Evaluation of Multi-staging and Weight Promotion for Game 2048

Evaluation of Multi-staging and Weight Promotion for Game 2048

Kiminori Matsuzaki*

School of Information, Kochi University of Technology
185 Miyanokuchi, Tosayamada, Kami City, Kochi 782–8502, JAPAN

* E-mail: matsuzaki.kiminori@kochi-tech.ac.jp

Abstract: Game 2048 is a stochastic single-player game categorized in slide-and-merge games. Several computer players have been developed for 2048, and among them, strong computer players uses N-tuple networks as evaluation functions that are adjusted by reinforcement learning methods. To enhance those computer players, multi-staging and weight-promotion techniques have been proposed. However, no paper has directly compared the effects of those techniques. In this paper, we conduct experiments exhaustively in terms of multi-staging and weight promotion including newly proposed ways to evaluate their effects. From the results, the most effective multi-staging implementation differs for the goals, that are, maximizing the average score and maximizing the maximum score. The computer player developed with the multi-staging implementation that used the maximum number of the tiles as the index and two weight-promotion implementations achieved average score 225,078 with the greedy plays and average score 453,776 with the 3-ply expectimax plays.

1. Introduction

The single-player stochastic game 2048[4], a derivative of the games Threes and 1024, is a very popular one among similar slide-and-merge games. According to the author, during the first three weeks after its release, people spent a total time of over 3000 years on playing the game. One of the reasons why the game attracts so many people is that it is very easy to learn but hard to master. The game also attracts researchers in the field of computer science and artificial intelligence. Several people have devoted themselves to the development of strong computer players[3,5–16].

A big trend on the computer players for 2048[7–10,12,13,15] uses N-tuple networks as approximated evaluation functions, which are adjusted by reinforcement learning methods such as temporal difference learning. Among those, the state-of-the-art computer player developed by Jaśkowski[7] achieved average score 609,104 under the timelimit of 1 s/move. The player was developed based on the temporal coherence learning method enhanced with four techniques, multi-staging, weight promotion, redundant encoding, and carousel shaping.

In my previous work[9], I showed that we can improve the learning of the evaluation functions with two techniques: *backward* temporal coherence learning and *restart*. In this paper, we further investigate the performance of N-tuple-network-based computer players focusing on the multi-staging and weight-promotion

techniques.

The multi-staging technique is to divide a game into multiple stages and use different evaluation functions for each of the stages. This technique was used in a master-level Othello playing program Logistello[2]. The multi-staging technique was also successfully applied for 2048 by Wu et al.[13], Yeh et al. [15] and Jaśkowski[7], while they used different implementation of multiple stages.

Application of the multi-staging technique would slow down the learning process especially for the later stages. This problem could be resolved with the weight-promotion technique that initialize weights with some existing learned weights. Jaśkowski[7] proposed an inter-stage weight-promotion implementation, which significantly improved the performance of the players.

The contributions in the paper are summarized as follows.

- **Multi-staging Technique**: I design five ways of dividing a game into stages including two existing ones[7,13,15].
- **Weight-promotion Technique**: In addition to the inter-stage weight-promotion implementation proposed by Jaśkowski[7], I design an intra-stage weight-promotion implementation.
- **Exhaustive Experiments**: I conducted a set of experiments exhaustively with the computer players developed with these multi-staging and weight-promotion techniques. We had several interesting findings from the results of these experiments.

Here I would like to enumerate interesting and important findings in this paper.

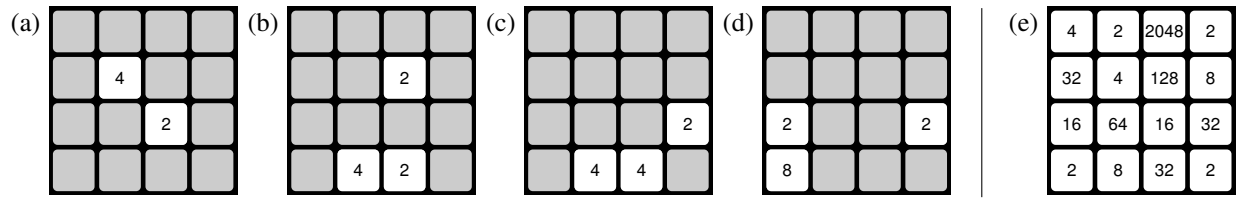
1. The most effective multi-staging implementation differs for the goal. For maximizing the average score, the multi-staging implementation that uses the maximum number of the tiles as index performed the best. For maximizing the maximum score, the multi-staging implementations that divide the game into almost the same size performed the best.
2. The most effective weight-promotion implementation differs for the multi-staging implementation combined. Generally speaking, use of both the inter-stage and intra-stage weight-promotion performed the best. However, for Jaśkowski's multi-staging implementation that divides the game into the same size[7], only the inter-stage weight-promotion performed well. The inter-stage weight-promotion implementation performed better than the intra-stage weight-promotion implementation when the size of stages were almost the same.

The computer player developed with the eight-stage implementation that uses the maximum number of the tiles as index and both the weight-promotion implementations achieved average score 225,078 with greedy plays and average score 453,776 with 3-ply expectimax plays. The computer player developed with the eight-stage implementation that divides the game based on the existence of 8192-, 16384-, and 32768-tiles and the inter-stage weight-promotion implementation achieved maximum score 802,050 with greedy plays and maximum score 828,361 with 3-ply expectimax plays.

The rest of the paper is organized as follows. Section 2 briefly introduces the rule of the game 2048. Section 3 reviews the idea of applying N-tuple networks and reinforcement learning methods to the game 2048. We will use the backward temporal-coherence learning with restart developed in my previous work[9]. Section 4 shows the multi-staging implementations and the weight-promotion implementations including newly proposed ones. Section 5 reports the experiments and discusses our findings from the experiment results. Finally, Section 6 concludes the paper.

2. Rules of Game 2048

Game 2048 is played on a 4×4 grid. The objective of the original game is to reach a 2048-tile by moving and merging the tiles on the board according to the rules described below.



(a) An example of the initial state. Two tiles are put randomly.

(b) After the first move: *down*. Two tiles move to the bottom and a new 2-tile appears at cell (3,2).

(c) After the second move: *down*. Two 2-tiles are merged to a 4-tile, and score 4 is given. A new tile appears at cell (4,3).

(d) After the third move: *left*. Two 4-tiles are merged to a 8-tile, and score 8 is given. A new tile appears at cell (4,3).

(e) An example of the final state where the player cannot move nor merge in any direction.

Figure 1. The process of game 2048

In the initial state (Figure 1 (a)), two tiles are put randomly with numbers 2 ($p_2 = 0.9$) or 4 ($p_4 = 0.1$). The player selects a direction (either of up, left, down, and right), and then all the tiles will move in that direction (Figure 1 (b)–(d)). When two tiles of the same number collide they create a tile with the sum value and the player get the sum as the score. Here, the merges occur from the far side and a newly created tile do not merge again on the same move: moves to the right from `222␣`, `␣422` and `2222` result in `␣␣24`, `␣␣44`, and `␣␣44`, respectively. Note that the player cannot select a direction in which no tiles move nor merge. After each move, a new tile appears randomly at an empty cell with number 2 ($p_2 = 0.9$) or 4 ($p_4 = 0.1$).

If the player cannot move the tiles, the game ends (Figure 1 (e)).

3. N-tuple Networks and Reinforcement Learning for 2048

In this section, we first review the idea of applying N-tuple networks and reinforcement learning methods to the game 2048. We then introduce the learning method used in the paper, that is, backward temporal coherence learning with restart[9]. Note that the original method of using N-tuple networks for 2048 was first given by Szubert and Jaśkowski[12] (they showed three methods and the method used in this paper was called TD-AFTERSTATE), and the temporal coherence learning was first introduced for 2048 by Jaśkowski[7].

3.1 Evaluation Function with N-tuple Networks

N-tuple networks consist of a number of N-tuples associated with tables of feature weights (Figure 2 (a)). Let N be the number of cells that an N-tuple covers and K be the number of possible values, then the number of feature weights (the number of rows) in a table is K^N. In this study, we use four 6-tuples shown in Figure 3 and limit the number of possible values $K = 16$ (let the maximum number of tiles be 32768): the number of feature weights is $4 \times 16^6 = 67{,}108{,}864$.

Given a set of N-tuples and corresponding tables of feature weights, we calculate the evaluation value of a state as follows. Since the board of the game 2048 has rotational and reflectional symmetries, we can consider eight symmetric boards and look up feature weights for each of them. The evaluation value is the sum of the feature weights. See an example in Figure 2 with two 3-tuples. We have eight symmetric boards for a state, and look up two feature weights for each board (in Figure 2 (c), the first two are from the upper-left board and the last two from the lower-right board). Therefore, in this example, the evaluation value of a state is the sum of 16 feature weights.

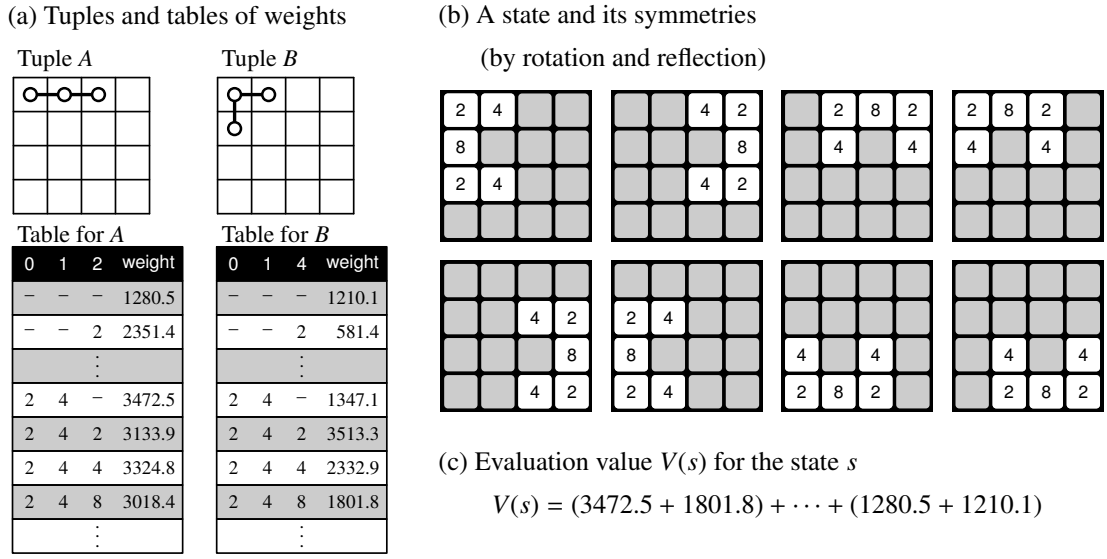


Figure 2. An example for calculating an evaluation value of a state[9)]

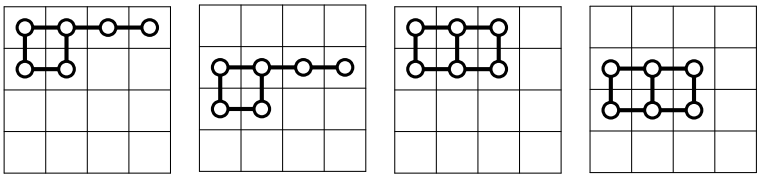


Figure 3. N-tuple networks used in the paper: four 6-tuples designed by Wu et al.[13)]

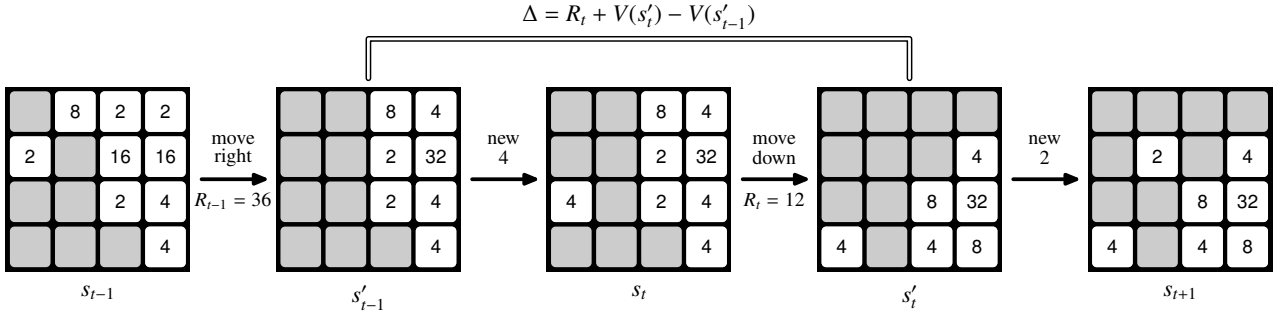


Figure 4. Transition of states

Hereafter, we will use the following notation. A state is denoted by s, which is often associated with time t as s_t. The feature weight for a specific tuple is denoted by $V_i[s]$, and the evaluation value is $V[s]$. The number of N-tuples is denoted by m, where it is multiplied by eight (due to symmetric sampling).

3.2 Temporal Coherence Learning

After designing N-tuple networks, we apply reinforce learning methods to adjust the feature weights. In reinforcement learning methods, we update feature weights so that certain errors (often called TD-errors) are minimized. In this paper, I uses the error first formalized by Szubert and Jaśkowski[12)] (the method was called TD-AFTERSTATE in their paper).

See Figure 4. Let s_{t-1} be a state at time $t-1$. After a player selects a move, all the tiles are moved and merged: we obtain reward R_{t-1} and state s'_{t-1} after the move. Then a new tile appears randomly: this is the state s_t at time t. Similarly, after a player selects a move at s_t, we have state s'_t (before the appearance of a new

Algorithm 1 Online (Forward) Temporal Coherence Learning TC(λ) [7,9]

1: **function** LearnFromSelfplay()
2: $t \leftarrow 0$; $s_0 \leftarrow$ InitialState()
3: **while not** Terminal(s_t) **do**
4: $a_t \leftarrow \mathrm{argmax}_{a \in A}$Evaluate($s_t, a$)
5: $(R_t, s'_t, s_{t+1}) \leftarrow$ MakeAction(s_t, a_t)
6: **if** $t > 0$ **then** TC-Update($t-1, R_t + V(s'_t) - V(s'_{t-1})$)
7: $t \leftarrow t + 1$
8: TD-Update($t-1, -V(s'_{t-1})$)
9: **function** Evaluate(s, a)
10: $(R, s', s'') \leftarrow$ MakeAction(s, a)
11: **return** $R + V(s')$
12: **function** TC-Update(t, Δ)
13: $k \leftarrow 0$
14: **while** $t - k \geq 0$ **and** $\lambda^k \Delta > \epsilon$ **do**
15: **for** $i = 1$ **to** m **do**
16: $\alpha =$ **if** $A_i[s'_{t-k}] = 0$ **then** 1 **else** $|E_i[s'_{t-k}]|/A_i[s'_{t-k}]$
17: $\delta = \alpha \lambda^k \Delta / m$
18: $V_i[s'_{t-k}] \leftarrow V_i[s'_{t-k}] + \delta$
19: $E_i[s'_{t-k}] \leftarrow E_i[s'_{t-k}] + \delta$
20: $A_i[s'_{t-k}] \leftarrow A_i[s'_{t-k}] + |\delta|$
21: $k \leftarrow k + 1$

* Function InitialState generates an initial state. Function Terminal judges that given state reaches the end of the game. Function MakeAction takes a state and a move and return the reward, the state after the move, and the state after the appearance of a new title.

tile). Here, we define error Δ as

$$\Delta = R_t + V(s'_t) - V(s'_{t-1}) .$$

In the basic temporal difference algorithm (the online TD(0) algorithm), we update the feature weights with this error Δ as

$$V'(s'_{t-1}) \leftarrow V(s'_{t-1}) + \alpha\Delta$$

during the selfplays of the game. Parameter α (called *learning rate*) controls the speed of learning.

Temporal coherence learning (TC learning for short)[1] is an adaptive (learning-rate free) algorithm. TC learning was first introduced for the game 2048 by Jaśkowski[7]. Algorithm 1 shows the online version of the TC learning algorithm. In TC learning, we augment each feature weight $V_i[s]$ with two values: the accumulation of errors $E_i[s]$, and the accumulation of absolute errors $A_i[s]$. The learning rate is determined by the ratio $|E_i[s]|/A_i[s]$ (line 16): it is 1 in the beginning and then decreases if the error switches positive and negative values. Note that TC learning requires three times as much memory as the temporal difference learning does[1].

[1] In the implementation, $V_i[s]$ is represented by a 32 bit fix-point number (with 10 bits below the point) and $E_i[s]$ and $A_i[s]$ by 32 bit floating-point numbers.

Algorithm 2 Backward TC Learning[9)]

```
1: function LearnFromSelfplay()
2:     t ← 0; s ← InitialState()
3:     while not Terminal(s_t) do
4:         a_t ← argmax_{a∈A} Evaluate(s_t, a)
5:         (R_t, s'_t, s_{t+1}) ← MakeAction(s_t, a_t)
6:         t ← t + 1
7:     BackwardLearning(1, t)
8: function BackwardLearning(t_0, t)
9:     TC-Update(t − 1, −V(s'_{t−1}))
10:    for τ = t − 1 downto t_0
11:        TC-Update(τ − 1, R_τ + V(s'_τ) − V(s'_{τ−1}))
```

3.3 Backward Temporal Coherence Learning with Restart

The game 2048 has the following unique properties, i.e., difficulties of learning. (1) Long sequence of moves: A strong player would play tens of thousands of moves until the game end. (2) Larger reward at later part: A large reward would be obtained at a later part of the game when the player made a large-numbered tile, e.g., 32768. (3) Increasing difficulty: The game becomes more difficult toward the end of the game.

In my previous work[9)], I applied two techniques to improve the temporal coherence learning method.

Backward learning This technique is to update the feature weights from the end of the game to speed up the propagation of rewards. This would resolve the first two difficulties.

Restart This technique is to learn more actions in the later part of the game. This would resolve the third difficulty.

Considering the first two properties, the long sequence of moves and larger reward at later part, I expected that the online (forward) learning algorithms would not work well. If a large reward was given near the end of the game, we would need to update the feature weights more than ten-thousand times to reflect the reward to the beginning of the game. A solution to this problem is to perform the updates in a backward direction from the end of the game. Algorithm 2 shows the modification for the backward TC learning algorithm where the other functions are the same as Algorithm 1. In the backward TC learning, we use $\lambda = 0$.

The last property, increasing difficulty, implies that it is important to learn more actions in the later part. Jaśkowski proposed a technique called *carousel shaping*[7)] to improve the learning. In this paper, we use a simple restart strategy based on the history of states (Algorithm 3)[9)]: if the player ends the game, it recursively restarts the game at the middle of the history to learn the latter half. By this restart strategy, the player will learn more actions near the end of the game. The algorithm has additional parameter L: if the length of a restarted game is shorter than L, we start the next game from an initial state. We used parameter $L = 100$ in this paper.

4. Multi-staging and Promotion

4.1 Multi-staging

Multi-staging is a technique to divide a game into multiple stages and use different evaluation functions for each of the stages. This technique was used in a master-level Othello playing program Logistello[2)].

Multi-staging was also successfully applied for 2048[7,9,13,15)], but different implementations of stages were used in existing work. In the first implementation by Wu et al.[13)], a game is divided into three stages where

Algorithm 3 Learning with Restart[9)]

```
1: function LearnFromSelfplayWithRestart()
2:     t_start ← 1; t_end ← ∞; s_0 ← InitialState()
3:     while t_end − t_start > L do
4:         t ← t_start
5:         while not Terminal(s_t) do
6:             a_t ← argmax_{a∈A} Evaluate(s_t, a)
7:             (R_t, s'_t, s_{t+1}) ← MakeAction(s_t, a_t)
8:             t ← t + 1
9:         BackwardLearning(t_start, t)
10:        t_end ← t; t_start ← (t_start + t_end)/2
```

a later stage has shorter length. Yeh et al.[15)] extended the implementation up to six stages in the same manner. Jaśkowski[7)] used different implementation where all the stages have exactly same length. In my previous work[9)], the stages were simply defined by the maximum number of tiles. It remains unclear which implementation of the stages performs the best.

In this paper, we borrow the notation by Yeh et al.[15)] for dividing a game into stages. T_{512} denotes the first time when a 512-tile is created. Similarly, T_{1k}, T_{2k}, T_{4k}, T_{8k}, T_{16k} and T_{32k} denote the first time when 1024-, 2048-, 4096-, 8192-, 16384-, and 32768-tile are created, respectively. $T_{32k+16k}$ denotes the first time when both 32768-tile and 16384-tile are created on a board. Similarly, $T_{32k+16k+8k}$ denotes the first time when all the 32768-tile, 16384-tile, and 8192-tile are created on a board, and so on.

The baseline **ST-1** has a single stage. I use the following nine multi-stage implementations.

ST-TURN4, ST-TURN8 These implementations divide a game by the fixed number of moves. The idea was used in the implementation by Jaśkowski [7)]. In the four-stage case, a game is divided at every 8192 moves: 8192, 16384, and 24576. In the eight-stage case, a game is divided at every 4096 moves: 4096, 8192, 12288, 16384, 20480, 24576 and 28672.

ST-TOP4, ST-TOP8 These implementations divide a game into similar numbers of moves, based on the creation of large-numbered tiles: 16384-tile and 32768-tile in the four-stage case; 8192-tile, 16384-tile and 32768-tile in the eight-stage case. In the four-stage case, a game is divided at T_{16k}, T_{32k} and $T_{32k+16k}$. In the eight-stage case, a game is divided at T_{8k}, T_{16k}, T_{16k+8k}, T_{32k}, T_{32k+8k}, $T_{32k+16k}$ and $T_{32k+16k+8k}$.

ST-MAX4, ST-MAX8 These implementations simply uses the maximum number of the tiles as the index of stages. In the four-stage case, a game is divided at T_{8k}, T_{16k} and T_{32k}. In the eight-stage case, a game is divided at T_{512}, T_{1k}, T_{2k}, T_{4k}, T_{8k}, T_{16k} and T_{32k}. Note that the later stage is longer in these implementations.

ST-DECR4, ST-DECR8 These implementations divide a game into stages so that the later stage is shorter. The idea [2] was used in the implementation by Wu et al.[13)] and Yeh et al.[15)]. In the four-stage case, a game is divided at T_{32k}, $T_{32k+16k}$ and $T_{32k+16k+8k}$. In the eight-stage case, a game is divided at T_{32k}, $T_{32k+16k}$, $T_{32k+16k+8k}$, $T_{32k+16k+8k+4k}$, $T_{32k+16k+8k+4k+2k}$, $T_{32k+16k+8k+4k+2k+1k}$ and $T_{32k+16k+8k+4k+2k+1k+512}$.

ST-HAND8 This is a hand-designed implementation based on the following two ideas. Firstly, we keep the length of the shortest stage not too short. Secondly, we have more stages for difficult parts of the game: before creating a 32768-tile and before creating a 65535-tile. A game is divided at T_{16k}, T_{16k+8k}, $T_{16k+8k+4k}$, T_{32k}, $T_{32k+16k}$, $T_{32k+16k+8k}$, and $T_{32k+16k+8k+4k}$.

[2] In the original implementation, a game was divided at T_{16k} and T_{16k+8k}.

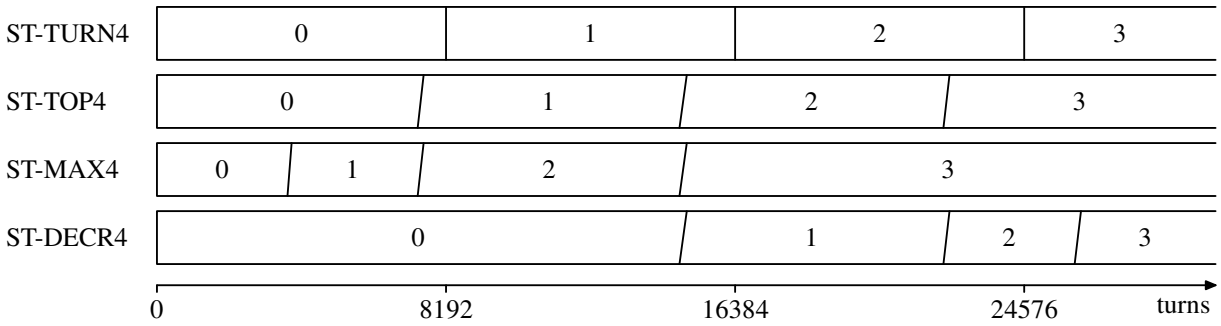


Figure 5. The length of stages for four-stage cases. Note that boundary of stages may change slightly in ST-TOP4, ST-MAX4 and ST-DECR4 (denoted by transverse lines).

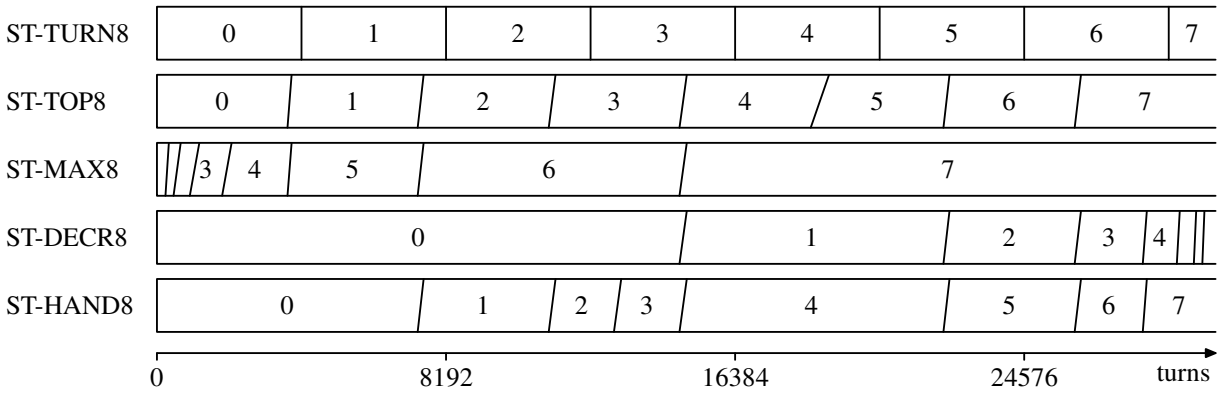


Figure 6. The length of stages for eight-stage cases. Note that boundary of stages may change slightly in ST-TOP8, ST-MAX8, ST-DECR8 and ST-DECR8 (denoted by transverse lines).

In the four-stage cases, the tables of feature weights occupy $4 \times 4 \times 16^6 \times (4\,\text{bytes} + 4\,\text{bytes} + 4\,\text{bytes}) = 3\,\text{GB}$. In the eight-stage cases, the tables occupy 6 GB. Figures 5 and 6 depict the stages of the four-stage implementations and eight-stage implementations, respectively.

4.2 Weight Promotion

Use of multi-staging techniques considerably increases the number of feature weights and it slow down the learning process. To resolve this problem, Jaśkowski[7] proposed a weight-promotion technique (we call it *inter-stage weight promotion* in this paper): Each feature weight is initialized, upon its first access, to the corresponding weight in the preceding stage. Jaśkowski reported that this weight-promotion technique improved the performance of developed players.

Since the number of feature weights is quite large, I propose another weight-promotion technique, namely, *intra-stage weight promotion*. With the intra-stage weight promotion, each feature weight is initialized, upon its first access, to the weight of the one-step-smaller board. Here, given a board, its one-step-small board is computed by halving the number of each tile (If the original board has 2-tiles, we replace them by empty cells). Figure 7 show an example.

With these weight-promotion techniques, we have following four implementations.

PR-NO No weight-promotion technique is applied.

PR-IN Only the intra-stage weight-promotion technique is applied.

PR-OVER Only the inter-stage weight-promotion technique is applied.

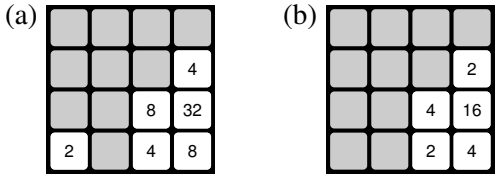


Figure 7. (a) Original Board. (b) The one-step-smaller board of (a).

PR-BOTH Both the intra-stage and the inter-stage weight-promotion techniques are applied. On the first-time access, the intra-stage weight promotion is applied first, and if it fails (in the first stage or the corresponding weight was not accessed) the intra-stage weight promotion is applied.

Note that we judge the first-time access with $V_i[s'] = 0$ to avoid additional storage for it. This implementation technique was also used in Jaśkowski's implementation[7].

5. Experiments and Discussion

We conducted experiments to evaluate all the combinations of the multi-staging and weight-promotion implementations in Section . Each implementation performed backward TC learning with restart until 4×10^{10} actions, monitoring the learning process at every 1×10^9 actions with 10000-game greedy plays and 300-game 3-ply expectimax plays (the number of learned actions followed that in Jaśkowski's paper[7]). The learning took about 10–13 hours and the monitoring took about 12 hours additionally, on a server with dual Intel Xeon E5645 CPUs (6 cores, 12MB cache, HT off) and 12 GB memory. Each implementation was executed five times.

Figures 8–17 show the learning process monitored with greedy plays. Figures 18–27 show the learning process monitored with 3-ply expectimax plays. In these figures, the averaged score of 5 runs is used. Tables 1–3 show the scores after learning 1×10^{10}, 2×10^{10} and 4×10^{10}. In these tables, each averaged score is accompanied with the 95% confidence interval. To avoid the effects of perturbation, Table 3 includes the maximum score monitored at 3.8×10^{10}, 3.9×10^{10} and 4.0×10^{10} actions.

Firstly, look at the results of the single-stage cases (Figures 8 and 18 and Table 3). The intra-stage weight promotion (PR-IN) improved the maximum score and the average score of expectimax plays by 8–10% while it degraded the average score of the greedy plays. Since the variance was large, the difference was not significant.

Secondly, look at the results of the ST-TURN4, ST-TURN8 and ST-TOP8 cases (Figure 9, 10, 12, 19, 20 and 22 and Table 3). In these cases, the inter-stage weight promotion (PR-OVER) improved both the average score and the maximum score of both greedy and expectimax plays. Compared with the inter-stage weight promotion (PR-OVER), the improvement by the intra-stage weight promotion (PR-IN) was small and sometimes negative. It is interesting that the best scores were with only inter-stage weight promotion (PR-OVER), and using both weight-promotion techniques (PR-BOTH) performed worse than PR-OVER for these cases.

From these results, we could conclude that the inter-stage weight promotion only (PR-OVER) achieved the best performance for multi-staging implementation that have stages of the same or similar sizes. An exception for this conclusion was for ST-TOP4 (Figure 11 and 21). In the case of ST-TOP4, all the player showed almost the same performance.

Thirdly, look at the results of the ST-MAX4 and ST-MAX8 cases (Figures 13, 14, 23 and 24 and Table 3). In these cases, the intra-stage weight promotion (PR-IN) improved more than the inter-stage weight promotion (PR-OVER), and the best performance was achieved by using both of these weight-promotion techniques (PR-BOTH). This also partially applied to the ST-DECR4 and ST-DECR8 cases (Figures 15, 16, 25 and 26 and Table 3). (The graphs had different shapes from other cases: the learning seemed not to proceed during

0.5×10^{10}–1.5×10^{10} actions and did slowly after that.)

From these results, we could conclude that use of both the inter-stage and intra-stage weight-promotion techniques (PR-BOTH) achieved the best performance for multi-staging implementation that have stages of different sizes.

Fourthly, look at the results of the ST-HAND8 case (Figures 17 and 27 and Table 3). Unfortunately, the results of these implementations with hand-designed stages were not good as expected. An interesting fact is that the graph of expectimax plays (Figure 27) looked quite similar to that of ST-TURN4 (Figure 19).

Now we look at the average scores of the expectimax plays (Table 3) in detail. The best average scores were achieved by ST-MAX8 with PR-BOTH (453,776) and by ST-MAX4 with PR-BOTH (452,303). It was not so surprising that these best scores were almost the same, because the first four stages in ST-MAX8 were almost useless as we can see in Figure 6. The good average scores were also achieved by ST-TOP4 with PR-IN and ST-TURN8 with PR-OVER.

We then look at the maximum scores of the expectimax plays (Table 3) in detail. The maximum scores of ST-MAX4 and ST-MAX8 were not so good, while the average scores of them were the best. The best maximum scores were achieved by ST-TURN4 with PR-OVER (831,120) and ST-TOP8 with PR-OVER (828,361), followed by ST-TOP4 with PR-BOTH (829,267), ST-DECR8 with PR-BOTH (826,751), ST-HAND8 with PR-OVER (826,613). From these results, short stages just before the creation of 65536-tile seemed to be important for the maximum scores.

6. Conclusion

The multi-staging and weight-promotion techniques are considered to improve the N-tuple-based computer players for 2048. In this paper, I conducted experiments exhaustively to evaluate nine multi-staging implementations and four weight-promotion implementations. From the experiment results some interesting and important findings were obtained.

First, the most effective multi-staging implementation differs for the goal. For maximizing the average score, multi-staging implementation that uses the maximum number of the tiles as the index of stages (ST-MAX4 and ST-MAX8) performed the best. For maximizing the maximum score, multi-staging implementations that divide the game into almost the same size (ST-TURN4 and ST-TOP8) performed the best. Secondly, the most effective weight-promotion implementation differs for the multi-staging implementation combined. Generally speaking, use of both the inter-stage and intra-stage weight-promotion (PR-BOTH) performed the best. However, for the multi-staging implementations that divide the game into the same size[7], use of the inter-stage weight-promotion only (PR-OVER) performed well. The inter-stage weight-promotion technique (PR-OVER) performed better than the intra-stage weight-promotion technique (PR-IN) when the size of stages was almost the same.

We achieved the average score 225,078 with greedy plays and the average score 453,776 with 3-ply expectimax plays by the player ST-MAX8 with PR-BOTH. We achieved the maximum score 802,050 with greedy plays and the maximum score 828,361 with 3-ply expectimax plays by the player ST-TOP8 with PR-OVER. These results are better than existing results in the similar settings (N-tuple networks defined with four or five 6-tuples, combined with 3-ply expectimax plays)[7,13,15].

In this paper I used temporal coherence learning with restart as the learning algorithm. I found in the experiments that the learning algorithm also affects the resulting performance significantly. We could strengthen the discussion and the conclusions in this paper if we apply different learning algorithms, such as Jaśkowski's carousel shaping technique[7] and Wu and Yeh et al.'s staged learning technique[13,15].

Acknowledgment

All the experiments in this paper were conducted on the IACP cluster of the Kochi University of Technology. The cumulative machine usage reached more than 240 machine-days including the preliminary experiments.

References

1) D. F. Beal and M. C. Smith. Temporal coherence and prediction decay in TD learning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, volume 1, pages 564–569, 1999.
2) M. Buro. Logistello: A strong learning othello program. In *19th Annual Conference Gesellschaft für Klassifikation e.V.*, 1995.
3) T. Chabin, M. Elouafi, P. Carvalho, and A. Tonda. Using linear genetic programming to evolve a controller for the game 2048. `http://www.cs.put.poznan.pl/wjaskowski/pub/2015-GECCO-2048-Competition/Treecko.pdf`, 2015.
4) G. Cirulli. 2048. `http://gabrielecirulli.github.io/2048/`, 2014.
5) A. Dedieu and J. Amar. Deep reinforcement learning for 2048. Available online: `http://www.mit.edu/~adedieu/pdf/2048.pdf`, 2017.
6) H. Guei, T. Wei, J.-B. Huang, and I-C. Wu. An early attempt at applying deep reinforcement learning to the game 2048. In *the Workshop Neural Networks in Games in the International Conference on Computers and Games (CG 2016)*, 2016.
7) W. Jaśkowski. Mastering 2048 with delayed temporal coherence learning, multi-stage weight promotion, redundant encoding and carousel shaping. *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.
8) K. Matsuzaki. Systematic selection of N-tuple networks with consideration of interinfluence for game 2048. In *Technologies and Applications of Artificial Intelligence (TAAI 2016)*, 2016.
9) K. Matsuzaki. Developing 2048 player with backward temporal coherence learning and restart. In *Proceedings of Fifteenth International Conference on Advances in Computer Games (ACG2017)*, 2017.
10) K. Oka and K. Matsuzaki. Systematic selection of N-tuple networks for 2048. In *In Proceedings of 9th International Conference on Computers and Games (CG2016)*, 2016.
11) P. Rodgers and J. Levine. An investigation into 2048 AI strategies. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–2, 2014.
12) M. Szubert and W. Jaśkowski. Temporal difference learning of N-tuple networks for the game 2048. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8, 2014.
13) I.-C. Wu, K.-H. Yeh, C.-C. Liang, C.-C. Chang, and H. Chiang. Multi-stage temporal difference learning for 2048. In *Technologies and Applications of Artificial Intelligence*, volume 8916 of *Lecture Notes in Computer Science*, pages 366–378, 2014.
14) R. Xiao, W. Vermaelen, and P. Moraŕek. AI for the 2048 game. `https://github.com/nneonneo/2048-ai`, 2015.
15) K.H. Yeh, I.C. Wu, C.H. Hsueh, C.C. Chang, C.C. Liang, and H. Chiang. Multi-stage temporal difference learning for 2048-like games. *IEEE Transactions on Computational Intelligence and AI in Games*, 2016.
16) A. Zaky. Minimax and expectimax algorithm to solve 2048. `http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2013-2014-genap/Makalah2014/MakalahIF2211-2014-037.pdf`, 2014.

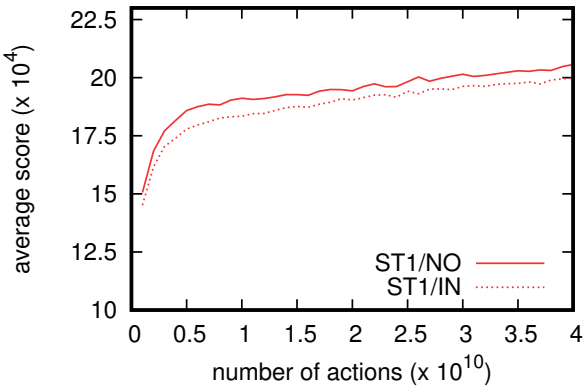


Figure 8. Average scores of greedy plays with ST-1

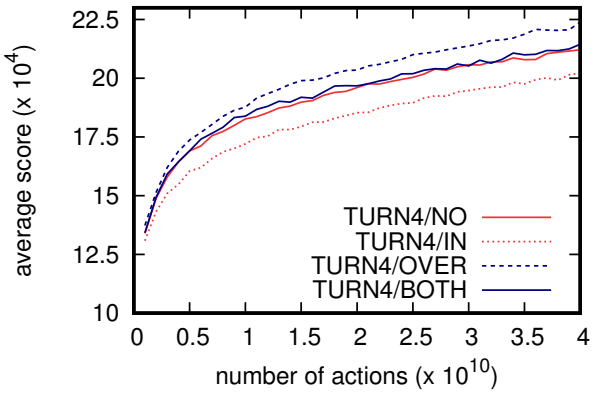


Figure 9. Average scores of greedy plays with ST-TURN4

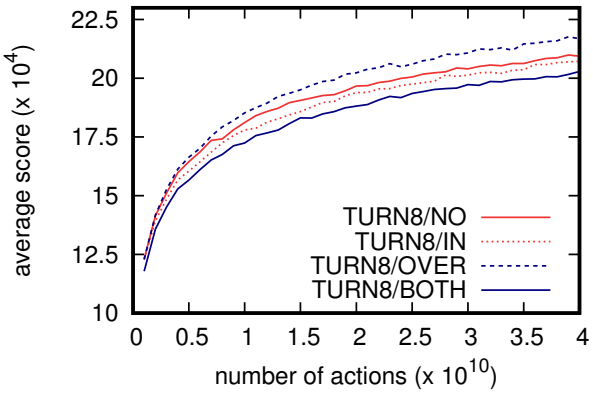


Figure 10. Average scores of greedy plays with ST-TURN8

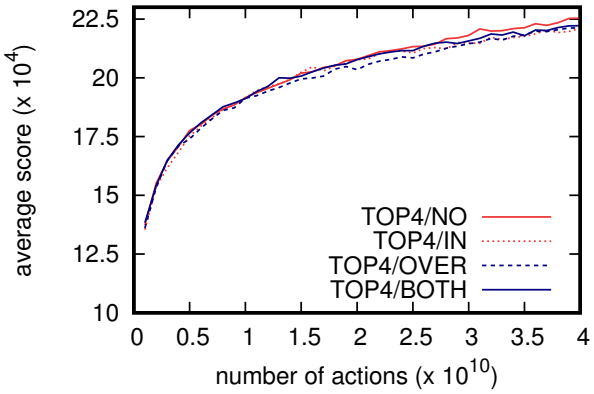


Figure 11. Average scores of greedy plays with ST-TOP4

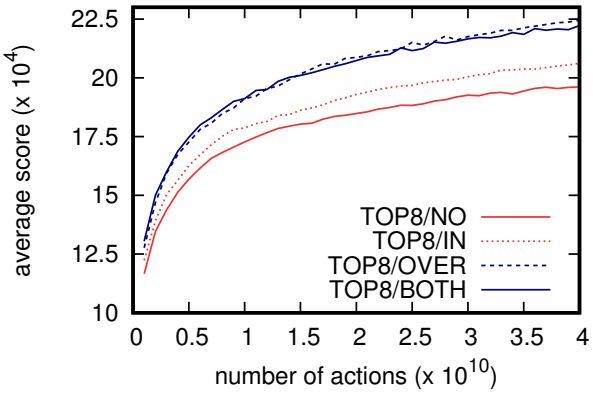


Figure 12. Average scores of greedy plays with ST-TOP8

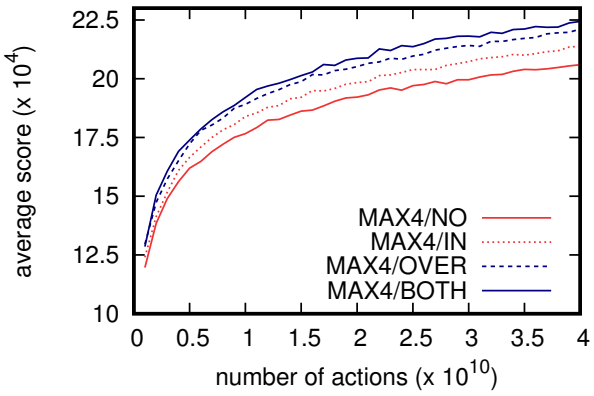


Figure 13. Average scores of greedy plays with ST-MAX4

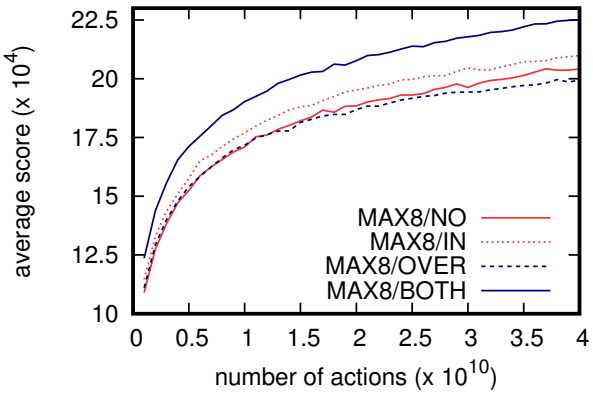


Figure 14. Average scores of greedy plays with ST-MAX8

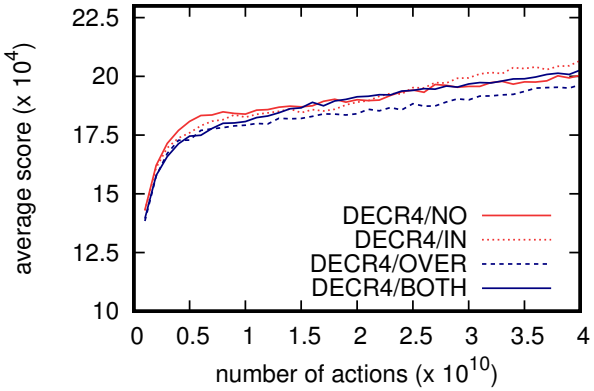


Figure 15. Average scores of greedy plays with ST-DECR4

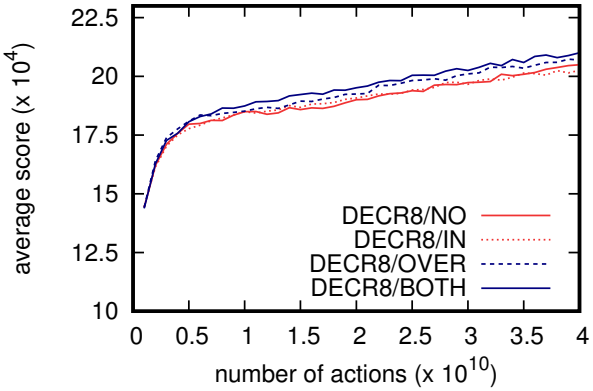


Figure 16. Average scores of greedy plays with ST-DECR8

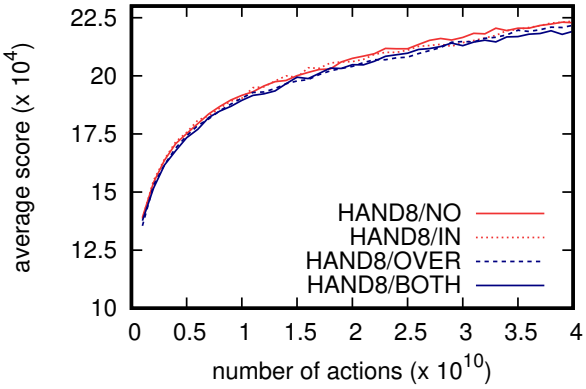


Figure 17. Average scores of greedy plays with ST-HAND8

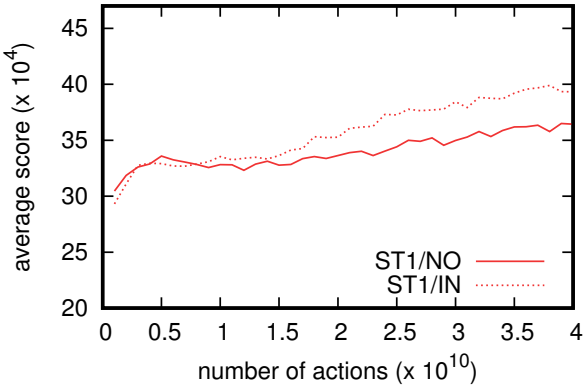


Figure 18. Average scores of 3-ply expectimax plays with ST-1

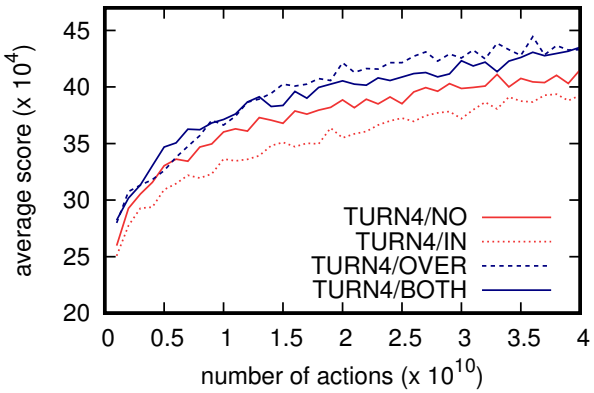


Figure 19. Average scores of 3-ply expectimax plays with ST-TURN4

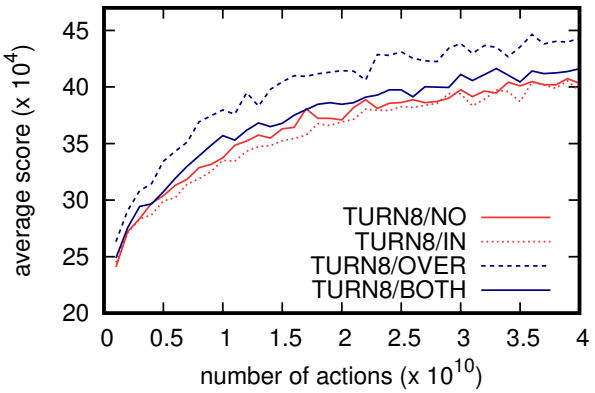


Figure 20. Average scores of 3-ply expectimax plays with ST-TURN8

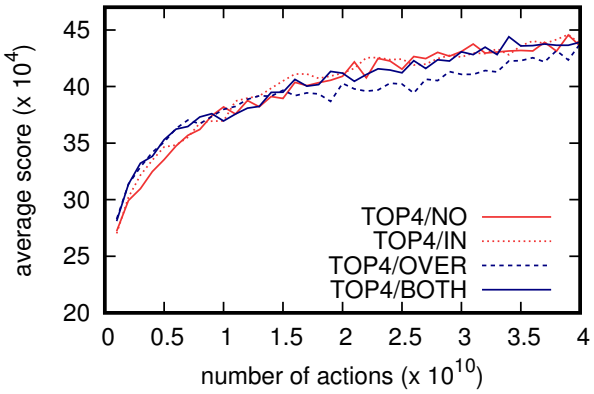


Figure 21. Average scores of 3-ply expectimax plays with ST-TOP4

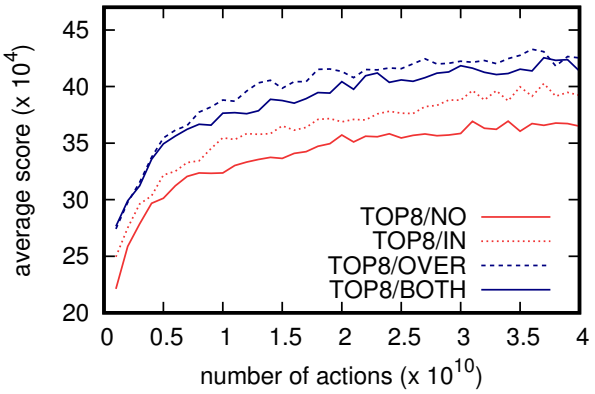


Figure 22. Average scores of 3-ply expectimax plays with ST-TOP8

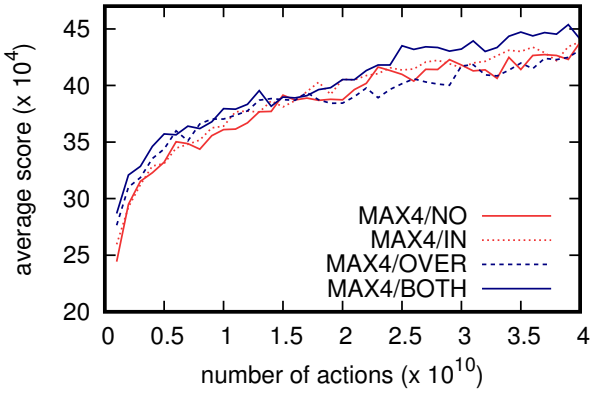


Figure 23. Average scores of 3-ply expectimax plays with ST-MAX4

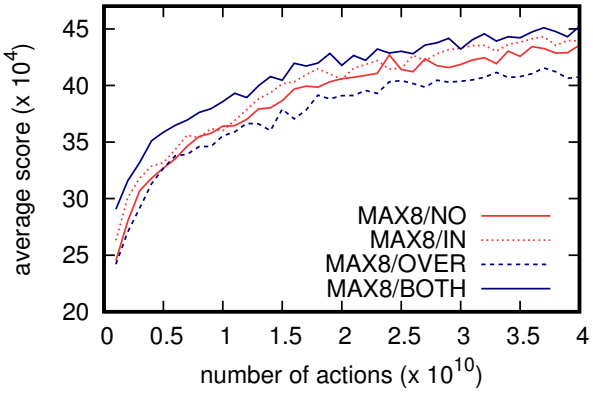


Figure 24. Average scores of 3-ply expectimax plays with ST-MAX8

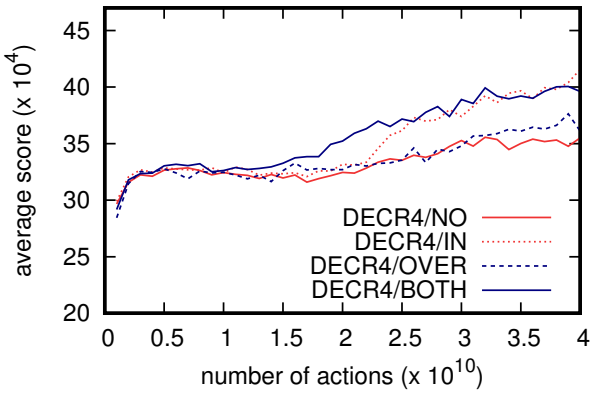


Figure 25. Average scores of 3-ply expectimax plays with ST-DECR4

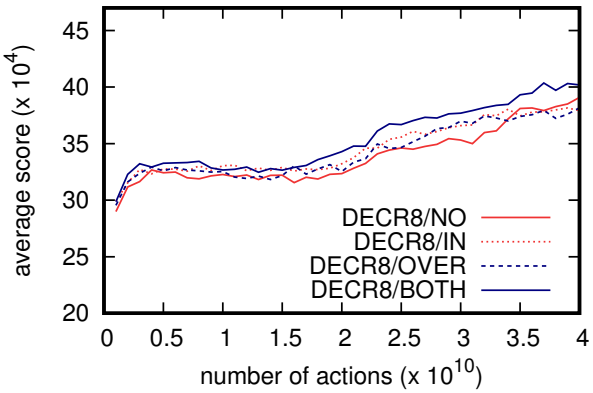


Figure 26. Average scores of 3-ply expectimax plays with ST-DECR8

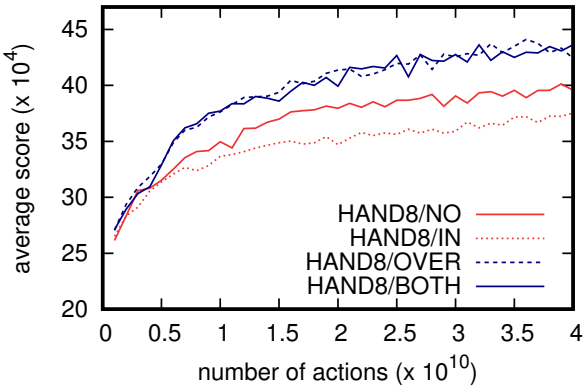


Figure 27. Average scores of 3-ply expectimax plays with ST-HAND8

Table 1. The best scores after learning 1×10^{10} actions. The best scores among the settings with the same number of stages are in red. The best scores among the settings with the same staging method are in blue (except for the red ones). The range shows the 95% confidence interval.

settings		Greedy		Expectimax (3-ply)	
		Average	Maximum	Average	Maximum
ST-1	PR-NO	191,105 ± 9,072	602,674 ± 49,093	328,259 ± 8,317	574,062 ±223,133
	PR-IN	183,413 ± 8,062	617,840 ± 8,942	335,341 ±17,035	723,726 ± 69,711
ST-TURN4	PR-NO	182,648 ± 5,479	619,460 ± 14,314	360,178 ± 5,564	697,912 ± 48,872
	PR-IN	172,006 ±17,637	571,919 ± 60,809	336,227 ±37,382	688,895 ± 60,147
	PR-OVER	187,690 ± 4,412	660,650 ± 68,716	366,376 ±21,422	739,780 ± 4,079
	PR-BOTH	183,804 ±11,580	655,443 ± 38,002	371,207 ±21,525	742,008 ± 25,257
ST-TOP4	PR-NO	191,588 ± 6,432	644,069 ± 34,566	381,873 ±21,351	708,709 ± 21,924
	PR-IN	191,540 ±11,874	661,712 ± 29,623	369,471 ±13,512	720,161 ± 18,815
	PR-OVER	191,266 ± 5,550	707,668 ± 27,937	379,863 ±29,976	772,717 ± 34,753
	PR-BOTH	191,149 ± 6,187	695,739 ± 70,792	369,522 ± 9,762	777,060 ± 31,926
ST-MAX4	PR-NO	176,616 ±11,813	667,452 ± 80,069	360,941 ±16,402	747,325 ± 20,264
	PR-IN	183,930 ± 3,363	645,340 ± 45,502	364,074 ±14,790	758,870 ± 30,916
	PR-OVER	189,090 ± 7,323	727,040 ± 47,104	370,324 ±21,317	797,101 ± 23,113
	PR-BOTH	191,990 ±10,881	701,542 ± 27,707	379,454 ±22,310	788,651 ± 41,476
ST-DECR4	PR-NO	183,888 ±11,004	457,968 ± 81,658	324,369 ± 8,893	504,031 ± 90,391
	PR-IN	182,578 ± 8,780	543,626 ± 38,432	324,180 ± 4,439	602,633 ± 31,453
	PR-OVER	179,208 ± 7,783	530,653 ±108,045	324,682 ± 6,996	575,672 ±132,289
	PR-BOTH	180,727 ± 4,060	584,952 ± 48,144	326,177 ±10,892	678,284 ± 87,288
ST-TURN8	PR-NO	181,097 ± 7,723	590,018 ± 10,279	337,391 ±16,362	607,850 ± 16,485
	PR-IN	177,926 ± 8,459	563,432 ± 26,947	335,080 ±19,354	600,372 ± 5,494
	PR-OVER	185,220 ±11,430	700,059 ± 42,183	379,681 ±26,955	718,041 ± 28,531
	PR-BOTH	172,424 ±19,055	600,288 ± 25,904	356,993 ±41,618	712,031 ± 72,122
ST-TOP8	PR-NO	172,737 ±13,782	609,090 ± 19,754	323,570 ±25,949	622,199 ± 4,032
	PR-IN	178,649 ± 7,133	619,263 ± 43,976	354,611 ±10,994	668,213 ± 47,384
	PR-OVER	191,130 ±10,631	713,830 ± 62,947	388,033 ±24,909	777,738 ± 37,401
	PR-BOTH	191,128 ±10,965	685,677 ± 54,869	376,403 ± 9,457	761,118 ± 33,014
ST-MAX8	PR-NO	170,945 ±11,292	685,484 ± 15,904	363,967 ±18,729	766,110 ± 32,337
	PR-IN	176,987 ± 9,566	679,396 ± 62,328	360,022 ±18,069	777,741 ± 10,864
	PR-OVER	171,682 ±24,331	642,304 ± 71,747	355,414 ±56,487	742,683 ± 98,100
	PR-BOTH	190,260 ± 7,379	710,836 ± 96,784	385,696 ±22,571	807,007 ± 19,893
ST-DECR8	PR-NO	184,949 ± 5,944	504,403 ± 25,702	322,618 ± 8,146	580,269 ± 60,548
	PR-IN	184,941 ± 9,605	564,382 ± 48,034	330,445 ± 7,626	607,803 ± 42,921
	PR-OVER	185,135 ± 2,913	564,087 ± 61,479	325,309 ± 5,939	483,021 ±163,304
	PR-BOTH	187,413 ± 6,027	572,336 ± 60,203	326,613 ±14,047	594,220 ±190,487
ST-HAND8	PR-NO	191,583 ± 7,127	622,954 ± 63,009	349,689 ±22,487	649,608 ± 37,972
	PR-IN	190,335 ± 9,608	579,525 ± 46,641	336,537 ±14,779	647,101 ± 30,625
	PR-OVER	190,418 ± 3,941	655,175 ± 44,474	376,241 ±24,552	726,916 ± 30,254
	PR-BOTH	189,480 ± 4,917	659,546 ± 62,328	377,066 ±22,525	731,672 ± 10,476

Table 2. The best scores after learning 2×10^{10} actions. The best scores among the settings with the same number of stages are in red. The best scores among the settings with the same staging method are in blue (except for the red ones). The range shows the 95% confidence interval.

settings		Greedy		Expectimax (3-ply)	
		Average	Maximum	Average	Maximum
ST-1	PR-NO	194,340 ± 9,750	617,701 ± 18,054	336,271 ±14,773	613,472 ±176,388
	PR-IN	190,426 ± 8,870	663,674 ± 76,387	352,650 ±26,932	742,834 ± 23,841
ST-TURN4	PR-NO	195,976 ± 4,346	663,924 ± 59,905	388,545 ± 8,361	755,055 ± 36,979
	PR-IN	185,301 ±22,783	639,536 ± 53,721	355,048 ±42,886	721,838 ± 19,703
	PR-OVER	203,457 ± 3,536	725,868 ± 13,428	421,471 ±15,707	783,701 ± 37,920
	PR-BOTH	196,750 ±14,068	723,875 ± 15,144	405,351 ±16,891	786,048 ± 16,420
ST-TOP4	PR-NO	207,912 ± 7,412	724,649 ± 9,016	409,099 ±14,100	771,526 ± 38,413
	PR-IN	207,747 ±12,847	716,194 ± 14,855	411,805 ±15,504	764,543 ± 28,063
	PR-OVER	203,485 ± 6,004	776,343 ± 42,803	402,789 ±27,923	810,677 ± 6,207
	PR-BOTH	207,745 ±10,862	759,503 ± 47,171	411,853 ± 9,159	817,232 ± 17,498
ST-MAX4	PR-NO	192,165 ±13,296	741,230 ± 54,315	387,257 ±18,266	810,477 ± 24,792
	PR-IN	198,342 ± 3,827	760,640 ± 54,564	405,531 ±25,874	818,141 ± 19,235
	PR-OVER	205,145 ± 7,297	775,147 ± 27,627	384,367 ±11,466	808,854 ± 19,672
	PR-BOTH	208,680 ±12,459	755,752 ± 39,463	405,184 ±32,934	804,019 ± 45,551
ST-DECR4	PR-NO	190,004 ± 6,758	550,709 ± 72,783	324,444 ±12,397	538,263 ±181,401
	PR-IN	188,989 ± 8,476	635,400 ± 24,371	331,733 ±11,410	647,925 ± 80,854
	PR-OVER	184,076 ± 5,872	604,156 ±113,596	326,960 ±14,574	587,882 ±231,482
	PR-BOTH	191,304 ± 7,740	681,431 ± 70,001	352,213 ±27,442	749,678 ± 24,045
ST-TURN8	PR-NO	196,661 ± 6,517	666,222 ± 57,982	370,884 ±25,155	706,318 ± 72,299
	PR-IN	193,886 ± 7,473	617,816 ± 15,096	369,090 ±16,098	671,519 ± 51,185
	PR-OVER	202,277 ±15,916	741,746 ± 35,154	414,352 ±13,436	794,433 ± 18,914
	PR-BOTH	188,098 ±25,770	671,800 ± 71,987	384,626 ±47,967	747,756 ± 34,763
ST-TOP8	PR-NO	184,799 ±17,081	658,957 ± 52,837	357,085 ±38,050	736,619 ± 34,137
	PR-IN	192,916 ± 9,935	686,704 ± 48,137	368,697 ±27,314	737,193 ± 23,073
	PR-OVER	208,689 ±11,473	715,280 ± 59,442	413,291 ±30,438	796,742 ± 40,847
	PR-BOTH	207,338 ±12,136	708,478 ± 64,635	404,274 ±14,479	764,408 ± 33,240
ST-MAX8	PR-NO	188,430 ±16,700	747,304 ± 29,998	406,011 ±24,726	818,402 ± 5,949
	PR-IN	195,170 ±10,116	748,357 ± 25,564	405,331 ±20,129	814,432 ± 21,072
	PR-OVER	186,877 ±28,477	736,267 ± 80,880	390,952 ±66,801	795,540 ± 40,132
	PR-BOTH	207,687 ± 6,101	759,861 ± 28,856	417,712 ±18,350	818,780 ± 18,304
ST-DECR8	PR-NO	190,005 ± 5,423	573,472 ± 48,751	323,445 ±11,988	623,301 ± 10,443
	PR-IN	190,775 ±10,129	629,240 ± 75,328	332,100 ±16,292	689,988 ± 50,895
	PR-OVER	192,451 ± 5,411	610,637 ± 98,328	325,216 ±23,340	572,732 ±235,544
	PR-BOTH	195,141 ± 3,680	681,556 ± 68,322	342,823 ±21,635	706,875 ± 90,820
ST-HAND8	PR-NO	207,522 ± 7,322	687,883 ± 53,330	379,524 ±43,073	705,548 ± 56,846
	PR-IN	206,109 ± 7,512	637,828 ± 54,542	347,013 ±16,780	713,320 ± 31,132
	PR-OVER	204,064 ± 6,629	737,680 ± 43,130	413,629 ±26,463	791,467 ± 15,215
	PR-BOTH	204,746 ± 4,044	736,442 ± 44,254	399,183 ±10,759	797,419 ± 32,972

Table 3. The best scores after learning 3.8×10^{10}, 3.9×10^{10} or 4.0×10^{10} actions. The best scores among the settings with the same number of stages are in red. The best scores among the settings with the same staging method are in blue (except for the red ones). The range shows the 95% confidence interval.

settings		Greedy		Expectimax (3-ply)	
		Average	Maximum	Average	Maximum
ST-1	PR-NO	205,693 ± 7,643	682,379 ±65,972	364,908 ±50,207	707,538 ±107,314
	PR-IN	199,748 ± 7,380	734,709 ± 8,485	398,933 ±47,054	773,132 ± 43,895
ST-TURN4	PR-NO	212,059 ± 5,690	724,877 ±37,265	415,263 ±18,666	795,099 ± 11,145
	PR-IN	202,110 ±29,055	704,207 ±56,979	393,525 ±52,388	796,976 ± 29,702
	PR-OVER	223,884 ± 6,528	783,951 ±41,068	436,666 ± 7,169	831,120 ± 5,235
	PR-BOTH	214,411 ±14,180	783,806 ±51,386	435,113 ±13,585	828,444 ± 7,313
ST-TOP4	PR-NO	225,350 ± 7,635	772,820 ±22,417	445,429 ± 9,602	810,192 ± 22,297
	PR-IN	221,073 ±12,243	769,588 ±23,713	445,950 ± 8,266	803,621 ± 23,765
	PR-OVER	221,273 ±12,484	789,704 ±39,760	437,984 ±26,002	823,971 ± 11,874
	PR-BOTH	222,210 ±13,518	799,786 ±21,957	439,192 ±16,026	829,267 ± 8,326
ST-MAX4	PR-NO	205,989 ±17,623	776,203 ±30,764	435,660 ±37,597	814,376 ± 17,328
	PR-IN	213,778 ± 6,686	783,090 ±12,558	439,730 ±19,067	818,372 ± 4,819
	PR-OVER	221,139 ± 9,582	794,576 ±10,590	412,056 ±58,594	816,891 ± 19,227
	PR-BOTH	224,430 ±12,315	790,468 ± 6,690	452,303 ±24,151	816,567 ± 7,016
ST-DECR4	PR-NO	200,401 ± 7,102	703,081 ±67,252	390,898 ±52,574	738,821 ± 44,676
	PR-IN	206,716 ±15,169	735,319 ±18,315	381,499 ±29,678	757,686 ± 33,724
	PR-OVER	196,471 ±11,281	765,772 ±70,459	381,582 ±63,982	790,544 ± 54,847
	PR-BOTH	202,670 ± 8,086	795,707 ±24,785	403,038 ±26,243	824,875 ± 7,085
ST-TURN8	PR-NO	209,905 ± 7,720	733,087 ±13,423	407,298 ±33,118	794,913 ± 41,462
	PR-IN	207,230 ± 9,518	726,439 ±22,037	405,330 ±18,542	754,338 ± 39,292
	PR-OVER	217,613 ±15,060	769,738 ±31,206	443,226 ±14,821	814,533 ± 2,698
	PR-BOTH	202,876 ±32,186	739,674 ±49,641	416,163 ±53,672	818,976 ± 10,399
ST-TOP8	PR-NO	196,143 ±20,204	719,239 ±65,025	367,673 ±33,781	774,576 ± 32,594
	PR-IN	206,156 ±10,538	731,569 ±74,433	394,826 ±34,661	782,421 ± 20,430
	PR-OVER	224,440 ±12,131	802,050 ±27,617	426,433 ±22,167	828,361 ± 6,186
	PR-BOTH	222,235 ±11,127	761,245 ±38,619	423,731 ±17,663	825,834 ± 10,352
ST-MAX8	PR-NO	204,151 ±18,631	773,231 ± 7,968	438,248 ±21,916	810,544 ± 17,142
	PR-IN	209,861 ±10,160	789,212 ±20,301	438,537 ±13,533	817,980 ± 8,213
	PR-OVER	199,620 ±32,232	757,054 ±56,752	431,379 ±29,783	810,409 ± 11,304
	PR-BOTH	225,078 ±10,506	793,422 ±13,224	453,776 ±17,662	817,252 ± 2,861
ST-DECR8	PR-NO	204,924 ±13,961	706,982 ±64,746	355,115 ±49,098	730,139 ± 60,083
	PR-IN	202,557 ±14,537	730,839 ±29,040	415,652 ±15,894	779,689 ± 7,854
	PR-OVER	207,300 ± 8,292	770,782 ±78,546	376,304 ±43,852	796,070 ± 52,797
	PR-BOTH	210,002 ± 2,409	791,286 ±16,743	400,411 ±29,243	826,751 ± 6,141
ST-HAND8	PR-NO	223,052 ±12,242	732,931 ±40,305	401,296 ±51,577	766,470 ± 48,576
	PR-IN	223,519 ± 9,020	697,366 ±63,042	375,294 ± 9,801	752,434 ± 26,361
	PR-OVER	221,715 ±11,827	783,018 ±32,430	433,189 ±26,861	826,613 ± 5,338
	PR-BOTH	219,352 ± 7,972	783,557 ±41,944	436,291 ±12,885	823,411 ± 14,496

2048におけるマルチステージ化と重み昇進手法の有効性評価

松崎　公紀*

高知工科大学情報学群
〒782–8502　高知県香美市土佐山田町宮ノ口185

* E-mail: matsuzaki.kiminori@kochi-tech.ac.jp

要約：ゲーム「2048」は，スライド＆マージ型の確率的一人ゲームである.「2048」におけるコンピュータプレイヤとして，Nタプルネットワークを評価関数とし，それを強化学習により調整したものが提案され，強いコンピュータプレイヤが作られている．さらにその改良法として，評価関数のマルチステージ化や学習における重み昇進手法についていくつかの論文で提案されている．しかし，それらの手法の有効性を直接比較した結果は示されていない．本論文では，マルチステージ化と重み昇進手法について，本論文で新しく提案する手法を含め網羅的に実験を行い，その有効性を比較評価する．その結果，平均値と最大値のどちらを最大化するかによって，最適なマルチステージ化と重み昇進手法が異なった．タイルの最大値を用いるマルチステージ化と2つの重み昇進手法を用いて学習した評価関数を用いるプレイヤは，貪欲的なプレイで平均225,078点，3層Expectimaxによるプレイで平均453,776点を達成した．