Journal of Information Science and Engineering 38, 591-603 (2022)
DOI: 10.6688/JISE.202205_38(3).0006

Improving Mini-Shogi Engine Using Self-Play and Possibility of White's Advantage*

Masahiro Shioda and Takeshi Ito
Graduate School of Informatics and Engineering
The University of Electro-Communications
Tokyo, 182-0021 Japan
E-mail: shioda@minerva.cs.uec.ac.jp; taito@mbc.nifty.com

The artificial intelligence (AI) in Shogi has made rapid progress recently, owing to the recent establishment of a method of learning evaluation via self-play. In this paper, we applied this method to Mini-Shogi to verify the effect. Specifically, we used YaneuraOu Shogi engine to develop the Mini-Shogi program and trained a neural network-based evaluation function. Our program won all competitions in which we participated in 2020. Moreover, the experimental results suggest the second-move (White) advantage in Mini-Shogi.

Keywords: artificial intelligence in game, evaluation function, machine learning, self-play, Shogi

1. INTRODUCTION

Zero-sum games have been the targets of study in the field of artificial intelligence (AI) for a long time. In particular, the study of computer Shogi is very popular and has been mainly conducted in Japan. Ponanza defeated a professional Shogi player for the first time in 2013, swept all professionals since 2016, achieving superhuman performance. One of the reasons is the establishment of a self-play method of learning evaluation. Mini-Shogi is played on a 5×5 board, and the state-space is much narrower than that of standard Shogi. There are no experts in Mini-Shogi, and it is commonly played at competitions where computer engines compete against each other: UEC-cup Computer Mini-Shogi Tournament is an event held by the Entertainment and Cognitive Science Research Station since 2007. However, until recently, there has been no evidence to suggest that which player has an advantage. Although the machine learning method described previously may also be effective in games similar to Shogi, it has not yet been introduced to Mini-Shogi. Thus, we expect they will contribute well to the improvement of Mini-Shogi engine's strength.

This paper extends a previous work [1], and we applied the described method to Mini-Shogi, training its neural network-based evaluation function from scratch

Received February 2, 2021; revised March 21 & May 17, 2021; accepted June 25, 2021.
Communicated by Yasufumi Takama.
* This work was supported by JSPS KAKENHI Grant No. 18H03347.

to make a sufficiently strong Mini-Shogi program. We train an efficiently updatable neural network (NNUE) as our conventional evaluation function. NNUE has contributed to the improvements in strength in both computer Shogi and computer chess, and we believe that its superiority is also reflected in our experiment. Comparing the strength against leading programs, our program is found to perform comparatively well, and it shows high performance for the White side in particular. We also investigate the bias of win rate between Black and White.

2. MINI-SHOGI

2.1 About Mini-Shogi

Mini-Shogi was invented around 1970 in Japan and is derived from Shogi (Japanese chess). A board is composed of 25 squares arranged into five rows and five columns. Each player[1] has six pieces (*i.e.*, one king, one rook, one bishop, one gold general, one silver general, and one pawn). At the beginning of the game, pieces are placed as shown in Fig. 1. The promotion zone is the last rank farthest away from each player. The rules of Mini-Shogi are nearly the same as those of Shogi.

When we calculated the average branching factor and the average game length from records of the UEC-cup Mini-Shogi Tournament, the results were 21 and 60, respectively. The game-tree complexity of Mini-Shogi was computed to be 10^{80}; thus, we estimated that it is larger than Othello (10^{60}) and smaller than chess (10^{120}).

2.2 Studies Using Mini-Shogi

Mini-Shogi retains the characteristics of Shogi but has a much narrower state-space. Therefore, Mini-Shogi has been used as a testbed for new game-play and computer learning techniques, because regular Shogi is too time-consuming and computationally expensive. Obata *et al.* utilized Mini-Shogi to test the effectiveness of the council algorithm for the first time [2]. This work was later applied to Shogi and chess and was developed into various consultation algorithms [3–5]. Morioka *et al.* proposed the PGLeaf learning algorithm and verified its effectiveness in Mini-Shogi games [6].

2.3 Imbalanced Winning Percentage Between Black and White

Because there are few move choices at the beginning of a game, and the initial actions of both players tend to be identical, it has been suggested that Mini-Shogi is prone to deadlock. In many tournaments, such as the UEC-cup, a rule has been adopted to force Black to break. Furthermore, White wins if the same position repeats four times during the game (*i.e.*, sennichite). However perpetual check is exceptionally illegal, and the checking player loses the game. It has been suggested that this rule might increase White's win rate, but clear evidence has been lacking, even between top-level programs until 2019. However, at the 12th UEC-cup in

[1] one Black (the first player) and one White (the second player).

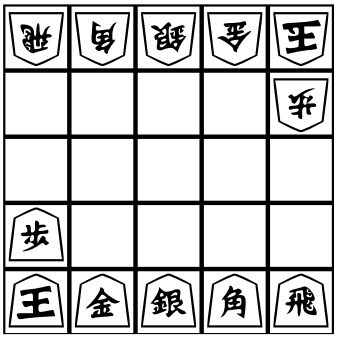


Fig. 1. Initial position of Mini-Shogi.

2020, we discovered a trend toward a large White advantage in the predecessor program to this paper and the runner-up program, Fairy-Stockfish. This finding may be due to the fact that the performance of each engine has far surpassed previous levels. Hence, we are seeking new insights into this phenomenon.

3. RELATED WORKS

Shogi is a popular game in Japan, and many computer shogi tournaments have been held. It is difficult to tune a large number of parameters by hand. Thus, tuning parameters of evaluation functions has been an important issue for a long time. These techniques for computer Shogi are described in detail in [7]. This section summarizes attempts at machine learning in computer Shogi.

3.1 Learning Method by Bonanza

Bonanza, developed by Kunihito Hoki, won the 16th World Computer Shogi Championship in 2006. Hoki proposed a method of enabling positional learning evaluation, which provided an automated learning function for the first time for computer Shogi [8]. Let $\mathcal{P}$ be a set of positions, and let $\boldsymbol{w}$ be the feature weight vector. The purpose of learning is to find a better $\boldsymbol{w}$ so that the minimax search matches well with the desired moves for each position in $\mathcal{P}$. This vague goal can be replaced with an optimization problem by the objective function:

$$J(\mathcal{P}, \boldsymbol{w}) = \sum_{p \in \mathcal{P}} \sum_{m \in M'_p} T\left(s(p.m, \boldsymbol{w}) - s(p.d_p, \boldsymbol{w})\right), \tag{1}$$

where $p.m$ is the position after move m in position p; d_p is the desired move in position p; M'_p is the set of legal moves in p excluding d_p; $s(p, \boldsymbol{w})$ is the minimax value for p; and $T(x) = 1/(1 + \exp(-ax))$, which is a sigmoid function. Hoki succeeded in adjusting more than 10,000 weights using about 60,000 game records from professional players and an online Shogi site (*i.e.*, Shogi Club 24). Later, this method was extended to the Minimax Tree Optimization [9].

Since then, the evaluation function has commonly been used with tens of millions of parameters in computer Shogi. Kanazawa suggested that the evaluation

function could make value judgment mistakes with positional evaluations, owing to the lack of training data [10]. To address this issue, Kanazawa proposed a method similar to reinforcement learning. Supposing that the minimax value under a fixed depth is the desired value for a given position, the feature weights of the evaluation function are adjusted so that the result of the quiescence search, which is an algorithm to extend positional evaluation until a stable node is reached, approximates the desired value. After applying the method several times, the strength of Bonanza 6.0 improved by about 150 Elo points.

3.2 Learning by Self-Play

This section illustrates a widely used method of computer Shogi. The aim of learning is similar to the method proposed by Kanazawa [10]. Let $\boldsymbol{w}$ and $\boldsymbol{w'}$ be the feature weight vector for self-play and learning, respectively. The first step of learning is to generate a set of training positions by self-play with $\boldsymbol{w}$. Self-play games are mainly conducted under a depth-limited tree search to save time. Moreover, it is necessary to devise a way to create training positions so that the learning does not fall into the local minima. For example, a random move will occasionally be added. The second step includes adjusting weights $\boldsymbol{w'}$ using the training positions. We have the minimax value $\xi(p, \boldsymbol{w})$, gained by self-play and the result of the quiescence search $\xi'(p, \boldsymbol{w'})$, where p is a game position. These evaluation values are converted by a function f, that is both single-valued and monotonically increasing in the range $[0, 1]$, like a sigmoid function. Let us denote $x = f\left(\xi(p, \boldsymbol{w})\right)$ and $y = f\left(\xi'(p, \boldsymbol{w'})\right)$ for simplicity. The goal of learning is to bring y close to x in each position; the objective function is

$$H(x, y) = -x \log y - (1 - x) \log(1 - y). \tag{2}$$

Takizawa showed that the method using the combination of learning variables from the evaluation value (as mentioned) and those from the information about whether the player moved and won corresponded well [11]. A modified objective function is thus

$$H(m, y) = -m \log y - (1 - m) \log(1 - y), \tag{3}$$

$$m = (1 - \lambda)t + \lambda x, \tag{4}$$

where λ is a constant, $0 \leq \lambda \leq 1$, and $t \in \{0, 0.5, 1\}$ is a variable reflecting whether a player won from position p in a given record: 1 for a win, 0.5 for a draw, or 0 for a loss.

4. DEVELOPMENT

YaneuraOu[2] was developed by Motohiro Isozaki and is one of the strongest open-source Shogi engines. It was the winning program of the 29th World Computer Shogi Championship in 2019. We modified the following two items and implemented a Mini-Shogi program based on YaneuraOu version 5.31.

[2] `https://github.com/yaneurao/YaneuraOu`, accessed on 01/31/2021.

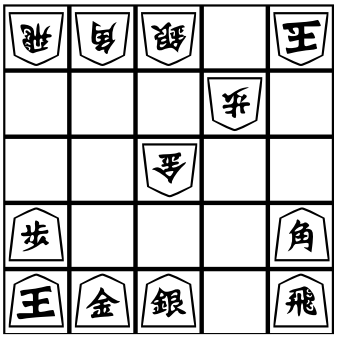


Fig. 2. Example position that *mate1ply* misjudges: Black's turn. White will be checkmated by a discovered check after `+1432KA`.

1. Bitboards
   Bitboards provide a method of representing the board state using bits. Generally, one bit is assigned to each square. An advantage of bitboards is that all squares can be operated in parallel by bitwise operations (*e.g.*, AND, OR, and XOR). Bitboards speed-up searches by calculating attacks and updating bitboard statuses. A Mini-Shogi board can be represented with one 32-bit integer.

2. *Mate1ply*
   Mate1ply is a function used to solve whether the king is checkmated in one move without a search. We modified YaneuraOu's *mate1ply* for Mini-Shogi and measured the performance. It accurately judged 98% of the positions that would result in a checkmate in one move. Therefore, some positions could not be correctly judged as a checkmate; Fig. 2 shows an example position of them. It seems to be difficult to judge the positions that are checkmated by a discovered check. Besides, the false positive rate was zero. The benchmark results showed the nodes per second decreased by about 9% using *mate1ply*, compared with not using it. However, we expect that this function raises the efficiency of the search and improves the playing strength.

4.1 Search

Computer Shogi developers have recently borrowed many techniques from Stockfish, which is the strongest chess engine in the world. YaneuraOu also applies Stockfish's search to Shogi and implements an advanced alpha-beta search algorithm. In this work, we used the part of YaneuraOu's search as is.

4.2 Evaluation Function

Nasu proposed a neural network-based evaluation function, the efficiently updatable neural network (NNUE) in 2018 [13]. It was mainstream to use evaluation functions that are a linear combination of weights before then. NNUE performs as fast as conventional evaluation functions and has impacted both computer Shogi

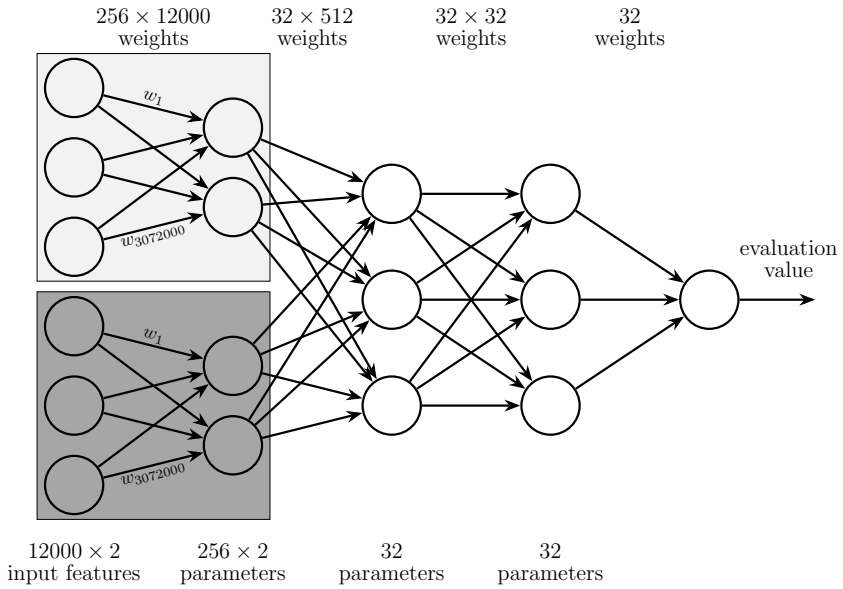


Fig. 3. NNUE structure in Mini-Shogi. Each half of the input layer is fully connected to each half of the first hidden layer. Each output of the first hidden layer is computed by the same $256 \times 12,000$ weights, but it is obtained by the incremental update during a search. The hidden layers apply a ReLU activation [12].

and computer chess. Stockfish improved more than 100 Elo points after introducing NNUE in 2020. Fig. 3 shows the network structure of Mini-Shogi. The input feature (*i.e.*, HalfKP) contains the Boolean values [13].

5. RESULTS AND DISCUSSIONS

5.1 Learning Experiments

This section shows the results of the learning evaluation function. The outline of the process of learning is described in Section 3.2. The first step is the generation of training positions using self-play. We prepared a few game records in advance and set the start position for self-play at the position after one or two random moves from the position until the 32nd move in each game record. During the second step, after removing duplicates from the training positions generated in the previous step, we tuned the weights of the evaluation function. We iterated this process 40 times and trained NNUE from scratch. All parameters of NNUE were zero at first, and the skill level of the initial state of NNUE was nearly the same as a random-move player. The learning conditions were as follows: 10 million positions as training data per epoch, a mini-batch size of 1,000, and a learning rate of 0.1. However, we varied the search depth and λ in Eq. (4), as shown in Table 1, to increase the efficiency of learning. It took about 1 day to complete the entire learning process using an AMD Ryzen 5950X processor. To measure the perfor-

Table 1. Values of search depth in self-play and λ in Eq. (4) by epoch.

Epoch	Search depth	λ
1–10	3	0.00
11–20	6	0.33
21–30	9	0.50
31–40	12	0.50

Table 2. Results of 2,000 games. Each program was given 1 s thinking time per move. "2,000 − 0" in the first row means that the 10th epoch wins all 2,000 games against 1st epoch.

Match	Result	Elo difference
10th epoch – 1st epoch	2,000 − 0	$+\infty$
20th epoch – 10th epoch	1,490 − 510	+186
30th epoch – 20th epoch	1,154 − 846	+54
40th epoch – 30th epoch	1,181 − 819	+64

mance of each evaluation function, we conducted matches over 2,000 games at time controls of 1 s per move. All matches had statistical significance, as shown in Table 2. Hence, these results suggest that there is still room to improve the accuracy, provided we conduct additional training using greater search-depths. However, the deeper the search depth of self-play, the more time-consuming generating training positions becomes.

We also investigated the effect of the search depth during self-play as it pertains to learning. Fig. 4 shows the number of learning positions versus Elo rating. Elo rating was computed from the game results against a single program, and time controls were set to 1 s per move. Overall, with greater search depth during self-play, the more improved the playing strength. Note that there were worse strengths at some points, making it necessary to review some of the learning conditions.

5.2 Comparison with Existing Programs

We participated in some tournaments in 2020, and our program won first place in all of them (*e.g.*, 12th UEC-cup Mini-Shogi Tournament and the Computer Olympiad 2020). However, it is necessary to conduct more games to evaluate whether our program was sufficiently stronger than the others. Thus, in this study, our program was matched against the second-to-fourth-place programs of the 12th UEC-cup Mini-Shogi Tournament, including Fairy-Stockfish[3], Gasyou, and Shokidoki, in 300 games with time controls of 10 min per game. Our program ran using NNUE, as trained in Section 5.1. For Fairy-Stockfish, we obtained the latest version as of January 2021 from GitHub and compiled it, whereas we used the same UEC-cup version from the tournaments for the other programs. The results are shown in Table 3. Our program was stronger than Gasyou and Shokidoki, and

[3] `https://github.com/ianfab/Fairy-Stockfish`, accessed on 01/19/2021.

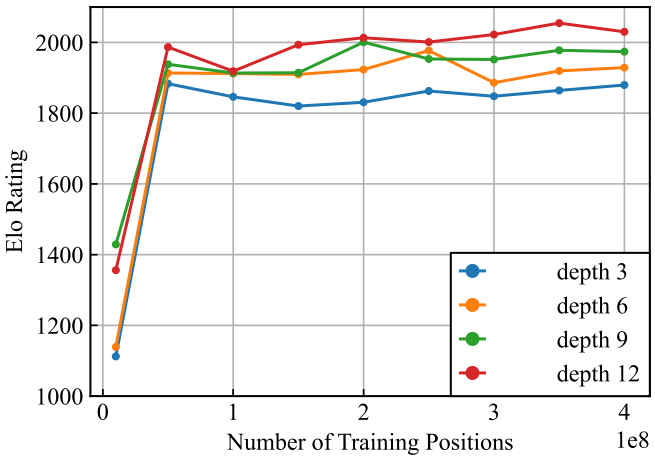


Fig. 4. Elo progression during learning. Elo rating was computed from the results against a single program, provided that its rating was 2000.

Table 3. Results against three programs in 300 games. Each program was given 10 min of thinking time per game. The number of wins and losses are shown from our program's perspective.

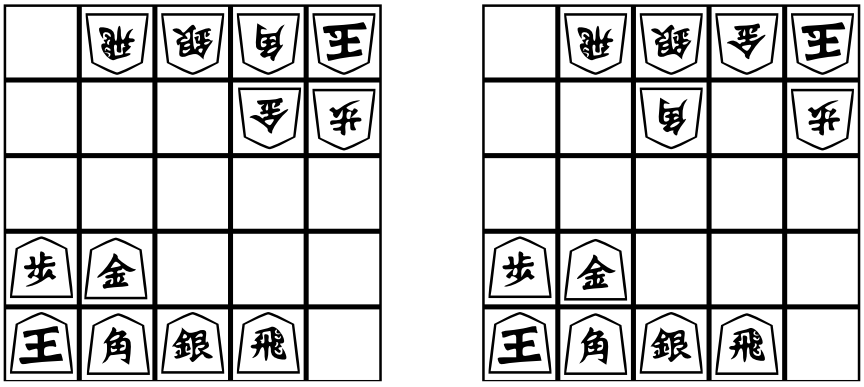
Black	White	Win	Loss
Ours	Fairy-Stockfish	19	131
Fairy-Stockfish	Ours	150	0
Ours	Gasyou	141	9
Gasyou	Ours	149	1
Ours	Shokidoki	126	24
Shokidoki	Ours	150	0

the winning percentage against them exceeded 90%, revealing a high performance for the White side. On the other hand, our program won 56% of the games against Fairy-Stockfish, including only 13% of the games on the Black side.

5.3 White's Advantage

Reviewing the results of Table 3, it is natural to propose that there is a White's advantage during gameplay. Under the condition that sennichite is treated as a draw, we trained the evaluation function by self-play as in Section 5.1 and conducted games. Unfortunately, each game ended due to draws by repetition, meaning that neither player found moves that improved their situation. Thus we posit that White has advantage, owing to the rule that White wins during sennichite.

To verify this, we used our program using the function trained in Section 5.1 and conducted 300 games. Time controls were set to 10 min per game. As a result, the winner was strongly biased toward White: White won 296 games and lost 4. Fig. 5 displays two positions after 8 moves, which was often observed. Fig. 5 (a)



(a) (b)

Fig. 5. Positions after 8 moves: Black's turn.

was observed in 183 out of 300 games, and White won 180. Moreover, 91 games reached the same position as the one shown in Fig. 5 (b). Hence, White won all of them. From Fig. 5 onwards, there were many games in which Black was completely defeated. Some of them are shown in Appendix A. We expect that it is difficult for Black to win from beginning to end, because Black must change one's move to avoid sennichite. In contrast, White is not required to do this. When our program searched for 1 billion nodes and calculated the evaluation value at each position in Figs. 5 (a) and (b) were 1,078 and 1,030, respectively. The evaluation value are represented from White's perspective with the pawn's value normalized to 100. We also performed the same experiment using Fairy-Stockfish. Consequently, White won 254 games and lost 46. In 207 out of 300 games, the position after 8 moves was identical to that shown in Fig. 5 (b). Hence, White scored 194 wins and 13 losses.

In summary, both the result using our program and those using Fairy-Stockfish suggest that White has an advantage, although each of the two programs ran different evaluation functions. This new insight supports the proposition.

6. CONCLUSION

For this work, we developed a Mini-Shogi program based on YaneuraOu, one of the strongest Shogi engines. Then, we applied the machine learning method via self-play, which has been successful in recent computer Shogi to train the evaluation functions. We experimented against leading programs for more than 300 games. As a consequence, our program performed comparatively well and showed high performance for the White side in particular. For the evaluation fuction, we adopted NNUE with a shallow neural network. In computer chess, NNUE has resulted in the improvement in the Elo rating, and our experiment also reflected its superiority. After analyzing the game records from our program and those of Fairy-Stockfish, it seems that White may have an inherent advantage

in Mini-Shogi. Although there is not so many variations in the game records, we believe the perceived advantage may be caused by a few opening moves influencing the result.

In a future work, we intend to confirm whether Mini-Shogi is really a game having a White's advantage by exploring a wider range of variations in openings. We used YaneuraOu's search with little modification in this case. However, YaneuraOu carries out a large amount of pruning because the average branching factor in Shogi is around 80. Mini-Shogi has a much smaller branching factor, so we intuitively expect that it will be better to reduce the amount of pruning. Doing so, especially with the openings, we will continue to identify the must-win positions early to build a strong opening book.

REFERENCES

1. M. Shioda and T. Ito, "Learning of evaluation functions on mini-shogi using self-playing game records," in *Proceedings of International Conference on Technologies and Applications of Artificial Intelligence*, 2020, pp. 1-6.
2. T. Obata, M. Hanawa, and T. Ito, "Consultation algorithm in brain game: Effect of simple majority system," IPSJ SIG Technical Report, No. 2009-GI-22, Vol. 2009, 2009, pp. 1-5.
3. T. Sugiyama, T. Obata, K. Hoki, and T. Ito, "Optimistic selection rule better than majority voting system," in *Proceedings of International Conference on Computers and Games*, 2010, pp. 166-175.
4. T. Obata, T. Sugiyama, K. Hoki, and T. Ito, "Consultation algorithm for computer shogi: Move decisions by majority," in *Proceedings of International Conference on Computers and Games*, 2010, pp. 156-165.
5. K. Hoki, S. Omori, and T. Ito, "Analysis of performance of consultation methods in computer chess," *Journal of Information Science and Engineering*, Vol. 30, 2014, pp. 701-712.
6. Y. Morioka and H. Igarashi, "Reinforcement learning algorithm that combines policy gradient method with alpha-beta search," in *Proceedings of the 17th Game Programming Workshop*, 2012, pp. 1-5.
7. T. Kaneko and T. Takizawa, "Computer shogi tournaments and techniques," *IEEE Transactions on Games*, Vol. 11, 2019, pp. 267-274.
8. K. Hoki, "Optimal control of minimax search results to learn positional evaluation," in *Proceedings of the 11th Game Programming Workshop*, Vol. 2006, 2006, pp. 78-83.
9. K. Hoki and T. Kaneko, "Large-scale optimization for evaluation functions with minimax search," *Journal of Artificial Intelligence Research*, Vol. 49, 2014, pp. 527-568.
10. Y. Kanazawa, "Refinement of machine learning results generated from insufficient sample data," *Journal of IPSJ*, Vol. 57, 2016, pp. 2382-2391.
11. M. Takizawa and T. Ito, "Computer shogi continues evolving: 2. development and technology on "elmo" – from winner programmer interview of the 27th

world computer shogi championship," *IPSJ Magazine*, Vol. 59, 2018, pp. 153-156.
12. "Stockfish NNUE - chessprogramming wiki," https://www.chessprogramming.org/Stockfish_NNUE, 2021.
13. Y. Nasu, "Efficiently updatable neural-network-based evaluation functions for computer shogi," https://github.com/ynasu87/nnue/blob/master/docs/nnue.pdf, 2018.

A. GAME RECORDS

Table 4. Five games between our programs. Each program was given 10 minutes of thinking time per game.

Game 1
+2534KA -4132KA +4544KI -2122KI +1525HI -5141HI +3445KA -2221KI
+4433KI -3122GI +3322KI -2122KI +0034GI -0021KI +3544GI -2131KI
+5453FU -3214KA +2524HI -1432KA +4433GI -2233KI +3433GI -0013GI
+2425HI -1322GI +3344GI -3221KA +2524HI -2132KA +0035KI -2213GI
+2425HI -1322GI +2515HI -3221KA +3534KI -2132KA +1525HI -3214KA
+2535HI -1432KA +3515HI -3221KA +3435KI -3132KI +1525HI -4144HI
+3544KI -0043GI +0041HI -4344GI +4144RY -0043KI +0035GI -4344KI
+3544GI -0041HI +0035KI -4144HI +3544KI -0043GI +0041HI -4344GI
+4144RY -0043KI +0035GI -4344KI +3544GI -0042HI +0034KI -4244HI
+3444KI -0043GI +0041HI -4344GI +5544OU -3233KI +4435OU -0043KI
+0055GI -4353KI +4142RY -0034FU +4534KA -3334KI +3534OU -0043KA
+3435OU -4325UM +3525OU -5343KI +4222RY -1122OU +0035KI -0033HI
+0011GI -2211OU +0022KA -1122OU +5544GI -0023HI +0024FU -0034GI
+3534KI -4334KI +2514OU -3424KI
Game 2
+4544KI -4132KA +2534KA -2122KI +1525HI -5141HI +3445KA -3221KA
+4434KI -2232KI +5453FU -3122GI +4554KA -4142HI +2524HI -3231KI
+5421UM -3121KI +0043KA -2131KI +3544GI -4243HI +4443GI -0015KA
+2425HI -1533UM +3433KI -2233GI +0035KA -0022KA +2522HI -3322GI
+0044KA -0025HI +4334GI -2535RY +4435KA -0015KA +0025HI -1542UM
+0052HI -4252UM +5352FU -0032HI +0043KA -3234HI +4334KA -0033KI
+3553KA -3121KI +3445KA -0034GI +0041HI -3425GI +4554KA -0032HI
+4121RY -1121OU +0042KI -0034HI +4232KI -3332KI +0051HI -0031KI
+5545OU -3454HI +4554OU -0043KA +5455OU -2534NG +0041HI -1213FU
+5331UM -2231GI +0011KI -2122OU +4143RY -3243KI +5131RY -2231OU
+0032GI -3132OU +0021KA -3233OU +2143UM -3343OU +0054KI -4352OU
+5443KI -5243OU +1112KI -0044KI

Game 3

+2534KA -4132KA +4544KI -2122KI +3445KA -3221KA +1525HI -5141HI
+4434KI -2232KI +5453FU -3122GI +4554KA -4142HI +2524HI -3231KI
+5421UM -3121KI +0043KA -2131KI +3544GI -4243HI +4443GI -0015KA
+2425HI -1533UM +3433KI -2233GI +0035KA -0022KA +2522HI -3322GI
+0044KA -0025HI +4334GI -2535RY +4435KA -0015KA +0045HI -0024KI
+3524KA -1524UM +0025KI -2451UM +2535KI -0033KA +5554OU -3315UM
+5455OU -3142KI +3544KI -1524UM +0035HI -2233GI +3433GI -4233KI
+3533HI -5133UM +0031GI -3344UM +4544HI -0025HI +0045KI -0033GI
+4424HI -2524RY +0022KA -3322GI +3122GI -1122OU +0044KA -2232OU
+0033GI -2433RY +4433KA -0054KI +5554OU -0043GI +5455OU -0054HI
+4554KI -4354GI +5554OU -0043GI +5445OU -0054KA +4555OU -0045KI

Game 4

+2534KA -4132KA +4544KI -2122KI +1525HI -5141HI +3445KA -2221KI
+4433KI -3122GI +3322KI -2122KI +0044GI -0021KI +3534GI -2131KI
+4433GI -2233KI +3433GI -0022GI +3344GI -3221KA +0024KI -3132KI
+2434KI -3231KI +5453FU -2132KA +5352FU -3214KA +2535HI -3142KI
+3515HI -1432KA +1525HI -4121HI +4554KA -3254KA +5554OU -4252KI
+0043KA -5243KI +4443GI -0014KA +2524HI -0025KA +0042KI -0041FU
+4232KI -1432KA +4332GI -2534UM +3221NG -1121OU +2434HI -0053KI
+5453OU -0042KI +5344OU -0043GI +4445OU -4334GI +4534OU -4233KI
+3445OU -0043HI +4535OU -3344KI +3524OU -2223GI +2415OU -4434KI
+0035KA -3435KI +0011HI -2111OU +0055KA -0044KA +5544KA -4344HI
+0033KA -0022KA +3322KA -1121OU +2211UM -2111OU +0022KI -1122OU
+0011KA -2211OU +0025GI -3525KI +1525OU -0024KI

Game 5

+4544KI -4132KA +2534KA -2122KI +3445KA -5141HI +1525HI -2221KI
+2515HI -2122KI +4434KI -3243KA +3444KI -3142GI +3524GI -4332KA
+1535HI -3221KA +4534KA -2132KA +3515HI -4151HI +1525HI -3214KA
+2515HI -1432KA +3445KA -5141HI +4434KI -3221KA +2423GI -4233GI
+2322GI -3322GI +1525HI -0033GI +3435KI -2143KA +2515HI -4321KA
+1514HI -1213FU +1415HI -2112KA +4512KA -1112OU +0045KA -0023KA
+5453FU -2345UM +3545KI -0023KA +4535KI -1314FU +1525HI -2334KA
+3534KI -3334GI +2522HI -1222OU +0044KA -2212OU +0023GI -3423GI
+0022KI -1213OU +2223KI -1323OU +0045KA -2313OU +4435KA -0024GI
+3524KA -1324OU +0033GI -2433OU +0034GI -3342OU +5544OU -0022KA
+4435OU -0044GI +3524OU -0013KI +2425OU -0015HI

Masahiro Shioda received his bachelor's degree from the University of Electro-Communications in 2020 and is currently a master's student at the same university. His research interests include artificial intelligence and computer games.

Takeshi Ito has been working as an Assistant Professor at the University of Electro-Communications since 1994, and an Associate Professor since 2018. He received his Ph.D. from Nagoya University graduated schools in Aichi, Japan in 1994. His research interests include human cognitive processes and learning processes in playing thinking games or solving difficult problems.