Improving Strategic Play in Shogi by Using Move Sequence Trees

Reijer Grimbergen
Department of Informatics, Yamagata University
Jonan 4-3-16, Yonezawa-shi, 992-8510 Japan
E-mail: grim@yz.yamagata-u.ac.jp

Abstract

The main weakness of shogi programs is considered to be in the opening and the middle game. Deep search is not enough to cover the lack of strategic understanding, so most strong shogi programs use a hill-climbing approach to build castle and assault formations. The problem of a hill-climbing approach is that if the final result of two different paths is the same, then the final score will also be the same. In shogi, this leads to high priority moves being pushed down the principal variation and in extreme cases such moves will never be actually played. To solve this problem, *move sequence trees* are proposed, which guide the program through the right order of moves. This approach was tested in the *yagura* opening. A preliminary self-play experiment shows that using move sequence trees could be an enhancement of hill-climbing approaches for strategic play in shogi.

Keywords: Move sequence trees, otoshiana method, hill-climbing, strategic play, computer shogi.

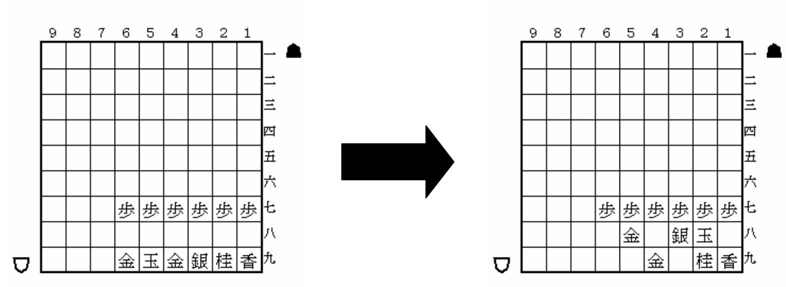
1 Introduction

In recent years, shogi programs have become strong enough to beat all but the strongest human shogi players. Despite this, shogi programs are considered weak in the opening and middle game due to a lack of strategic awareness. This is caused by the fact that shogi pieces have only limited movement, so the opening phase of a game of shogi consists of building a piece formation that is in a large part independent of the piece formation the opponent is building. The problem of playing the opening and the early middle game in shogi is that even though the opponent moves can often be ignored, this is not always possible. When a piece formation is not completely finished yet, the position is actually quite vulnerable, often having several weaknesses. Professional shogi players spent a lot of time (studying at home or while playing a game) to find ways to explore these weaknesses. Sometimes a small difference in the order in which a piece formation is built can mean the difference between winning and losing a game.

In contrast, current shogi programs in general ignore the strategic subtleties of the opening phase in shogi. Programmers spent a lot of time tuning their programs to reach a middle game position that is not obviously bad. The programs are strong enough after that to have a good chance of winning from a reasonable middle game position against almost all human players. However, this may not be enough to take the final steps to beating the best professional shogi players. The strategic deficiencies of game programs are expected to be explored by the top players. Therefore, this is one of the most important topics in computer shogi. If the opening problems of the programs can be resolved, shogi programs will no longer have weaknesses and should be ready in a few years to challenge the top human players.

Of course, this is not an easy task. Building piece formations in shogi can take a long time. For example, to finish an *anaguma* castle formation requires 13 moves by one player, so from the starting position a 25 ply search is needed to just find a complete formation. Add to this that the opening is not only about playing a castle formation, but also about developing other pieces, and it is easy to imagine that doing search until a desired piece formation is reached is beyond the abilities of current search algorithms.

In shogi programs, methods based on hill-climbing have been proposed to improve the opening play in shogi ([3], [1]), but these methods are not the final solution to the problem. In this paper, a method based on *move sequence trees* will be proposed that is expected to be an enhancement of the current hill-climbing methods. The content of the rest of this paper will be as follows. In Section 2, the current hill-climbing methods will be reviewed. In Section 3 the extension of these methods using move sequence trees will be explained. In Section 4



0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	**3**	**6**	**9**	0
0	0	0	0	0	0	0	0	0

玉（美濃囲い）

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0
0	0	0	0	0	**9**	0	0	0

金（美濃囲い）

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	**2**	0	0
0	0	0	0	0	0	0	0	0

銀（美濃囲い）

Figure 1: Hill-climbing in the opening: the *otoshiana* method.

some preliminary results using move sequence trees in a limited part of the shogi opening will be presented. Finally, in Section 5 the conclusions and ideas for future work will be given.

2 Current Methods

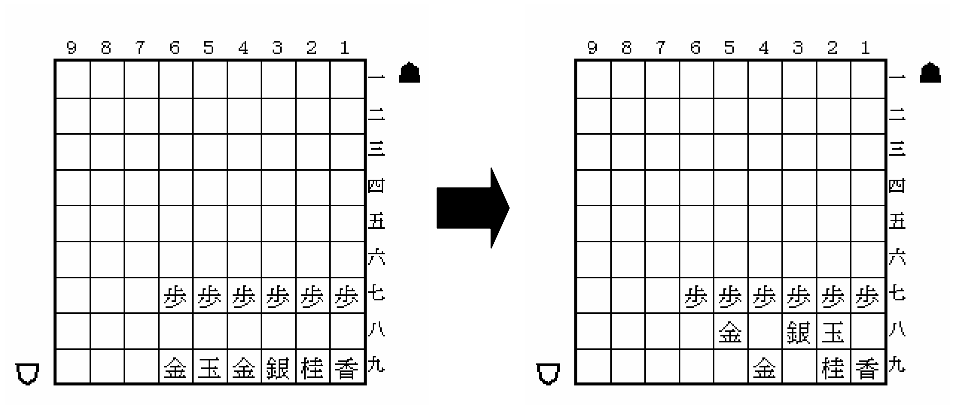
The method that is used most often to guide a shogi program through the opening is the *otoshiana* method [3], which was first used by Hiroshi Yamashita in his program YSS, the reigning World Champion. This method is basically a hill-climbing approach. For each piece a 9×9 table is defined, which assigns a value to each square on the shogi board where the piece is placed. For example, for the *mino* castle in Figure 1, suppose that there are three maps for the king, gold and silver respectively. In this case, considering the values given in these maps, improving the position of the pieces would be in the following order (the values over the arrows indicating the improvement):

59K $\overset{+3}{\Longrightarrow}$ 48K, 48K $\overset{+3}{\Longrightarrow}$ 38K, 38K $\overset{+3}{\Longrightarrow}$ 28K, 39S $\overset{+2}{\Longrightarrow}$ 38S, 69G $\overset{+1}{\Longrightarrow}$ 58G

Note that the right gold is not moving, because it is already on its optimal square in the starting position. The values in the table are added to the piece values in the evaluation function, so even if the formation is not completed, pieces will be awarded for being "on their way" to their optimal square. In this example, the king will get 3 points for being on 48, and 6 points for being on 38. This is better than the 0 points for being on 59, so the program will prefer to move the king in the right direction even if the search is not able to go deep enough to put the king on the optimal square 28.

Looking at this natural sequence to move into a castle formation, it seems that this method is a very good way of guiding a shogi program through the opening. Some additions were proposed, for example using the same method to guide the building of attacking formations [1], and this method quickly became the most popular way for shogi programs to play the opening. An important reason for this is that by using this method, the program has a tendency to restore broken formations if for some tactical reason the proper building of the piece formation has been disrupted.

Despite its popularity, there are two important problems with the *otoshiana* method. First, even though it works well with shallow search, in the case of deep search there is the problem of having different move sequences all leading to the same final score. An example is given in Figure 2. Here, all



1） 4八玉　3八玉　2八玉　3八銀　5八金(69)

2） 3八銀　4八玉　5八金(69)　3九玉　2八玉

3） 5八金(69)　3八銀　4八玉　3九玉　2八玉

:

:

Figure 2: Problem with hill-climbing methods (1): different move sequences leading to the same result.

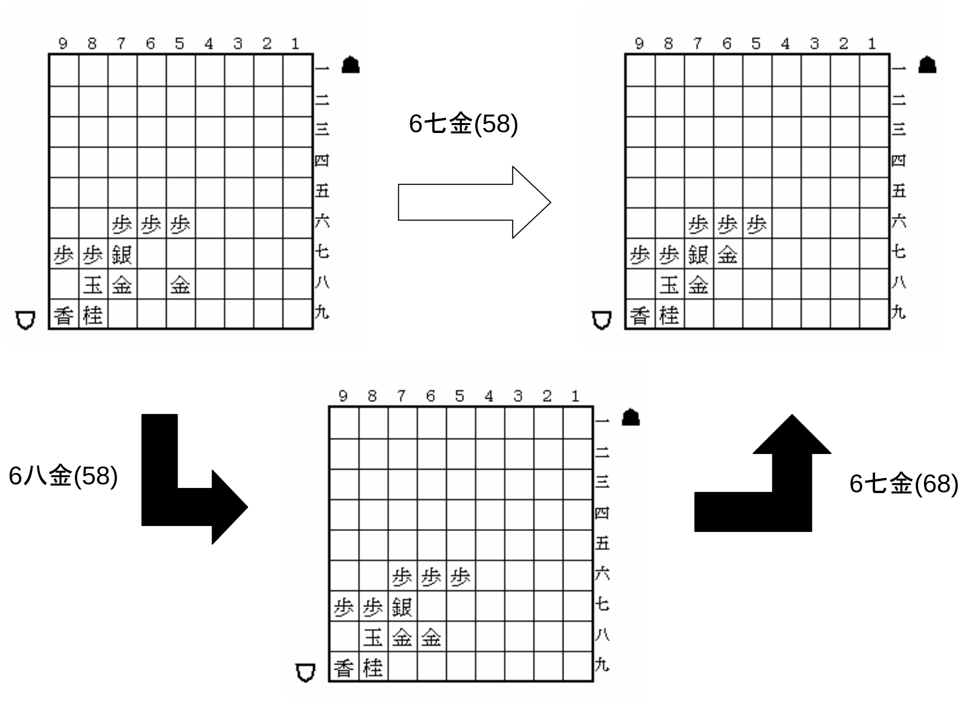


Figure 3: Problem with hill-climbing methods (2): losing tempo.

three move sequences will lead to same final score: the king on 28 will get 9 points , the gold on 58 will get 1 point, the gold on 49 will get 9 points and the silver on 38 will get 2 points for a total of 21 points. Because the evaluation function is only applied in the final position, for a program using the *otoshiana* method there is no difference between these three variations because they all lead to the same position and thus to the same score. However, from a human standpoint these variations are very different. Now if the program would actually always be able to reach this final position there would be no problem but unfortunately this is not the case. When using the *otoshiana* method, it can often be seen that a vital move that would make the position safe ends up at the end of the principal variation, never making it to the top and be selected as the next move.

The second problem of the *otoshiana* method is that during the search there are two ways to reach the same position, but one of these is taking one or more extra moves. An example of this is given in Figure 3 where the gold can go straight to 67 or first go to 68 and then go to 67. Again, even though the moves are different, the final position that is reached is the same, so for the evaluation function there is no way to tell the difference.

Both of the problems are caused by a general flaw of hill-climbing methods: only the final result counts, not the way this result was reached. In shogi terms: only the positions are being evaluated, but not the moves. To improve the *otoshiana* method, it is important to add information about moves. In the next section, the method of *move sequence trees* will be proposed to add information about moves that will solve the two problems given here.

3 Move Sequence Trees

The first thing that needs to be decided is what kind of move information is helpful for strategic play in shogi. Strong shogi players do not have all opening moves they have ever seen readily available in memory. To decide which moves are relevant in a position and which moves are not, shogi games are categorized using the opening strategy by each player.

Therefore, shogi players often make comments like: "This is a typical move for the yagura". This shows how higher level concepts like yagura are linked to the primitive information about moves. By categorizing a position into these higher level concepts, the number of moves that need to be considered can be kept to a minimum.

This categorization will be the basis for generating and storing move sequences. After this generation is finished, the move sequences will have to be accessed in the search to guide the program along the variations of the move sequences that have been stored. These two steps will now be explained in detail.

3.1 Extracting Move Sequences

Shogi games are categorized using the opening strategy played by each player, which is usually defined by the type of castle that is being played (*mino*, *yagura*, *anaguma*, etc.). The first task is therefore to extract move sequences in each type of opening strategy. This can be done in three steps:

1. Collecting games with a specific opening strategy
2. Collecting move frequency data for this strategy
3. Use the moves and the move frequency data to generate a move sequence tree, which contains all information about the move sequences played in this particular opening strategy.

These three steps will now be explained using the *yagura* opening as an example.

3.1.1 Collecting Games

The 240,000 games of the Shogi Club24 database [2] are the best way to collect a large number of games with a specific opening strategy. The moves are considered relevant to an opening strategy if the player was able to build a complete castle formation. Therefore, for the *yagura* move sequences, the database was scanned for those games where one of the players completed a full *yagura* castle.

There are 1815 games in the database where the sente player built a complete *yagura* castle. Added to this were 733 games by professional players, giving a total of 2548 games as the basis for the analysis of moves and move sequences in the *yagura* opening by the sente player.

For the gote player there were 1520 games in the database and adding 516 games by professional players gives a total of 2036 games.

3.1.2 Collecting Move Frequency Data

The next step is to collect move frequency data from the games. This is important to make a difference between moves that are typical for a *yagura* opening and moves that were tried only a few times or were strategically suspect but forced because of tactical reasons. Moves with a low frequency should not be

Table 1: Move frequency data for the yagura opening played by sente.

Move	Fr (%)	Move	Fr (%)	Move	Fr (%)	Move	Fr (%)
7六歩(77)	100	6八銀(79)	78	4六歩(47)	32	9六歩(97)	20
6六歩(67)	99	7七銀(68)	78	3五歩(36)	29	7五歩(同)	16
5八金(49)	99	1六歩(17)	72	6八玉(59)	27	6五歩(66)	15
7八金(69)	98	6九玉(59)	71	1五歩(16)	26	4七銀(48)	13
6七金(58)	97	8八玉(79)	71	2四歩(25)	24	5五歩(同)	12
5六歩(57)	91	7九玉(69)	69	5七銀(48)	24	8六銀(77)	11
4八銀(39)	89	2五歩(26)	60	3八飛(28)	23	7七銀(88)	11
2六歩(27)	89	6八角(79)	47	4六銀(37)	23	5五歩(56)	10
3六歩(37)	84	3七銀(48)	45	7八玉(68)	21	5八飛(28)	10
7九角(88)	83	3七桂(29)	38	8八玉(78)	21		

part of the move sequence trees that will be used during search or at least get a lower priority than moves that have been played with a high frequency.

In the 2548 games leading to a complete *yagura* castle for the sente player, there were a total of 1288 different moves played in the first 50 moves of each game. When analysing the frequency of these moves, only 153 moves were played in 1% or more of the games. As an example of the moves that can be considered typical *yagura* moves for the sente player, the 39 moves that were played in more than 10% of the games are giving in Table 1.

3.1.3 Generating Move Sequence Trees

Finally, the information of the order of moves and the frequency of the moves is stored in a move sequence tree. In this tree, all moves that have been played in sequence in any of the games are stored with a connection between them. Also, the move frequency of the moves is stored in the move sequence tree. An example of a small part of a move sequence tree for the *yagura* opening played by the sente player is given in Figure 4.

To avoid that the generated trees become too large, move transpositions need to be detected. During the generation of the tree, hashcodes are used to detect transpositions. This makes sure that the size of the trees stays within reasonable limits. Despite this, the move sequence tree that is generated in this way can become quite large. For example, the tree that was generated for sente in the *yagura* opening has 21820 nodes (for gote this is 19028 nodes).

3.2 Using Move Sequence Trees

Because of the size of the move sequence trees, it is not possible to keep all of them in memory at the same time. After the program is out of book, only the move sequence tree that is relevant to the current position is loaded. The selection is done by looking at the moves that were played by the sente side or gote side to reach the current position. If the move sequence that lead to the current position is in the move sequence tree, this move sequence tree is considered relevant to the current position and selected.

There might be multiple move sequence trees that are relevant, but to keep the current implementation as simple as possible, only one move sequence tree is loaded. To decide which tree is picked if there are multiple alternatives is an interesting issue. A good idea seems to pick the move sequence tree where the move frequency of the moves that can be played in the current position is highest. However, in this paper only partial results using the move sequence tree for sente and the move sequence tree for gote in the *yagura* opening are presented, so this decision procedure has not been resolved yet.

Once a move sequence tree is selected, it needs to be accessed in the tree. Our implementation has a pointer to the move sequence tree in every node of the search tree. If a move sequence is followed during the search, the pointer is updated. If a move sequence is abandoned during search, the pointer is set to null. By using this pointer it is possible to detect if the search is following a variation that is in the move sequence tree. For each move that is following a variation in the move sequence tree, a bonus is awarded. The total bonus for the moves

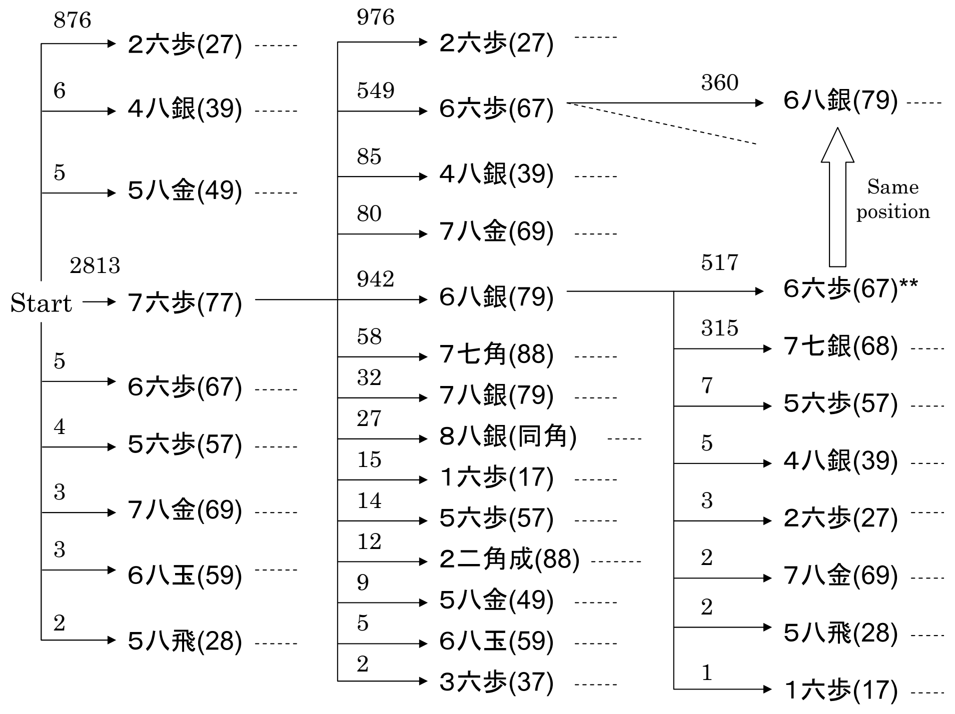


Figure 4: Partial move sequence tree for sente with move frequencies. (Note: (**) is a terminated sequence because of move transposition).

is added to the evaluation function value for positions at leaf nodes. It seems clear that the bonus value should be based on the move frequency of the moves in the move sequence tree, but our first implementation just adds a fixed bonus for each move played according to the move sequence tree.

Note that even though this process will slow down the search a little, it will only be used in the opening and early middle game phase of the game where the number of moves is still small and deep search was possible anyway. Therefore, the overhead of this method is not significant.

4 Experimental Results

As explained, the current implementation only has the move sequence trees for sente *yagura* and gote *yagura*. As a first test to evaluate the new method, self-play experiments were done between a program only using the *otoshiana* method and programs using both the *otoshiana* method and the move sequence trees with different move bonuses. These experiments were to evaluate if move sequence trees could improve the strategic play of a program using a hill-climbing approach.

Because there are only the move sequence trees for the *yagura*, the self-play experiments all started with positions that were reached after 10 moves from the initial position, where in each position the move sequence tree had been followed, i.e. a *yagura* opening according to the move sequence tree was still possible for either side. Time limitations only allowed for a limited number of games to be played. 25 positions were selected as starting positions for the experiment and both programs played sente once in each of these positions, so a total of 50 games was played. The time limit was 10 minutes per side.

Four versions of the program using move sequence trees in combination with the *otoshiana* method were used. Each version assigned a different, fixed bonus for moves that were in the move sequence tree. In version *MST25* the bonus was a quarter of the value of a pawn (in the test program SPEAR, the value of a pawn is 100 points), in version *MST50* the bonus was half a pawn, in version *MST75* the bonus was three quarters of a pawn and in version *MST100* the bonus was a the value of a pawn[1]. The results of these self-play experiments are given in Table 2.

[1]Because in shogi pieces don't disappear from the game after being captured, the evaluation value swing of a pawn exchange is actually twice the value of a pawn, i.e. 200 points.

Table 2: Results of the self-play experiments. All matches are against a program only using the *otoshiana* method. *WP* is the winning percentage of the programs using move sequence trees. *MST25* means that the bonus given to moves was 25 points, which is a quarter of the value of a pawn. In the column *Yagura*, the number of times a complete *yagura* formation was build by the program using move sequence trees (*MST*) and the program only using the *otoshiana* method (*OTO*).

Version	*Won*	*Lost*	*WP*	*Yagura*	
				MST	*OTO*
MST25	23	27	46%	20	21
MST50	27	23	54%	32	20
MST75	28	22	56%	27	15
MST100	25	25	50%	23	15

Although the results of the self-play experiments are not conclusive, they seem to indicate that using move sequence trees in combination with the *otoshiana* method can improve the playing strength of a program. The program versions with a bonus value of half a pawn and a quarter of a pawn both won their matches against the program that was only using the *otoshiana* method. The results of 27-23 for *MST50* and 28-22 *MST75* statistically only give a probability of 71% that *MST50* is stronger than using only the *otoshiana* method and a 80% probability that *MST75* is stronger. Matches with more games are needed to reach a conclusion about the merits of using move sequence trees.

From the self-play experiments there is a different piece of data that seems to indicate that using move sequence trees is an improvement. For each match, the number of times a complete *yagura* castle was built by either side was counted. The results are given in the column *Yagura* of Table 2. It was found that *MST50* managed to complete the *yagura* castle in 32 games, while the program only using the *otoshiana* method only completed the *yagura* 20 times. *MST75* also managed to complete the *yagura* more often: 27 times while the *otoshiana* method only completed the *yagura* 15 times.

Finally, from the self-play experiments it is clear that if move sequence trees are used, a bonus between half a pawn and a quarter of a pawn is best. *MST25* lost its match 23-27 and *MST100* was unable to win the match, indicating that a bonus of a full pawn is too much. Furthermore, *MST25* did not build a complete *yagura* more often than the program using only the *otoshiana* method, indicating that the bonus of a quarter of a pawn is too small to guide the program in the right direction. *MST100* built a complete castle more often than the *otoshiana* method, but less often than *MST50* and *MST75*. This indicates that the bonus of a full pawn is too large, leading to tactical oversights (like giving away a pawn). The games then become early fights and neither side is able to finish the castle formation.

5 Conclusions and Future Work

In this paper, a method using move sequence trees was proposed to deal with the problems of the hill-climbing methods used by shogi programs in the opening and early middle game. The idea of this method is to add evaluation not only to positions, but also to moves that lead to these positions. To do this, move sequences and information about move frequencies in common shogi opening strategies are extracted from a large database of shogi games and this information is stored in a move sequence tree. This tree is then accessed during normal search and an evaluation bonus is given to moves that follow a sequence in a move sequence tree.

Preliminary results for a partial implementation of this method were given and even though these first results are encouraging, further experiments are needed to reach a conclusion about the feasibility of the method.

The next step is to generate move sequence trees for all the major opening strategies. Also, the evaluation bonus attached to moves (which is currently a fixed value) should be based on the move frequency in the move sequence tree. Finally, it was sometimes observed that even though the move order from which a position resulted was not in the move sequence tree, the partial position (only sente pieces or only gote pieces) was in the database. Therefore, the initial selection of move sequence trees should not only be based on the moves that lead to the current position, but on whether the partial hashcode of the position is in the move sequence tree.

The proposed method may not be the solution to the problem of making a shogi program that can play strategically at the same level of top human players. However, based on the preliminary results presented here, a full implementation of move sequence trees may be a significant enhancement of the *otoshiana* method.

References

[1] R. Grimbergen and J. Rollason. Board Maps and Hill-climbing for Opening and Middle Game Play in Shogi. In J.Schaeffer, M.Muller, and Y.Bjornsson, editors, *Computers and Games: Proceedings CG 2002. LNCS 2883*, pages 171–187. Springer Verlag, Berlin, 2002.

[2] H. Kume. *Shogi Club 24 Saikyou Kifu Database*. Sekio Shobo, Japan, 2004. CD-ROM, (In Japanese).

[3] H. Yamashita. YSS: About its Datastructures and Algorithm. In H. Matsubara, editor, *Computer Shogi Progress 2*, pages 112–142. Tokyo: Kyoritsu Shuppan Co, 1998. ISBN 4-320-02799-X. (In Japanese).