

# Systematic Selection of N-Tuple Networks with Consideration of Interinfluence for Game 2048

Kiminori Matsuzaki  
School of Information  
Kochi University of Technology  
Kami, Kochi 782–8502 Japan  
Email: matsuzaki.kiminori@kochi-tech.ac.jp

**Abstract**—The puzzle game 2048, a single-player stochastic game played on a  $4 \times 4$  grid, is the most popular among similar slide-and-merge games. One of the strongest computer players for 2048 uses temporal difference learning (TD learning) on so called *N-tuple networks*, where the shapes of the *N*-tuples are given by human based on characteristics of the game. In our previous work (Oka and Matsuzaki, 2016), the authors proposed a systematic method of selecting *N*-tuples under an assumption that the interinfluence among those *N*-tuple networks are negligible. Though the selected *N*-tuple networks worked fine, there were large gaps between those *N*-tuple networks and the human-designed networks. In this paper, another systematic and game-characteristics-free method of selecting *N*-tuples is proposed for game 2048, in which the interinfluence among those *N*-tuple networks is captured. The proposed method is effective and generic: the selected *N*-tuple networks are as good as human-designed ones under the same setting, and we can obtain larger (or smaller) *N*-tuple networks in the same manner. We also report the experiment results when we combine the *N*-tuple networks and expectimax search.

## I. INTRODUCTION

The puzzle game 2048 [1], a single-player stochastic game played on a  $4 \times 4$  grid, is the most popular among similar slide-and-merge games like Threes and 1024. 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 artificial intelligence and computational complexity. For instance, the difficulty of the game was discussed from the viewpoint of computational complexity by Abdelkader et al. [2] and Langerman and Uno [3]. As a testbed of artificial intelligence methods, there have been some competitions of computer players for the game 2048 [4], [5] and for a two-player version of 2048 [6]–[8].

One of the strongest computer players for 2048 uses the expectimax algorithm with an evaluation function learned from a large number of self-plays, where the evaluation function is designed based on *N-tuple networks*. An *N*-tuple network consists of a tuple of size *N* and a corresponding table of feature weights: each *N*-tuple covers contributes a number of features each for one distinct occurrence of tiles that the *N*-tuple covers. Given *N*-tuple networks, the evaluation function simply calculates the summation of weights for all occurring features, where those weights were adjusted through the temporal difference learning (TD learning for short) [9] or its extended algorithm [10], [11].

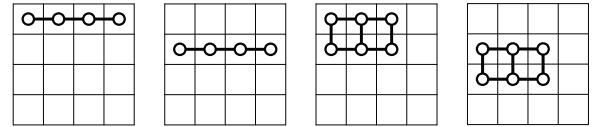


Fig. 1. The two 4-tuples and two 6-tuples by Szubert and Jaśkowski [9]

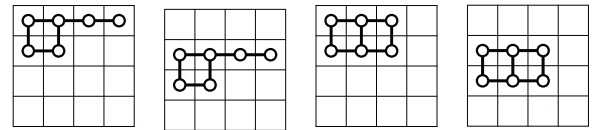


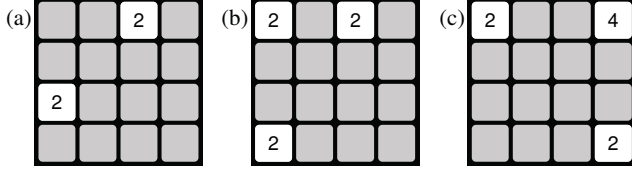
Fig. 2. The four 6-tuples by Wu et al. [11]

In this approach, the design of *N*-tuples is an important issue. The first work by Szubert and Jaśkowski [9] used two 6-tuples and two 4-tuples (Fig. 1), and Wu et al. [10], [11] extended them to four 6-tuples (Fig. 2). As we can easily imagine, the more and the larger tuples we use the better evaluation functions we would obtain, but computing resources limit the number and/or size of tuples. So, the question we have is like “*what is the best N-tuple networks that consist of at most eight 7-tuples.*”

In our previous work [12], we proposed a systematic method of selecting those *N*-tuples without heuristics or characteristics of the game. Since the method deals with the *N*-tuples to be independent, *i.e.* it does not take into account the interinfluence of *N*-tuples, the selected *N*-tuple networks are not good enough when the number of tuples is limited. In this paper, I propose another method of selecting *N*-tuples to resolve this problem. The idea is simple: starting from an empty set, we augment it by adding the best *N*-tuple greedily one by one. From the experiments, the proposed method successfully constructs *N*-tuple networks, and the constructed networks with four 6-tuples perform almost as well as the human-designed ones [11] do.

The main contributions of the paper are summarized as follows:

- I propose a systematic way of constructing *N*-tuple networks. It does not rely on heuristics or human knowledge of the games, and thus we can also apply this technique to other games in a same manner.



(a) An example of the initial state. Two tiles are put randomly.  
(b) After moving up. A new 2-tile appears at the lower-left corner.  
(c) After moving right. Two 2-tiles are merged to a 4-tile, and score 4 is given.

Fig. 3. The process of the game 2048

- The method does not pose an assumption that the  $N$ -tuples are independent. Therefore the constructed  $N$ -tuple networks perform as well as human-designed ones do.
- The simple greedy player achieved average score 255,198 (with eight 7-tuples). The expectimax player achieved the maximum score 629,964 (with eight 7-tuples, look-aheads to 3 moves).

The rest of the paper is organized as follows. Section II briefly introduces the rule of the game 2048. Section III reviews the idea of applying  $N$ -tuple networks and TD learning to the game 2048. Section IV reviews the related work focusing on TD-learning-based players for 2048. Section V proposes a new systematic method of constructing  $N$ -tuple networks, and reports the obtained  $N$ -tuple networks for the cases of  $N = 6$  and  $N = 7$ . With the obtained  $N$ -tuple networks, Section VI conducts more experiments including the application of expectimax search. Finally Section VII concludes the paper.

## II. RULES OF 2048

The game 2048 is played on a  $4 \times 4$  grid. The objective of the original 2048 game is to reach a 2048 tile by moving and merging the tiles on the board according to the rules below. A new tile will be put randomly with number 2 (with probability of 90%) or 4 (with probability of 10%). In the initial state, two tiles are put randomly (Fig. 3). The player selects a direction (either up, left, down, or right), and then all the tiles will move in that direction. When two tiles of the same number combine 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 does 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 at an empty cell with the same probability above. If the player cannot move the tiles, the game ends.

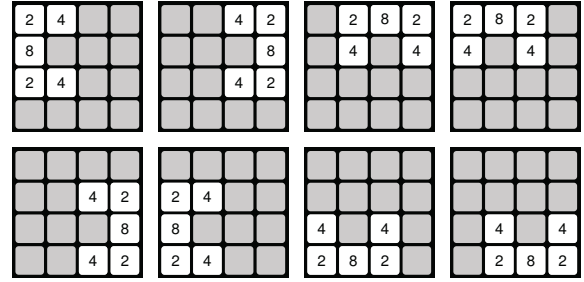
## III. $N$ -TUPLE NETWORKS AND TEMPORAL DIFFERENCE LEARNING FOR 2048

This section reviews the idea of applying  $N$ -tuple networks and TD learning to the game 2048. The algorithm was first given by Szubert and Jaśkowski [9] and it was called TD-AFTERSTATE in their paper.

(a) Two 3-tuples and their feature weights

<p>Tuple A</p>	Table for A				Table for B			
	0	1	2	weight	0	1	4	weight
	—	—	—	1280.5	—	—	—	1210.1
	—	—	2	2351.4	—	—	2	581.4
	—	—	—	⋮	—	—	—	⋮
<p>Tuple B</p>	2	4	—	3472.5	2	4	—	1347.1
	2	4	2	3133.9	2	4	2	3513.3
	2	4	4	3324.8	2	4	4	2332.9
	2	4	8	3018.4	2	4	8	1801.8
	—	—	—	⋮	—	—	—	⋮

(b) A state and its symmetries (by rotation & reflection)



(c) The evaluation value for the state  $s$

$$V(s) = (3472.5 + 1801.8) + \dots + (1280.5 + 1210.1)$$

Fig. 4. An example for calculating an evaluation value of a state

## A. Evaluation Function with $N$ -tuple Networks

An  $N$ -tuple network consists of an  $N$ -tuples and a corresponding table having feature weights, where the  $N$ -tuple covers  $N$  cells on the grid. In this paper,  $N$  denotes the number of cells in a tuple, and  $m$  the number of tuples used in a set. If each cell in the tuple may have one of 16 values (Empty, 2, 4, 8, 16, ..., 32768) an  $N$ -tuple contributes  $16^N$  features, that is, we can assign a feature weight for each of  $16^N$  features. Note that 6- and 7-tuple networks require 64 MB and 1 GB, respectively, if we assign 32 bits for each feature weight.

Given an  $N$ -tuple network, we calculate the value of an evaluation function of a state as follows. Since the board of the game 2048 is symmetric in terms of rotation and reflection, we can consider 8 sampling for each  $N$ -tuple. We take the feature weight for each sampling, and compute the sum of those values as the evaluation value of the state. Given a state  $s$ , the evaluation value  $V(s)$  of the state is the sum of the feature weights for all  $N$ -tuple networks and for all symmetric boards.

Let us see an example in Fig. 4 where we illustrate  $N$ -tuple networks with two 3-tuples. We have eight symmetric boards for a state  $s$ , and each board has two feature weights. Therefore, in this example, the evaluation value of a state is the sum of 16 feature weights. If we have  $m$  networks of  $N$ -tuples, then the evaluation value of a state is the sum of  $8m$  feature weights.

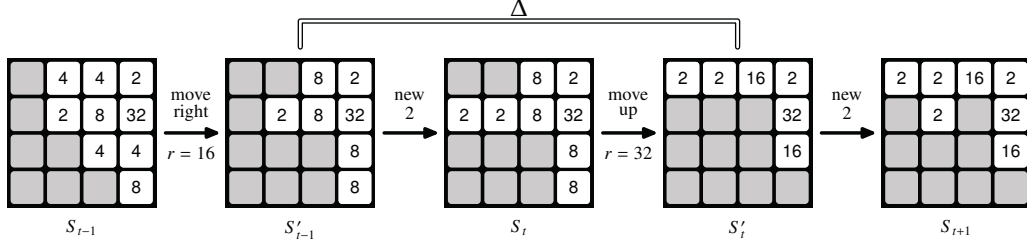


Fig. 5. Transition of states

The greedy 2048 player in this paper selects a move such that the sum of score and evaluation value is the maximum. For a state  $s$ , let the set of possible move be  $A(s) \subseteq \{N, E, S, W\}$ , the score given by move  $a$  be  $R(s, a)$ , and the next state by move  $a$  be  $N(s, a)$ , the player selects

$$\arg \max_{a \in A(s)} (R(s, a) + V(N(s, a))) .$$

### B. Temporal Difference Learning

Temporal difference learning (TD learning) is one of the reinforcement learning algorithms. Though the idea of TD learning was introduced by Sutton [13], its origins reach back to the 1950's for the famous program for checkers [14]. TD learning has been adapted to several games such as backgammon [15], Othello [16], and Go [17].

In our algorithm, the evaluation values are adjusted by TD learning as follows (Fig. 5). Let  $s_t$  be a state at time  $t$ . The player selects a move  $a$  such that the sum of score and evaluation value of the state after the move is the maximum. Let  $r = R(s_t, a)$  and  $s'_t$  be the score and the state after the move, respectively (note that  $s'_t \neq s_{t+1}$  because  $s_{t+1}$  is given by putting a tile on  $s'_t$ ). Then, the TD error  $\Delta$  for the evaluation value is defined as follows.

$$\Delta = r + V(s'_t) - V(s_t)$$

To reduce the TD error, we update the evaluation values  $V_j(s_{t-1})$  for all the networks by a certain portion of  $\Delta$ :

$$V'_j(s_{t-1}) = V_j(s_{t-1}) + \alpha \Delta$$

where the rate  $\alpha$  is called *learning rate* and it was set to  $\alpha = 2^{-10}$  throughout the experiments.

### IV. RELATED WORK

Several game-playing algorithms have been adapted to the game 2048 [9], [11], [18]–[22]. Among them, the state-of-the-art algorithms combine expectimax with TD learning or some other heuristics.

The first application of TD learning to 2048 players was done by Szubert and Jaśkowski [9]. They utilized hand-selected two 4-tuples and two 6-tuples and the player learned with 1,000,000 self-play games achieved the average score 100,178. The two 4-tuples were extended to two 6-tuples by Wu et al. [11] and the extension increased the average score to 142,727. In our previous work [12], we developed a systematic method of selecting  $N$ -tuples under an assumption that the

tuples are independent. The systematic selection worked fine if the number  $m$  of  $N$ -tuples is sufficiently large, but not enough for small  $m$ : the average score was 109,983 for four 6-tuples. Wu et al. [11] also proposed the multi-staged extension of the learning algorithm, and by the combination with expectimax search the player achieved the average score 328,946 (multi-staged TD, expectimax depth = 5). They recently achieved a 65536-tile [10].

The expectimax algorithm takes much more time when the depth is large. In the competition of computer players for the game 2048, it is often required to play a move in 1–10 milliseconds [4], [5]. For the execution times with expectimax and  $N$ -tuple networks, Yeh et al. [10] reports in details.

### V. AUGMENTATION-BASED CONSTRUCTION OF $N$ -TUPLE NETWORKS

Since an  $N$ -tuple covers a part of game board, it is straightforward to expect that a good selection of  $N$ -tuples should cover all the board. In fact, it is not enough to cover all the cells of the board, and we need to take into account the interinfluence between tuples to obtain good  $N$ -tuple networks. A possible way to do that is to define some degree of similarity between tuples, but it seems not so easy to define such a degree for the  $N$ -tuples in the game 2048. Therefore, in this paper, I propose another simple way, *augmentation-based construction* of  $N$ -tuple networks, that construct  $N$ -tuple networks by adding the best tuples one by one.

#### A. Algorithm

The following is the augmentation-based  $N$ -tuple network construction algorithm.

- 1) List up all the  $N$ -tuples and let  $\mathcal{T}$  be the set of possible tuples. In this work, we made all the tuples form a connected graph. We had 68 connected 6-tuples and 119 connected 7-tuples [12].
- 2) Initialized the set of  $N$ -tuples  $\mathcal{N}$  to be an empty set.
- 3) Repeat the following two subtasks.
  - a) For each tuple  $n_i$  in  $\mathcal{T} - \mathcal{N}$ , perform the TD learning over 1,000,000 self-play games with the tuples  $\mathcal{N} \cup \{n_i\}$ , and let the average of last 10,000 games be the score for  $n_i$ . To avoid variance due to the seeds of random numbers, we process this step 5 times and take the mean.
  - b) Pick up the tuple  $n_j$  with the largest score, and update the set  $\mathcal{N}$  with  $\mathcal{N} \cup \{n_j\}$ .

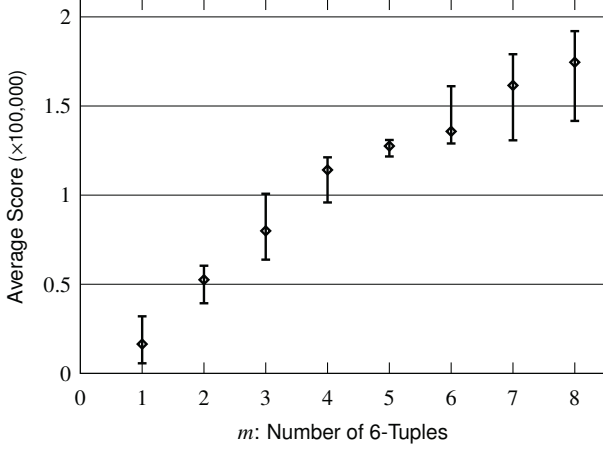


Fig. 6. The score of  $N$ -tuple networks in the augmentation-based construction for  $N = 6$ . The three points on the bar shows the score of the best and the worst networks and the average score.

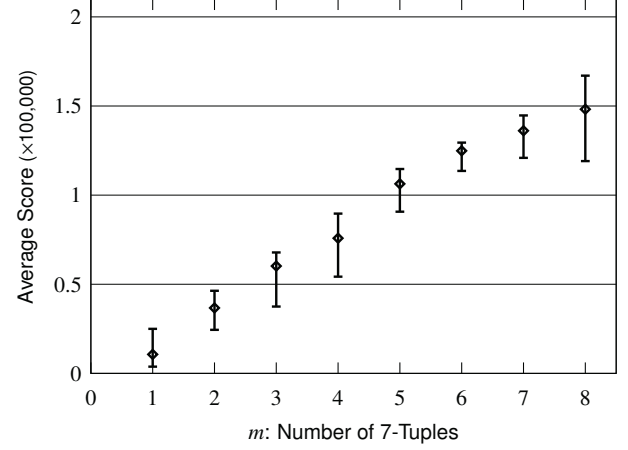


Fig. 7. The score of  $N$ -tuple networks in the augmentation-based construction for  $N = 7$ . The three points on the bar shows the score of the best and the worst networks and the average score.

### B. Experiment Results

Tables I and II show the results of the augmentation-based construction algorithm in Section V-A for  $N = 6$  and  $N = 7$ , respectively. Those tables show the best (with the highest score) four tuples and the worst (with the lowest score) tuple for each step of adding an  $N$ -tuple. We can obtain an  $N$ -tuple network with  $m$ -tuples just by selecting the first  $m$  tuples from the left in the *best* row.

We can find several interesting facts from these Tables.

- The four best tuples have a similar shape for each column ( $m$ ). In detail, only one cell is different between the best tuple and the second best one in five of eight cases for  $N = 6$  and six of eight cases for  $N = 7$ .
- For small  $m$ , the best tuples for  $m$  and those for  $m + 1$  have rather different shapes. This is an important evidence that the interinfluence of  $N$ -tuples is not negligible.
- (Surprising result) In the  $N = 7$  case, a tuple resulted in the worst one for  $m = 2, \dots, 6$ , but the same tuple became the best one for the following  $m = 7$ .
- (Important result) The best tuples for the first three corresponds to those in the hand-designed  $N$ -tuples by Wu et al. [11]. At  $m = 4$ , the right-most tuple in Fig. 2 was ranked at 43th of 65.

Figures 6 and 7 shows the variance of scores with respect to the number of  $N$ -tuples for  $N = 6$  and  $N = 7$ , respectively. We can see that the average score increases almost linearly with a little stepping down. Even though only one tuple was different for the same  $m$ , the variance of scores was rather large. Scores for  $m + 1$  tuples sometimes became smaller than those for  $m$  tuples, which is because we stop the TD learning at 1,000,000 games, i.e., they might not be converged yet.

### C. Time Complexity

Here, I would like to discuss the time complexity of the proposed augmentation-based construction algorithm. The algorithm requires to repeatedly perform learning with self-play

games. The total number of learning runs was  $5 \times (68 + 67 + \dots + 61) = 2580$  for 6-tuples and  $5 \times (119 + 118 + \dots + 112) = 4620$  for 7-tuples. Note that the time for each learning set depends on the number of tuples and the length of the game, and it could be 80 minutes for the case of  $N = 7$  and  $m = 8$  on a 12-core server environment.

In our previous work [8], we assumed independence among the tuples to reduce the number of learning runs. The total number of learning runs was 680 for 6-tuples and 1190 for 7-tuples. Therefore, the proposed algorithm takes 4 times as many runs as our previous work does.

## VI. EVALUATION OF SELECTED TUPLES

By using the  $N$ -tuples obtained in the previous section, I conducted two sets of experiments to evaluate them. The first set of experiments were to perform TD learning with more self-play games until the evaluation functions are converged. The second set of experiments were to apply the expectimax algorithm on top of the obtained  $N$ -tuple networks.

### A. Learning from More Self-Play Games

I performed the TD learning with 10,000,000 self-play games for each set of selected  $N$ -tuples.

Figures 8 and 9 shows the transition of scores (averaged for every 100,000 games) for  $N = 6$  and  $N = 7$  respectively. All the lines grew up smoothly without having significant dropping down. The lines for  $m = 5, 6, 7, 8$  eventually reached to similar scores (about 230,000 for  $N=6$  and about 260,000 for  $N=7$ ). These scores are in fact a bit larger than our estimation of upper bounds of (averaged) scores in [12], and I consider that the scores do not increase even if we use more tuples.

I also compared the  $N$ -tuple networks obtained by the proposed method with the human-designed networks [11] and the systematically generated ones in our previous work [12]. The other experiment conditions were the same except for the  $N$ -tuple networks. Figure 10 shows the experiment results for  $N = 6$ , which includes the proposed networks with

TABLE I  
LIST OF THE FOUR BEST AND THE WORST 6-TUPLES IN THE AUGMENTATION-BASED  $N$ -TUPLE NETWORK CONSTRUCTION FOR  $N = 6$ .

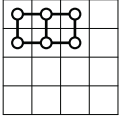
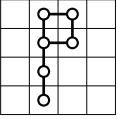
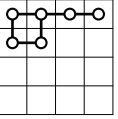
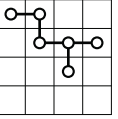
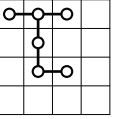
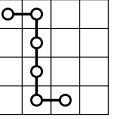
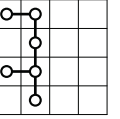
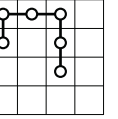
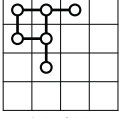
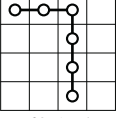
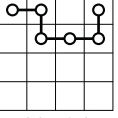
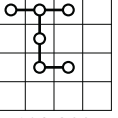
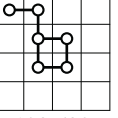
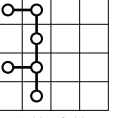
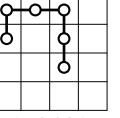
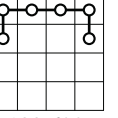
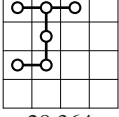
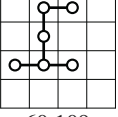
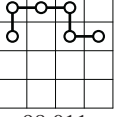
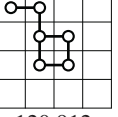
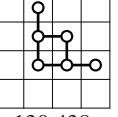
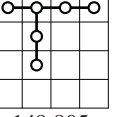
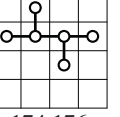
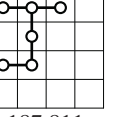
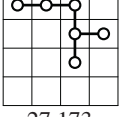
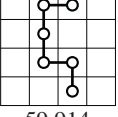
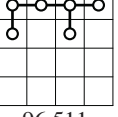
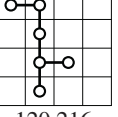
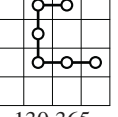
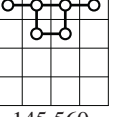
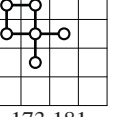
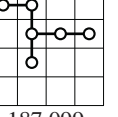
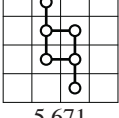
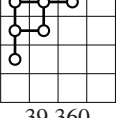
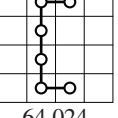
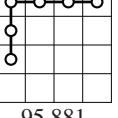
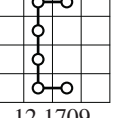
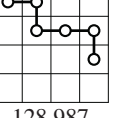
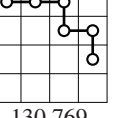
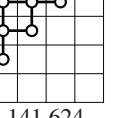
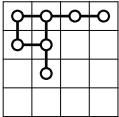
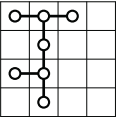
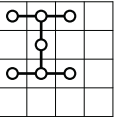
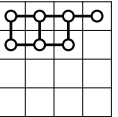
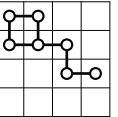
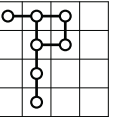
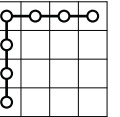
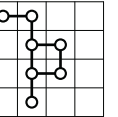
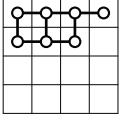
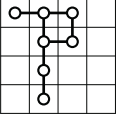
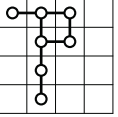
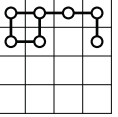
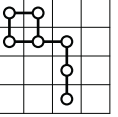
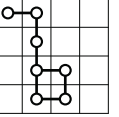
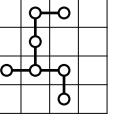
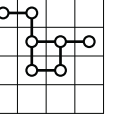
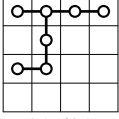
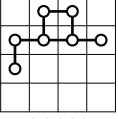
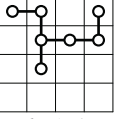
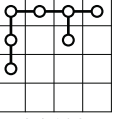
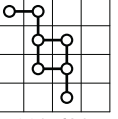
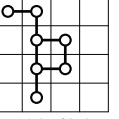
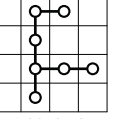
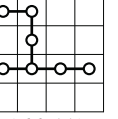
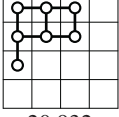
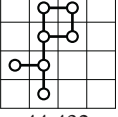
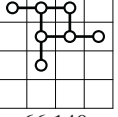
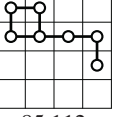
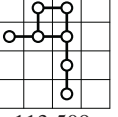
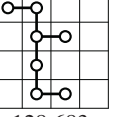
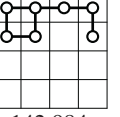
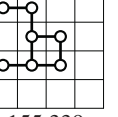
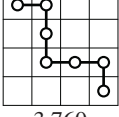
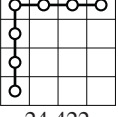
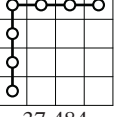
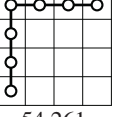
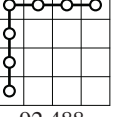
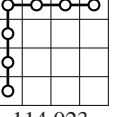
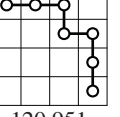
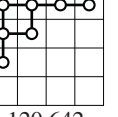
$m$	1	2	3	4	5	6	7	8
best	 32,019	 60,402	 100,790	 121,199	 130,946	 161,103	 179,072	 192,009
2nd	 31,611	 60,175	 98,725	 120,988	 130,508	 159,652	 174,301	 190,638
3rd	 28,364	 60,109	 98,011	 120,912	 130,438	 149,805	 174,176	 187,811
4th	 27,173	 59,914	 96,511	 120,216	 130,365	 145,560	 173,181	 187,099
worst	 5,671	 39,360	 64,024	 95,881	 12,1709	 128,987	 130,769	 141,624

TABLE II  
LIST OF THE FOUR BEST AND THE WORST 7-TUPLES IN THE AUGMENTATION-BASED  $N$ -TUPLE NETWORK CONSTRUCTION FOR  $N = 7$ .

$m$	1	2	3	4	5	6	7	8
best	 24,977	 46,321	 67,832	 89,645	 114,676	 129,469	 144,711	 167,035
2nd	 23,889	 46,166	 67,411	 85,874	 113,745	 128,835	 144,699	 166,865
3rd	 21,601	 45,030	 67,173	 85,128	 113,698	 128,695	 143,379	 166,551
4th	 20,932	 44,432	 66,140	 85,112	 113,509	 128,683	 142,984	 155,339
worst	 3,760	 24,422	 37,484	 54,261	 92,488	 114,023	 120,951	 120,642



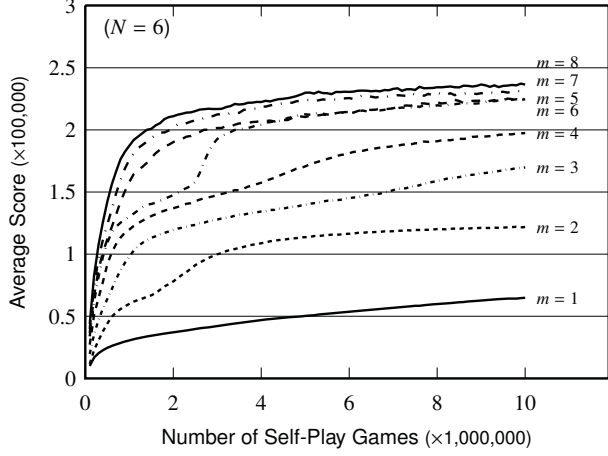


Fig. 8. Transitions of average scores over 10,000,000 self-play games for the constructed  $N$ -tuple networks for  $N = 6$  and  $m = 1, 2, \dots, 8$ .

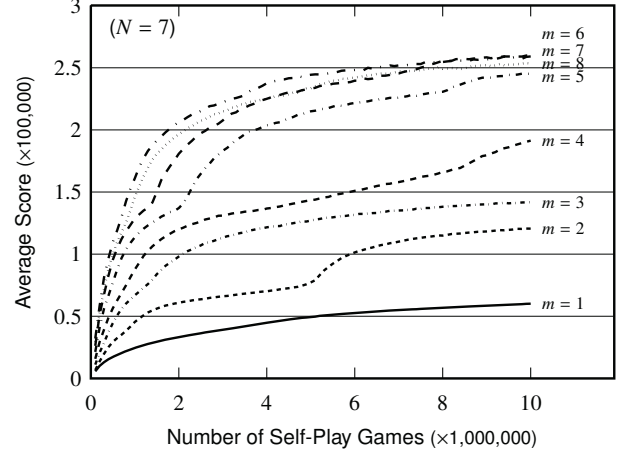


Fig. 9. Transitions of average scores over 10,000,000 self-play games for the constructed  $N$ -tuple networks for  $N = 7$  and  $m = 1, 2, \dots, 8$ .

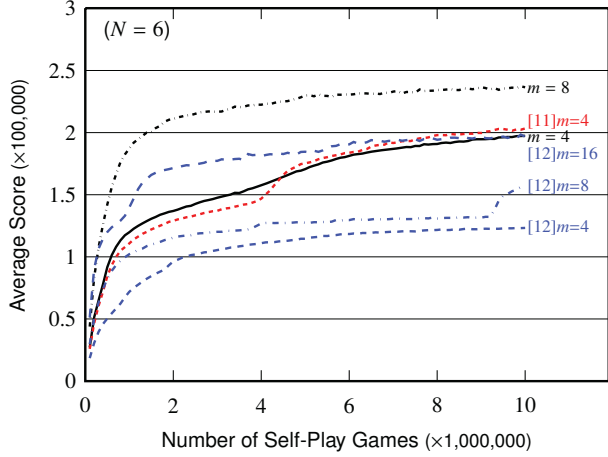


Fig. 10. Transitions of average scores over 10,000,000 self-play games compared with the  $N$ -tuple networks in our previous work [12] and the human-designed one by Wu et al. [11].

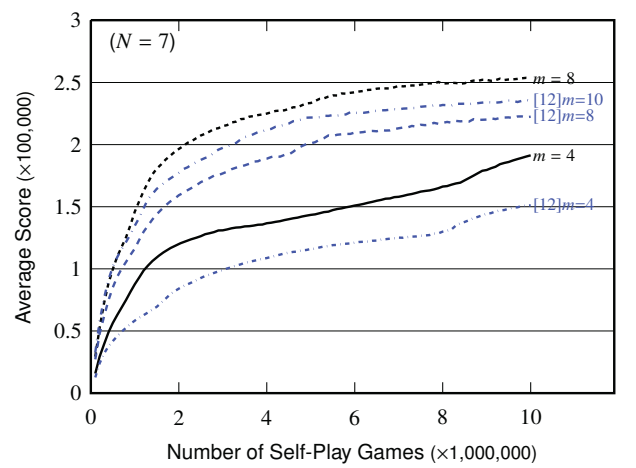


Fig. 11. Transitions of average scores over 10,000,000 self-play games compared with the  $N$ -tuple networks in our previous work [12].

$m = 4, 8$ , hand-designed ones [11] with  $m = 4$ , and those from our previous work with  $m = 4, 8, 16$ . Figure 11 shows the experiment results for  $N = 7$ , which includes the proposed networks with  $m = 4, 8$ , and those from our previous work with  $m = 4, 8, 10$ . From these results, we can clearly see that the  $N$ -tuple networks by the proposed method outperformed those by the previous work. Also, the obtained networks with four 6-tuples perform as well as human-designed ones do, and they outperformed until about 400,000 games.

### B. Combining with Expectimax Algorithm

It is known by several researchers that the minimax algorithm does not work well but the expectimax algorithm does for the game 2048. I implemented an expectimax player with the evaluation functions obtained in the first set of experiments, and evaluated the performance with a lot of experiments. The search depths  $d$  of the expectimax algorithm were from  $d = 1$  (choosing the maximum value without look-aheads) to  $d = 7$  (with look-aheads of 3 moves). For each  $N = 6, 7$  and

$m = 1, \dots, 8$ , I calculated the average and the maximum of scores of 5,000 games. Figures 12 and 13 show the average scores for  $N = 6$  and  $N = 7$  respectively, and Figures 14 and 15 show the maximum scores for  $N = 6$  and  $N = 7$  respectively.

The average scores for  $N = 6$  (Fig. 12) look reasonable. The gain of using expectimax search steps down with respect to the search depths. The average scores peak at  $m = 5$  tuples. The maximum scores for  $N = 6$  (Fig. 14) differ from our expectation: the peak of maximum scores are different for the search depths, but eventually at  $m = 8$  the maximum scores are almost the same regardless of the search depths. It remains as our future work to analyze in detail the reasons of these results.

The average scores for  $N = 7$  (Fig. 13) also differ from our expectation: the simple greedy algorithm ( $d = 1$ ) outperforms the expectimax algorithm for large  $m$ . A possible reason would be the overtraining to the greedy algorithm. In turn, the

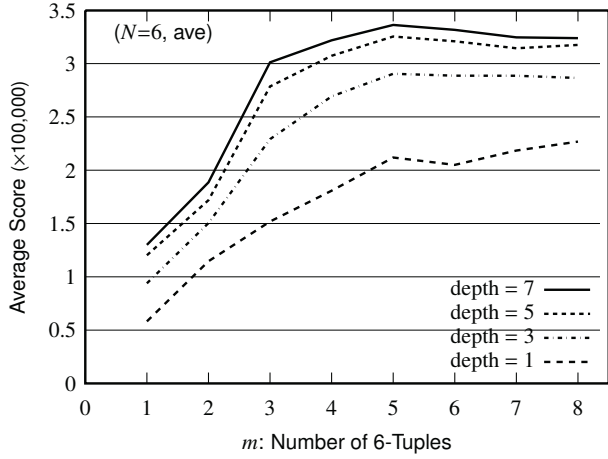


Fig. 12. The average scores of expectimax players (depth  $d = 1, \dots, 4$ ) based on the  $N$ -tuple network with  $N = 6$  and  $m = 1, \dots, 8$ .

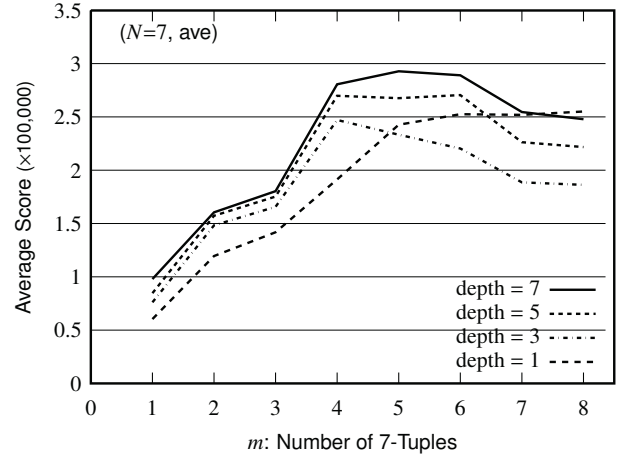


Fig. 13. The average scores of expectimax players (depth  $d = 1, \dots, 4$ ) based on the  $N$ -tuple network with  $N = 7$  and  $m = 1, \dots, 8$ .

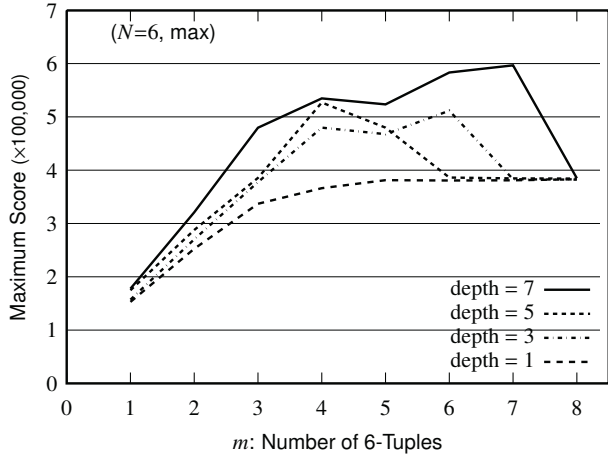


Fig. 14. The maximum scores of expectimax players (depth  $d = 1, \dots, 4$ ) based on the  $N$ -tuple network with  $N = 6$  and  $m = 1, \dots, 8$ .

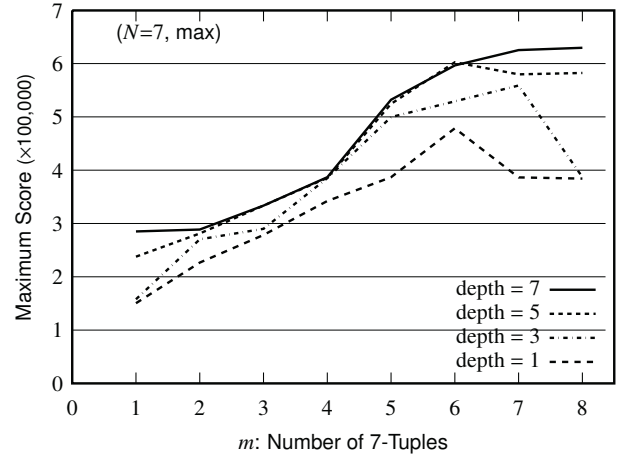


Fig. 15. The maximum scores of expectimax players (depth  $d = 1, \dots, 4$ ) based on the  $N$ -tuple network with  $N = 7$  and  $m = 1, \dots, 8$ .

maximum scores for  $N = 7$  (Fig. 15) look reasonable. The maximum scores 597,000 ( $N = 6, m = 7, d = 7$ ) and 629,964 ( $N = 7, m = 8, d = 7$ ) are larger than any other (simple) TD-learning players as far as the author knows. Note that the multi-staged algorithm [10], [11] could improve the scores for the networks of 6-tuples. (Due to the limitation of memory size, it may be hard to apply the multi-staged algorithm for many 7-tuples.)

Tables III and IV show the average scores, the maximum scores, and the reaching ratios that the player successfully made the tiles 2048, ..., 32768. In general, by using the expectimax search with deeper look-aheads, players made large tiles more successfully. In particular, the player with  $N = 6, m = 8, d = 4$  can make a 16,384 tile at the probability 93.9%. This would be a reason why the average scores increased for  $N = 6$ . In contrast, the results for  $N = 7, m = 8$  looks very strange. The reaching ratios decreased significantly with the expectimax search, and the reaching ratios for 2048 were even lower than 90%. It also remains

as our future work to analyze these results.

## VII. CONCLUSION

In this paper, a systematic method of constructing  $N$ -tuple networks was designed for the game 2048. The proposed method does not require heuristics or knowledge of characteristics about the game, and constructs  $N$ -tuple networks for given size and number of tuples.

Compared with our previous method, the proposed algorithm can cope with tuples that may have interinfluence each other. The constructed  $N$ -tuple networks perform as well as human-designed ones do, as we saw in the comparison with four 6-tuples by Wu et al. (Fig. 10). With the constructed  $N$ -tuple networks, the greedy player and the expectimax player achieved high scores. For example, the simple greedy player achieved average score 255,198 (with eight 7-tuples) and the expectimax player achieved the maximum score 629,964 (with eight 7-tuples, look-aheads to 3 moves).

TABLE III  
REACHING RATIOS, AVERAGE SCORES AND MAXIMUM SCORES OF THE EXPECTIMAX PLAYER (DEPTH  $d = 1, \dots, 4$ )  
BASED ON THE  $N$ -TUPLE NETWORKS WITH  $N = 6$  AND  $m = 4, 8$ .

$N = 6$	$m = 4$				$m = 8$			
	$d = 1$	$d = 3$	$d = 5$	$d = 7$	$d = 1$	$d = 3$	$d = 5$	$d = 7$
2048	97.6%	99.4%	99.8%	99.9%	98.3%	98.1%	99.6%	99.8%
4096	95.1%	99.0%	99.7%	99.8%	93.5%	96.6%	99.1%	99.6%
8192	83.0%	96.3%	99.0%	99.2%	86.4%	94.0%	98.2%	99.0%
16384	34.4%	72.3%	88.4%	92.6%	56.5%	81.3%	91.1%	93.9%
32768	0.0%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%
average	180,851	269,348	307,520	322,022	226,958	286,676	317,648	324,019
maximum	366,448	479,996	527,172	534,920	383,232	382,388	384,428	385,032

TABLE IV  
REACHING RATIOS, AVERAGE SCORES AND MAXIMUM SCORES OF THE EXPECTIMAX PLAYER (DEPTH  $d = 1, \dots, 4$ )  
BASED ON THE  $N$ -TUPLE NETWORKS WITH  $N = 7$  AND  $m = 4, 8$ .

$N = 7$	$m = 4$				$m = 8$			
	$d = 1$	$d = 3$	$d = 5$	$d = 7$	$d = 1$	$d = 3$	$d = 5$	$d = 7$
2048	98.7%	99.7%	99.8%	99.8%	98.9%	79.8%	84.4%	88.5%
4096	96.6%	99.2%	99.7%	99.7%	94.5%	60.7%	65.5%	71.4%
8192	87.7%	96.9%	98.9%	99.2%	88.6%	55.3%	64.3%	71.0%
16384	37.3%	69.9%	80.9%	86.4%	68.3%	49.9%	60.8%	68.1%
32768	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%	0.3%
average	191,488	247,213	269,962	280,759	255,198	186,346	221,792	247,964
maximum	342,308	385,244	385,824	387,060	384,416	387,260	582,732	629,964

Our future work includes the following three. Firstly, we would like to carefully analyze the results obtained in this work, especially for the not good results in Section VI. Secondly, we would like to apply the extended learning methods, such as multi-stage TD learning [10], [11], to obtain better player. Thirdly, we would like to confirm the applicability of the proposed method for similar games.

*Acknowledgment:* Most of the experiments were conducted on the IACP cluster of the Kochi University of Technology.

#### REFERENCES

- [1] G. Cirulli, "2048," <http://gabrielecirulli.github.io/2048/>, 2014.
- [2] A. Abdelkader, A. Acharya, and P. Dasler, "On the complexity of slide-and-merge games," *arXiv:1501.03837 [cs.CC]*, 2015.
- [3] S. Langerman and Y. Uno, "Threes!, Fives, 1024!, and 2048 are hard," *arXiv:1505.04274 [cs.CC]*, 2015.
- [4] K.-H. Yeh, C.-C. Liang, K.-C. Wu, and I.-C. Wu, "2048-bot tournament in Taiwan," <https://icga.leidenuniv.nl/wp-content/uploads/2015/04/2048-bot-tournament-report-1104.pdf>, 2014.
- [5] W. Jaśkowski and M. Szubert, "Game 2048 AI controller competition GECCO 2015," <http://www.cs.put.poznan.pl/wjaskowski/pub/2015-GECCO-2048-Competition/GECCO-2015-2048-Competition-Results.pdf>, 2015.
- [6] "GPCC (Games and Puzzles Competitions on Computers) problems for 2015," <http://hp.vector.co.jp/authors/VA003988/gpcc/gpcc15.htm>, 2015, (in Japanese).
- [7] "GPCC (Games and Puzzles Competitions on Computers) problems for 2016," <http://hp.vector.co.jp/authors/VA003988/gpcc/gpcc16.htm>, 2015, (in Japanese).
- [8] K. Oka and K. Matsuzaki, "An evaluation function for 2048 players: evaluation for the original game and for the two-player variant," in *Proc. 57th Programming Symposium*, pp. 9–18, 2016 (in Japanese).
- [9] M. Szubert and W. Jaśkowski, "Temporal difference learning of  $N$ -tuple networks for the game 2048," in *Proc. 2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8, 2014.
- [10] K. Yeh, I. Wu, C. Hsueh, C. Chang, C. Liang, and H. Chiang, "Multi-stage temporal difference learning for 2048-like games," *arXiv:1606.07374 [cs.LG]*, 2016.
- [11] I.-C. Wu, K.-H. Yeh, C.-C. Liang, C.-C. Chang, and H. Chiang, "Multi-stage temporal difference learning for 2048," in *Proc. Technologies and Applications of Artificial Intelligence (TAAI 2014)*, pp. 366–378, 2014.
- [12] K. Oka and K. Matsuzaki, "Systematic selection of  $N$ -tuple networks for 2048," in *Proc. 9th International Conference on Computers and Games (CG2016)*, 2016, to appear.
- [13] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.
- [14] A. L. Samuel, "Some studies in machine learning using the game of Checkers," *IBM Journal of Research and Development*, vol. 44, no. 1, pp. 206–227, 1959.
- [15] G. Tesauro, "TD-Gammon, a self-teaching backgammon program, achieves master-level play," *Neural computation*, vol. 6, no. 2, pp. 215–219, 1994.
- [16] M. van der Ree and M. Wiering, "Reinforcement learning in the game of Othello: Learning against a fixed opponent and learning from self-play," in *Proc. IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, pp. 108–115, 2013.
- [17] N. N. Schraudolph, P. Dayan, and T. J. Sejnowski, "Learning to evaluate Go positions via temporal difference methods," in *Proc. Computational Intelligence in Games*, pp. 77–98, 2001.
- [18] 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.
- [19] R. Xiao, "nneonneo/2048-ai," <https://github.com/nneonneo/2048-ai>, 2015.
- [20] P. Rodgers and J. Levine, "An investigation into 2048 AI strategies," in *Proc. 2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–2, 2014.
- [21] K. Oka, K. Matsuzaki, and K. Haraguchi, "Exhaustive analysis and Monte-Carlo tree search player for two-player 2048," *Kochi University of Technology Research Bulletin*, vol. 12, no. 1, pp. 123–130, 2015. (in Japanese)
- [22] 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/Treeco.pdf>, 2015.