P(i \text{ is selected}) = \frac{\text{the number of times } i \text{ was selected in the current position according to the database}}{\text{the number of times the current position is in the database}} \qquad (2)$$

Selecting the moves from the database (i.e. the set of game records of strong human players) in this way has some important drawbacks. First, there may be bad moves in the database that are trusted and played by the program. Second, the program could end up in a position that may be easy to play for a human player, but difficult for the computer (an example is a long strategic battle which is especially difficult for shogi programs as was mentioned before).

To solve these problems, UCT can be used. First, UCT is done with the positions in the opening database. That is, as long as the position is in the opening book, instead of generating all child nodes in the tree, only the moves that lead to positions in the opening book will be generated. When the program is out of book, normal UCT is performed with the tree that was built.

By using this method, it is possible to evaluate positions in the opening database on their long-term prospects because the playouts from the opening book will continue until the game is finished. Also, information about the types of positions that the program plays well could be added based on the evaluation function features. This can guide the move selection during playouts.

4. EXPERIMENTS

We will now describe the experiments that were carried out using the UCT based shogi program described in this paper. For all experiments, the test environment given in Table 2 was used.

Table 2: Experimental environment.

CPU	Quad-Core Xeon X5355 2.66GHz $\times$2 (8 Cores)
Memory	2GB

4.1 Characteristics of Playouts Using Elo Rating

Table 3 shows the ratings of move features used in the MCTS shogi program. To estimate these ratings, 300 games from the Meijin tournament were used[6]. Moves with a higher rating are selected more frequently by the human players.

In the table, some values of γ_i are not fixed, but can be within a certain interval. For example, in *Escape*, the value very much depends on which piece is attacking which piece. A rook attacked by a pawn is more likely to move than a rook attacked by a bishop, so has a higher rating.

In the table, it can be seen that moves that lead to a positive material balance, recapturing of pieces, checks and escaping threats to piece with a higher value are the most likely to be played.

Table 4 shows the difference between the playout characteristics of random move selection and rating-based move selection. The *prediction ratio* is the average number of positions where the move played by the human expert was selected by the UCT based program in 200 game records. The *termination ratio* is the percentage of playouts that terminated (i.e. reached the end of the game) within 256 moves for 10,000 playouts from the root position. *Speed* shows the number of playouts per second from the root position for a single CPU.

From these results it can be concluded that even though there is a major loss of speed when using rating for move generation, the prediction ratio and termination ratio are improved significantly. Moreover, the improvement of

[6]The Meijin tournament is one of the seven professional title tournaments and the one with the longest history.

Table 3: Elo ratings of move categories.

Feature	Detail	γ_i			
Material balance (SEE value)	> 650	20.04	Capture	Recapture	12.75
	550 to 650	14.37		Other captures	1.88
	450 to 550	9.54	Change of piece table values (move)	Pawn	1.04 – 2.19
	350 to 450	6.08		Lance	0.16 – 0.90
	250 to 350	3.89		Knight	0.57 – 2.25
	150 to 250	2.90		Silver	0.60 – 3.00
	50 to 150	2.55		Gold	0.33 – 2.22
	-50 to 50	1.11		Bishop	0.66 – 1.45
	-150 to -50	0.47		Rook	0.38 – 1.00
	-250 to -150	0.35		Horse	0.78 – 1.49
	-350 to -250	0.32		Dragon	0.66 – 0.99
	-450 to -350	0.10		Other promoted pieces	1.05 – 2.21
	-550 to -450	0.06	Change of piece table values (drop)	Pawn	0.50 – 2.21
	-650 to -550	0.05		Lance	0.24 – 0.97
	< -650	0.02		Knight	0.17 – 1.66
Check		7.55		Silver	0.19 – 1.35
Promote		1.47		Gold	0.28 – 1.45
Escape	Pawn	1.13 – 1.27		Bishop	0.16 – 1.17
	Lance	1.90 – 3.19		Rook	0.22 – 1.89
	Knight	1.70 – 2.59	Change of king danger if the king moves	Not in check	0.30 – 1.01
	Silver	2.20 – 2.60		In check	0.57 – 4.21
	Gold	4.38 – 4.78	Move previous piece		0.90 – 1.55
	Bishop	1.77 – 5.12	Return to previous square		0.10 – 0.87
	Rook	6.85 – 23.68	Distance to	Previous move	0.89 – 1.97
	Horse	3.26 – 9.98		The move before the previous move	0.94 – 1.86
	Dragon	8.53 – 14.62			
	Other promoted pieces	0.82 – 1.65			

the termination ratio indicates that the problem reported in earlier work about a significant number of playouts being unable to reach the end of the game is no longer an issue.

Table 4: Difference between random move selection and the use of rating.

Move selection	Prediction ratio	Termination ratio	Speed
Random	0.037	0.193	approx. 3,500 playouts/second
Rating	0.172	0.905	approx. 900 playouts/second

In the experiments that follow, playouts that did not terminate were given the draw score of 0.5. The alternative is to use an evaluation function as a tie breaker, but because the use of an evaluation function as a tie breaker has not been successful in shogi (Hashimoto *et al.*, 2006) we decided against this. Also, this requires the selection of an appropriate evaluation function and experiments about how to set the window for the win-loss decision. We feel that such details are outside the scope of the current paper.

4.2 Results for Tactical Shogi Positions

Table 5 shows the results for a set of 98 tactical shogi problem positions. This set is a combination of the problem positions in (Matsubara, 1998) and (Shogi Town, 1999). The program was given 30 seconds for each problem. *MC/UCT* is the baseline MCTS program based on UCT where *MC/UCT(1)* is a version running on a single core and *MC/UCT(8)* is the parallel version running on 8 cores.

Check is a version that uses information of checkmate in the UCT tree, *rating* is a version that uses Elo ratings in playouts, *PW* is a version that uses progressive widening, *killer* is a version that uses killer moves and *history* is a version that uses the history heuristic.

The results are compared to those of the program TOHMI, which is the predecessor of KIRI. It is is slightly weaker than KIRI and is estimated to have a strength of about 1-dan amateur[7]. TOHMI is a conventional program based on search and an evaluation function and is not using MCTS. Also, it has a special mate solver, which gives it an advantage in the 12 mating problems that are in the test set. Therefore, we have also given the results for the 86 problems that are not mating problems.

Table 5: Results for tactical problem positions.

Method (number of threads)	Solved	Solved (excluding mating problems)
MC/UCT(1)	4 / 98	4 / 86
MC/UCT(8)	9 / 98	8 / 86
MC/UCT(8) + check	12 / 98	11 / 86
MC/UCT(8) + check + rating	36 / 98	32 / 86
MC/UCT(8) + check + rating + PW	41 / 98	38 / 86
MC/UCT(8) + check + rating + PW + killer	43 / 98	41 / 86
MC/UCT(8) + check + rating + PW + killer + history	49 / 98	45 / 86
TOHMI	60 / 98	48 / 86

As can be seen from the table, the number of solved problem positions increases for each modification, which indicates that all proposed enhancements are improvements. Also, when looking at the results for the non-mating problem positions, we see that the number of solved problems for the program version with all the enhancements is close to TOHMI. Based on the results in positions where tactics play a role (which happens often in shogi), the strength of the MCTS program is estimated to be about 1-dan amateur. Furthermore, the results also show that MCTS is not well-suited to solve mating problems because this requires exhaustive search.

In the rest of the experiments, we used *MC/UCT(8) + check + rating + PW + killer + history*, which will be abbreviated to MCTS.

4.3 Match Results

Table 6 shows the results of two matches played between MCTS and TOHMI. Each match consisted of 200 games and each program was given 10 seconds per move.

Table 6: Match results of MCTS against TOHMI.

Method	Win rate
MCTS	0.04
MCTS (using quiescence search)	0.32

Despite having similar results in tactical shogi positions, MCTS won only 4% of its games against TOHMI. The reason for this is that it is hard for MCTS to evaluate minor material losses, especially in the opening and middle game. Giving up a pawn without compensation early in the game has little impact on the results of the playouts, but will have the MCTS program at a long-term disadvantage. In tactical shogi positions, these minor losses do not really matter because there is a single move that leads to a clear advantage.

Another problem we observed was that bad moves that are counting on a mistake by the opponent were often selected in positions where MCTS was already at a disadvantage. To deal with this problem, we added quiescence search[8] to the MCTS program using the following ad-hoc approach. For each move played from the root position, if the value of the quiescence search was lower than -700, the win rate was reduced by 15%. If the value of the quiescence search was between -700 and -420, the win rate was reduced by 10%. Finally, if the value of the quiescence search was between -420 and -100, the win rate was reduced by 5%. Although we did a number of experiments to come up with reasonable values, detailed experiments aiming to find the optimal values have not yet been carried out.

The results in Table 6 show that MCTS is winning a lot more games against TOHMI if quiescence search is added, but it is also clear that MCTS is weaker than TOHMI and is therefore not close to a 1-dan amateur player

[7]TOHMI finished 19th in the 17th World Computer Shogi Championship.

[8]The evaluation in the quiescence search is based on the piece values in Table 1.

as suggested by the results in tactical positions. Our conclusion is that it will be difficult for MCTS based methods to surpass the playing strength of conventional methods in shogi.

4.4 Opening Book Selection Using Monte-Carlo Tree Search

Next, we compared the MCTS based opening book selection with a method using probability. For this experiment we used KIRI. As long as the game was still in the opening book, one program version would select the next move based on MCTS and one program would select the next move based on the probability method used in KIRI, which uses Equation (2). When the program was out of book, the game was played to a finish using KIRI for both program versions.

In this experiment, each program was given 2 seconds per move and a 200 game match was played. [9] The opening book consisted of 30,000 games played by human expert players (professional players and strong amateurs). We only used those moves in the opening book that eventually lead to a win for the human expert who played it. The result of the match is given in Table 7. Here *Probability* means that the selection of moves from the opening book was based on probability using Equation (2).

Table 7: Match result of opening book selection using MCTS against a probability based method.

Comparative method	Win rate	Average evaluation function value
Probability	0.61	−2.01

The result shows that opening book move selection using MCTS is superior to a method using probability and that MCTS is a promising new method for opening book move selection. Interesting is that the average evaluation function value given by KIRI after the program was out of book is close to zero (the value of a pawn is 100). This indicates that evaluating positions in the opening is very difficult in shogi and that search-based opening book selection methods may not improve as long as a database of expert games is being used as opening book.

4.5 MCTS in the Endgame

In this section, we will discuss the effectiveness of MCTS in the endgame. As an example, we will analyze the endgame of the game between top professional Watanabe[10] and BONANZA[11] (Daiwa Cup, 2007). This game was the first official encounter between a top professional player and the computer. Until the middle game, the game was even but in the endgame there were a number of positions where the weaknesses of current shogi programs showed and in the end BONANZA lost the game.

The first position where BONANZA made a mistake is given in Figure 2, where BONANZA selected P*2d, an all-out attacking move. However, this attack will not be in time and as Watanabe pointed out after the game, after L*2g instead, the position would have been unclear (Watanabe, 2007).

In the position of Figure 2, both players have made a *Bear-in-the-Hole* castle. In positions like this, in order to make the right move decision, it is necessary to search specific nodes very deeply. With conventional search and evaluation, it is hard to play these positions accurately because the position cannot be searched deep enough.

Table 8 shows the results of our MCTS program when given 60 seconds to analyze the position. Here, the *Evaluation* is the win rate of the move in the UCT playouts. The MCTS program selects L*2g as the best move. Because MCTS playouts are basically deep searches, this method is particularly suited for these types of positions.

The position in Figure 3 is already lost for BONANZA. However, most shogi programs, including BONANZA still think the position is about even. This is caused by BONANZA's material advantage and the difficulty of evaluating that Watanabe's attack cannot be stopped.

[9]The reason for having less time per move compared to the earlier experiments is to reflect that in general less time is used for selecting moves during the opening than for moves in the middle game and endgame.

[10]At the time when this game was played, Watanabe held the prestigious title of *Ryu-O*, which is one of the seven major professional titles in shogi.

[11]BONANZA is one of the top shogi programs and winner of the 16th World Computer Shogi Championship.

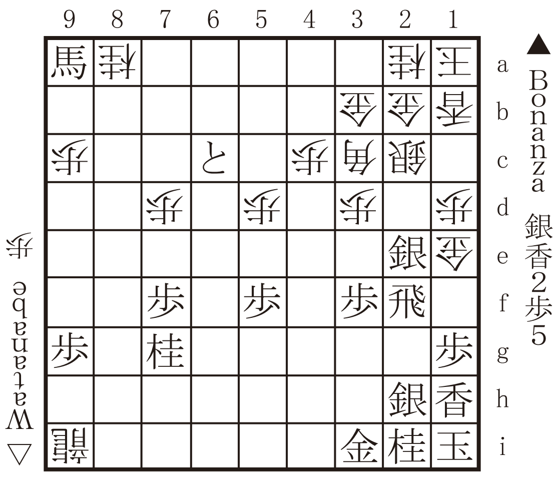

Figure 2: Watanabe vs. BONANZA(1)

Table 8: Evaluation of the position in Figure 2 using Monte-Carlo Tree Search.

Order	Move	Evaluation
1 (best)	L*2g	0.471
2	+B3g	0.412
3	S*3g	0.398
...	...	...
21	P*2d	0.301
...	...	...

Table 9 gives the results of the MCTS program for the position of Figure 3. Without going into the details of the moves, the important result in this table is that all of the moves get an evaluation of about 30%. For MCTS it is therefore clear that this position is lost.

The problem for conventional programs here is that in the position of Figure 3 the features evaluating the danger to the king and the turn to move are more important than material. Most shogi programs use a measure of progress to adjust the balance between king danger and material, but in complex positions like the one given here, this adjustment cannot be done accurately.

MCTS evaluates positions based on win rate and therefore does not depend on game-specific heuristic evaluation like material, king danger, relative piece position and turn to move. As a result, MCTS enables a more general positional assessment than existing methods.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a shogi program based on Monte-Carlo Tree Search. Compared to earlier work, the performance of the program was improved using techniques that have been successful in Go. In playouts, we used move selection using Elo ratings, which solves the problem that many playouts will not reach the end of the game. We also improved UCT in shogi by adding progressive widening, killer moves, the history heuristic and simple parallelization. In tactical shogi positions, the performance of naive MCTS is poor, but our enhancements improved the performance in such a way that the results were similar to an amateur 1-dan program. This showed that improvements that were successful in Go are also effective in shogi. Furthermore, it shows that MCTS can be an alternative to conventional search methods in chess-like games.

MCTS was convincingly beaten by a conventional shogi program of intermediate strength. Shogi programs have become very strong recently, expecting to beat the top human players within ten years. Therefore, it seems unlikely that a pure Monte-Carlo-based shogi program will surpass the top programs. However, we also showed that MCTS could outperform conventional methods in the opening and in certain types of endgame positions.

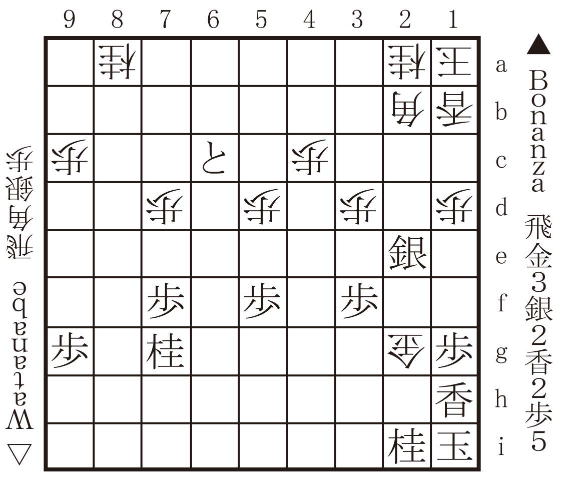

Figure 3: Watanabe vs. BONANZA(2).

Table 9: Evaluation of Figure 3 using Monte-Carlo Tree Search

Order	Move	Evaluation
1 (best)	G*2h	0.334
2	S*3g	0.326
3	S*3i	0.315
...	...	...

This suggests that a combination of conventional search and MCTS could improve the overall strength of a shogi program, just like specific mate solvers are now improving the overall strength of conventional shogi programs.

Most existing shogi programs are based on similar methods, which means that they have similar strengths and weaknesses. Particularly the weaknesses can be studied and taken advantage of by strong human players. To avoid this, incorporating a completely different method like MCTS into an existing shogi engine has the potential for erasing these weaknesses. This is not only true for shogi, but could very well be the case in other games as well.

As future work, we are planning to create a hybrid shogi program combining conventional search with MCTS.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments on the earlier version of this paper.

6. REFERENCES

Akl, S. and Newborn, M. (1977). The Principal Continuation and the Killer Heuristic. *ACM'77: Proceedings of the 1977 annual conference*, pp. 466–473.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, Vol. 47, pp. 235–256.

Cazenave, T. and Saffidine, A. (2009). Utilisation de la recherche arborescente Monte-Carlo au Hex. *Revue d'Intelligence Artificielle*, Vol. 23, Nos. 2–3, pp. 183–202. (in French).

Ciancarini, P. and Favini, G. (2009). Monte Carlo Tree Search Techniques in the Game of Kriegspiel. *Proceedings of the 21st International Joint Conference on Artifical intelligence*, pp. 474–479.

Coulom, R. (2006). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *5th International Conference on Computer and Games, LNCS 4630*, pp. 72–83, Turin, Italy.

Coulom, R. (2007). Computing “Elo Ratings” of Move Patterns in the Game of Go. *ICGA Journal*, Vol. 30, No. 4, pp. 198–208.

Daiwa Cup (2007). http://www.daiwashogi.net/. (In Japanese).

Gelly, S. and Silver, D. (2007). Combining online and offline knowledge in UCT. *ICML ’07: Proceedings of the 24th international conference on Machine learning*, pp. 273–280.

Gelly, S., Wang, Y., Munos, R., and Teytaud, O. (2006). Modification of UCT with Patterns in Monte-Carlo Go. Technical Report RR-6062, INRIA.

Hashimoto, J., Hashimoto, T., and Nagashima, J. (2006). A Potential Application of Monte-Carlo Method in Computer Shogi. *The 11th Game Programming Workshop*, pp. 195–198. (In Japanese).

Hoki, K. (2006). Optimal Control of Minimax Search Results to Learn Positional Evaluation. *The 11th Game Programming Workshop*, pp. 78–83. (In Japanese).

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo Planning. *15th European Conference on Machine Learning (ECML)*, pp. 282–293.

Lorentz, R. (2008). Amazons Discover Monte-Carlo. *6th International Conference on Computer and Games*, pp. 13–24, Beijing, China.

Matsubara, H. (1998). *Computer Shogi Progress 2.* Kyoritsu Shuppan. (In Japanese).

Schaeffer, J. (1983). The History Heuristic. *Journal of the International Computer Chess Association*, Vol. 6, No. 3, pp. 16–19.

Shogi Town (1999). http://www.shogitown.com/school/judge/judgetop.html. (In Japanese).

Sturtevant, N. (2008). An Analysis of UCT in Multi-Player Games. *ICGA Journal*, Vol. 31, No. 4, pp. 195–208.

Watanabe, A. (2007). http://blog.goo.ne.jp/kishi-akira/. (In Japanese).

Winands, M. and Björnsson (2009). Evaluation Function Based Monte-Carlo LOA. *Advances in Computer Games, LNCS 6048*, pp. 33–44.