Systematic Selection of N-Tuple Networks for 2048

Kazuto Oka and Kiminori Matsuzaki[✉]

Kochi University of Technology, Kami 782–8502, Japan
`195061f@gs.kochi-tech.ac.jp`, `matsuzaki.kiminori@kochi-tech.ac.jp`

Abstract. The puzzle game 2048, a single-player stochastic game played on a 4 × 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) with N-tuple networks, and it matters a great deal how to design N-tuple networks. In this paper, we study the N-tuple networks for the game 2048. In the first set of experiments, we conduct TD learning by selecting 6- and 7-tuples exhaustively, and evaluate the usefulness of those tuples. In the second set of experiments, we conduct TD learning with high-utility tuples, varying the number of tuples. The best player with ten 7-tuples achieves an average score 234,136 and the maximum score 504,660. It is worth noting that this player utilize no game-tree search and plays a move in about $12\,\mu\text{s}$.

1 Introduction

The puzzle game 2048 [4], a single-player stochastic game played on a 4×4 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. The difficulty of the game was discussed from the viewpoint of computational complexity by Abdelkader et al. [2] and Langerman and Uno [6]. As a testbed of artificial intelligence methods, there have been some competitions of computer players for the game 2048 [5,18] and a two-player version of 2048 [1,8].

One of the strongest computer players for 2048 uses temporal difference learning (TD learning for short) with N-tuple networks together with the expectimax algorithm [16]. An N-tuple network consists of a number of N-tuples: each N-tuple covers N cells on the grid and it contributes a number of features each for one distinct occurrence of tiles on the covered cells. Given an N-tuple network, the evaluation function simply calculates the summation of feature weights for all occurring features, where the weights can be obtained through TD learning over a number of self-plays.

In this approach, it matters a great deal how to design (or select) N-tuple networks. The authors of the previous work used hand-designed networks: Wu et al. [16] used a network with four 6-tuples; former work by Szubert and Jaśkowski [14] used a network with two 6-tuples and two 4-tuples. As we can
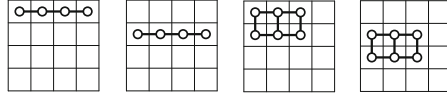
Fig. 1. The N-tuple network (with two 4-tuples and two 6-tuples) by Szubert and Jaśkowski [14]
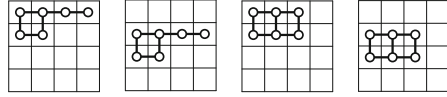

**Fig. 2.** The N-tuple network (with four 6-tuples) by Wu et al. [16]

easily imagine, the more and the larger tuples we use, the higher score we would obtain. The resources such as memory size or computation time, however, limit the available number and/or size of tuples.

In this paper, we study the N-tuple networks for the game 2048. In the first set of experiments, we conduct TD learning exhaustively for 6- and 7-tuples, and evaluate the usefulness of those tuples. By looking closely at the usefulness of those tuples, we find several interesting facts about them. In the second set of experiments, we conduct TD learning with high-utility tuples, varying the number of tuples. We confirm that the more tuples we use the higher score we obtain up to around 20 tuples where the score peaks for the case of 6-tuples.

The main contributions of the paper are summarized as follows.

– A systematic way of selecting N-tuple networks. The way we select N-tuple networks in this paper does not rely on heuristics or human knowledge of the games.
– Comparing usefulness of N-tuples. We evaluate the usefulness of tuples from exhaustive experiments. The results are consistent with the heuristics of the game.
– The best player with ten 7-tuples achieves the average score 234,136 and the maximum score 504,660. It is worth noting that this player does not utilize game-tree search like expectimax and plays a move in about $12\,\mu\text{s}$ (about 88,000 moves per second).

Rules of 2048. The game 2048 is played on a 4×4 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 $10\,\%$). In the initial state, two tiles are put randomly (Fig. 3). The player selects a direction (either of up, left, down, and 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 do not merge again on the same move: moves to
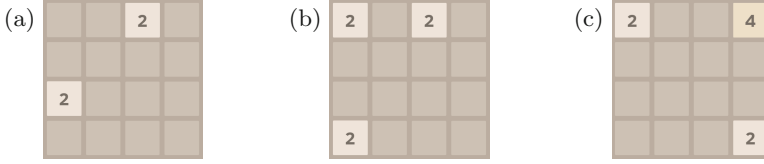
(a) An example of the initial state. Two tiles are put randomly.
(b) After moving up. A new 2-tile appears at the lower-left corner.
(c) After moving right. Two 2-tiles are merged to a 4-tile, and score 4 is given.

Fig. 3. The process of the game 2048

the 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. If the player cannot move the tiles, the game ends.

Paper Overview. The rest of the paper is organized as follows. Section 2 reviews the idea of applying N-tuple networks and TD learning to the game 2048. In Sect. 3, we analyze the usefulness of 6- and 7-tuples by experiments that selects those tuples exhaustively. Based on the analysis of usefulness of tuples, we select a number of high-utility tuples and conduct experiments in Sect. 4. Section 5 discusses related work and Sect. 6 concludes the paper.

2 N-Tuple Networks and Temporal Difference Learning for 2048

In this section, we review the idea of applying N-tuple networks (in Sect. 2.1) and TD learning to the game 2048 (in Sect. 2.2). The algorithm was given by Szubert and Jaśkowski [14] and it was called TD-AFTERSTATE in their paper.

2.1   Evaluation Function with N-Tuple Networks and Playing Algorithm

An N-tuple network consists of a number of N-tuples where each 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 in the network. If each cell in the tuple may have one of K values an N-tuple contributes K^N features, that is, we assign a feature weight for each of K^N features. We use $K = 16$, which means the maximum value of a tile is 32768. (We did not know any player that achieved a 65536 tile, at the time we did the experiments. Recently, Yeh et al. [19] reported their success of a 65536 tile.) Note that 6- and 7-tuples require 64 MB and 1 GB, respectively, under the condition of $K = 16$ and 32 bits for each feature weight.

Given an N-tuple network and corresponding set of feature weights, 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 and all symmetric boards.

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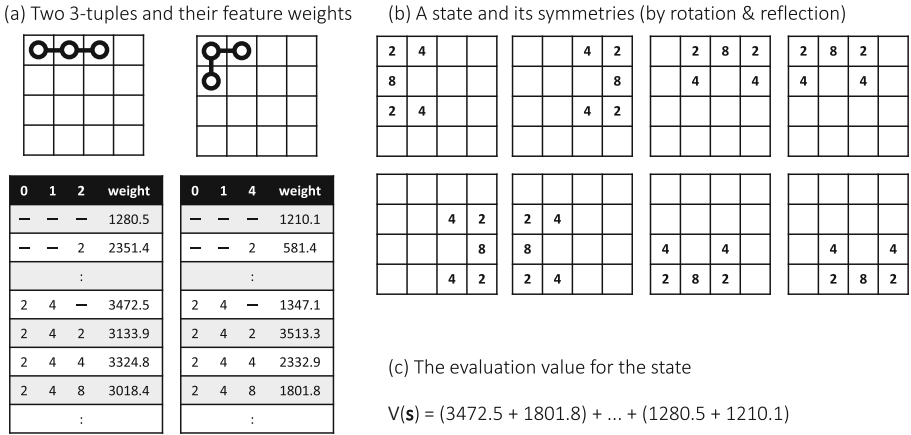
Fig. 4. An example for calculating an evaluation value of a state

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

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

2.2 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 referring to the famous program for checkers [11]. TD learning has been adapted to several games such as backgammon [15], Othello [9], and Go [12].

In our algorithm, the evaluation values are adjusted by TD learning as follows. 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 Δ for the evaluation value is defined as follows.

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

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

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

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

3 Exhaustive Analysis of Usefulness of N-Tuples

Though the game 2048 is a small game, there are still a large number of N-tuples. Table 1 shows the number of all the N-tuples and connected ones. (We count N-tuples that are the same after rotation or reflection once.) In the following, we only consider connected 6- and 7-tuples to keep the number of tuples manageable.

Table 1. Number of N-tuples

N	3	4	5	6	7	8	9	10	11	12	13
All	77	252	567	1051	1465	1674	1465	1051	567	252	77
Connected C_N	8	17	33	68	119	195	261	300	257	169	66

As a first task, we would like to order the N-tuples in terms of their usefulness.

When we form N-tuple networks by randomly selecting tuples and conduct TD learning, the (average) scores differ to a degree. Therefore, we have made the following two assumptions on the usefulness of N-tuples: (1) N-tuples contribute independently from each other; (2) N-tuples contribute linearly. With these assumptions, we consider scores come simply from the sums of partial scores of N-tuples selected in the networks.

Let 6-tuples be indexed from 1 to $C_6 = 68$ and 7-tuples be from 1 to $C_7 = 119$. Let p_i be the partial score of the i-th tuple, and s_{ji} be the 0–1 variable showing that the i-th tuple is selected in the j-th experiment. We assume that the score P_j of the j-th experiment is the sum of partial scores of selected tuples, $P_j = \sum_{i=1}^{C} s_{ji} p_i$. Then, given a set of experimental results with the selected tuples and scores, we can estimate the partial scores by the least squares method: we reduce the squared error $E = \sum_j (P_j - \sum_{i=1}^{C} s_{ji} p_i)^2$.
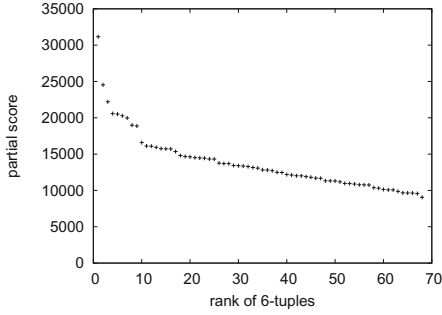
Fig. 5. Partial scores of 6-tuples
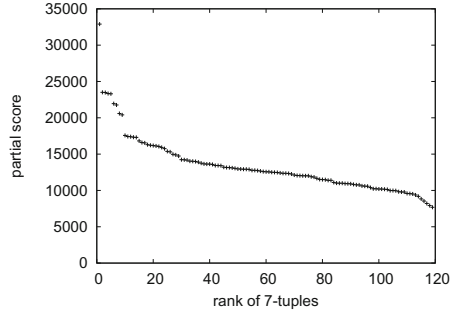
Fig. 6. Partial scores of 7-tuples

Here are the details of the experiments. For each experiment, we randomly selected 10 tuples and executed TD learning with 1,000,000 self-play games. We used the average of the scores of the last 10,000 games as the score of the experiment. We conducted 680 experiments for 6-tuples and 1190 experiments for 7-tuples so that each tuple was selected 100 times on average.

Figures 5 and 6 plot the partial scores in descending order. The medians of partial scores are $M_6 = 13,066$ for 6-tuples and $M_7 = 12,566$ for 7-tuples: those of the best tuples are $31,161 = 2.38M_6$ and $32,900 = 2.61M_7$, respectively; those of the worst tuples are $9,052 = 0.69M_6$ and $7,683 = 0.61M_7$, respectively. (The absolute values of partial scores may not be comparable, since we stop the experiments at 1,000,000 games before the scores saturate.)

Table 2 shows the four best and the four worst tuples for $N = 6$ and $N = 7$. In both cases, the best tuples ⟨6-01⟩ and ⟨7-001⟩ include an edge connecting two adjacent corners and are closely connected. This is reasonable for the slide-and-merge property and keep-large-tile-on-corner heuristics of the game. Seven out of the 8 worst tuples do not include corner cells and it is according to our expectation. The worst 7-tuple ⟨7-119⟩, however, includes two corner cells. We consider the following reason for this: since either of the two diagonal corners is often empty, the tuples with two diagonal corner cells have less information. We will see this again later.

Table 3 shows the results of other interesting tuples. Wu et al. [16] used the four 6-tuples ⟨6-01⟩ and ⟨6-04⟩ in Table 2 and ⟨6-40⟩ and ⟨6-48⟩ in Group 1 of Table 3. Although it is a common technique to make N-tuples by sliding existing (better) N-tuples, those N-tuples are not necessary good ones.

Group 2 of Table 3 shows N-tuples that apparently look good but are not so good. From the keep-large-tile-on-corner heuristics, one may design the 6-tuples ⟨6-24⟩ that covers the cells near a corner, but it is ranked 24th out of 68 and not so good. The case of 7-tuples is very surprising. One may design the 7-tuple ⟨7-096⟩ that has two edges among three corners, but it seems useless. This is more evidence of the reason of the worst 7-tuple. In fact, all tuples that include two diagonal corners (⟨7-119⟩ in Table 2 and ⟨7-096⟩, ⟨7-087⟩, ⟨7-079⟩ in Group 2 of Table 3) seem useless.

Table 2. Four best and four worst 6-tuples and 7-tuples

| *N*	4 best tuples				4 worst tuples			
	⟨6-01⟩	⟨6-02⟩	⟨6-03⟩	⟨6-04⟩	⟨6-65⟩	⟨6-66⟩	⟨6-67⟩	⟨6-68⟩
6	31,161	24,530	22,207	20,576	9,644	9,642	9,563	9,052
	⟨7-001⟩	⟨7-002⟩	⟨7-003⟩	⟨7-004⟩	⟨7-116⟩	⟨7-117⟩	⟨7-118⟩	⟨7-119⟩
7	32,900	23,504	23,483	23,338	8,543	8,204	7,918	7,683

Table 3. Other interesting tuples

group 1		group 2			
⟨6-40⟩	⟨6-48⟩	⟨6-24⟩	⟨7-096⟩	⟨7-087⟩	⟨7-079⟩
12,190	11,330	14,320	10,573	10,988	11,521

group 3					
⟨6-08⟩	⟨7-007⟩	⟨7-012⟩	⟨7-034⟩	⟨7-086⟩	⟨7-096⟩
18,990	21,766	17,394	14,018	10,995	10,573

Group 3 of Table 3 tells an interesting fact. Adding a cell to an existing (good) *N*-tuple is a possible way of generating an *N* + 1-tuple. It usually works well but not in some cases. By adding to a cell to the useful 6-tuple ⟨6-08⟩ we can generate five 7-tuples, among which two are useful but other two are useless. The converse does not hold in some cases either. The 6-tuple ⟨6-08⟩ is the best among those given by removing a cell from ⟨7-007⟩, ⟨6-16⟩ from ⟨7-016⟩, and ⟨6-17⟩ from ⟨7-017⟩. Note that the numbers of 6-tuples and 7-tuples are 68 and 119, respectively.

4   Performance with Respect to Number of *N*-Tuples

In the previous section, we ordered the *N*-tuples by their usefulness. Now we select the first *m* tuples to conduct TD learning with them.

In the second set of experiments, we selected at most 45 6-tuples or at most 10 7-tuples. Since a 6-tuple requires 64 MB and a 7-tuple does 1 GB to store
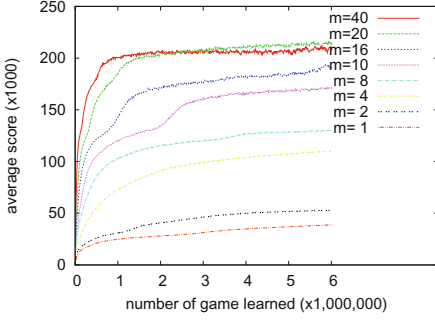
Fig. 7. Average scores with 6-tuples
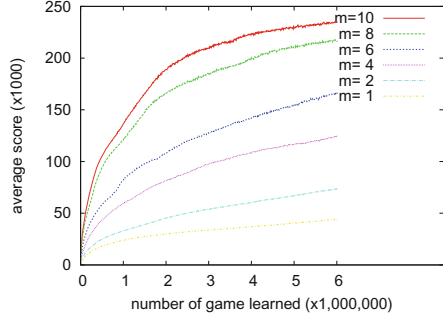
Fig. 8. Average scores with 7-tuples
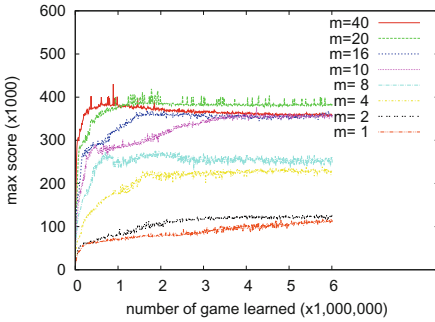
Fig. 9. Maximum scores with 6-tuples
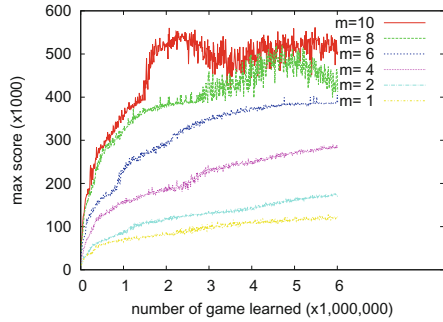
Fig. 10. Maximum scores with 7-tuples

feature weights, the program with $N = 6$ and $m = 45$ consumes about $3\,\mathrm{GB}$ of memory and that with $N = 7$ and $m = 10$ does about $10\,\mathrm{GB}$ of memory. The experiments were conducted on a PC with two Intel Xeon E5645 CPU (6 cores, $2.4\,\mathrm{GHz}$), $12\,\mathrm{GB}$ of memory, with the CentOS 5.5 (kernel 2.6.18–194.e15) and g++ 4.6.3.

For each set of N-tuples, we executed TD learning with 6,000,000 self-play games, and then had additional 10,000 games with the obtained feature weights. During the self-play and learning, we output the summary of the average score and the maximum score once every 10,000 games. For the additional games, in addition to the average and maximum scores, we measured the execution time to calculate the time for selecting a move, and the ratio of reaching 2048, 4096, 8192, 16384 and 32768 tiles. We conducted the experiments five times for each set of N-tuples and all the results (including the maximum score) were averaged among the five experiments.

Figures 7, 8, and 11 plot the average scores with respect to the number of games learned. Figures 9, 10, and 12 plot the maximum scores with respect to the number of games learned.
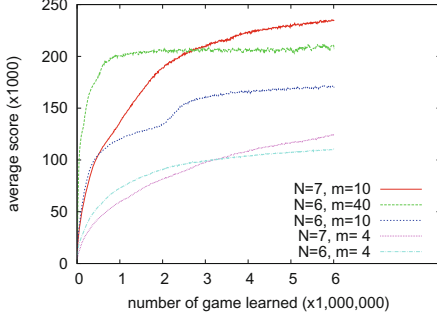
Fig. 11. Comparing average scores with 6- and 7-tuples
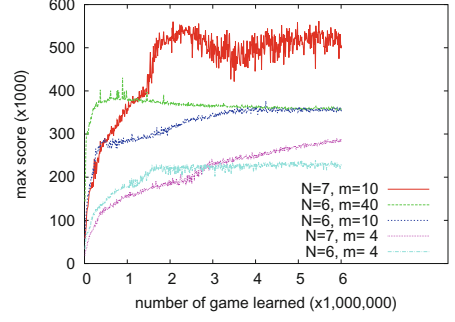
Fig. 12. Comparing maximum scores with 6- and 7-tuples
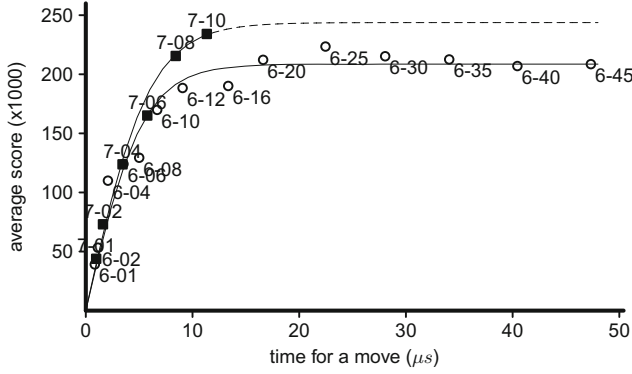
Fig. 13. Score w.r.t. computing time

In general, we confirmed the fact that the more N-tuples we use the higher score we obtain up to a certain number of N-tuples. For the case of $N = 6$, the more N-tuples we use the higher average score we obtain up to around $m = 20$, and the higher maximum score up to around $m = 10$. In terms of the learning speed, the more tuples we use the faster the learning proceeds. In the case of $N = 6$ and $m = 40$, after 1,000,000 games the learning seems to converge. For the case of $N = 7$, the more N-tuples we use the higher score we obtain (for $m \leq 10$).

Comparing $N = 6$ and $N = 7$, we can see that the learning proceeds faster for $N = 6$. This is due to the difference of numbers of features that an N-tuple contributes. For the $N = 7$ cases, 6,000,0000 games seem to be not sufficient to converge. Nonetheless, the results with $N = 7$ and $m = 10$ finally outperform all the results with $N = 6$.

Figure 13 plots the average scores to the time for selecting a move. The time for selecting a move is almost linear in the number m of used N-tuples. For the same number m, the program with 7-tuples took about 1.5 times as much time

Table 4. Ratio when the program achieves 2048, 4096, 8192, 16384, and 32768

	$N = 6$						
tile	$m = 1$	$m = 2$	$m = 4$	$m = 6$	$m = 8$	$m = 10$	$m = 12$
2048	78.41	88.36	95.66	96.00	97.45	97.16	97.78
4096	36.05	62.77	90.32	91.36	93.87	94.05	95.17
8192	0.07	0.33	56.38	70.28	75.67	79.29	84.08
16384	0.00	0.00	0.04	0.07	0.05	29.64	39.02
32768	0.00	0.00	0.00	0.00	0.00	0.00	0.00

	$N = 6$						
tile	$m = 16$	$m = 20$	$m = 25$	$m = 30$	$m = 35$	$m = 40$	$m = 45$
2048	97.87	98.38	98.45	98.42	98.46	98.41	98.30
4096	95.42	96.11	96.25	95.97	95.91	95.89	95.64
8192	85.01	86.51	87.61	86.41	86.14	85.75	86.45
16384	38.96	49.75	56.00	52.61	52.76	50.09	51.42
32768	0.00	0.00	0.00	0.00	0.00	0.00	0.00

	$N = 7$					
tile	$m = 1$	$m = 2$	$m = 4$	$m = 6$	$m = 8$	$m = 10$
2048	82.54	90.74	96.43	97.16	98.27	98.50
4096	40.39	71.75	86.97	89.18	93.59	96.87
8192	0.31	26.08	63.22	73.28	84.87	83.63
16384	0.00	0.00	6.71	27.91	47.17	56.57
32768	0.00	0.00	0.00	0.00	0.00	0.03

as that with 6-tuples did. The program with $N = 7$ and $m = 10$ obtained better results in less time than that with $N = 6$ and $m = 40$ (Weak points are the memory size and the cost of learning process).

In Fig. 13, the curves are the Logistic curves through the origin at the midpoint $f(x) = L\frac{(1-e^{-kx})}{(1+e^{-kx})}$, fitted to the points (the average scores with respect to the computing times). We can see the fact that the average score peaks for the case of $N = 6$. Since the average score does not peak for $N = 7$ up to $m = 10$, we fitted the curve with an additional result for $m = 30$ (the average score 249,625 and the computation time $40\,\mu s$)[1]. These facts suggest that we should combine N-tuples with another game-tree-search technique.

Table 4 shows the ratio when the program successfully achieves tiles 2048, 4096, 8192, 16384 and 32768. Since the program did not use any game-tree-search technique, it failed suddenly with a little probability (missing $1.5\,\%$ for 2048) even with 40 6-tuples or 10 7-tuples. In contrast, with 40 6-tuples or 10 7-tuples, the program succeeded to make 16,384 once for two tries. Furthermore, the program with $N = 7$ and $m = 10$ happened to reach 32,768.

---

[1] Since it requires $30\,GB$ of memory to conduct the experiment, we used a PC with $32\,GB$ memory for this additional experiment.

5   Related Work

Several game-playing algorithms have been adapted to the game 2048 [3,7,10, 14,16,17,20]. Among them, the state-of-the-art algorithm combines expectimax with TD learning or some other heuristics.

The first application of TD learning to the 2048 player was done by Szubert and Jaśkowski [14]. 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. [16] and the extension increased the average score to 142,727. The hand-selected four 6-tuples achieved better score than our systematically selected four 6-tuples (the average score was 109,983). Wu et al. 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 [19].

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 ms [5,18]. Our player with $N = 7$ and $m = 10$ requires a large memory (about 10 GB) but runs much faster (about 12 μs for a move).

6   Conclusion

In this paper we designed experiments, with which we can systematically evaluate the usefulness of N-tuples. In addition to confirming the usefulness of previous hand-selected N-tuples, we found several interesting properties of N-tuples for the game 2048. By selecting the N-tuples from the head of the lists, we can easily obtain N-tuple networks. From the second set of experiments, we confirmed the fact that the more N-tuples we use the higher scores we obtain up to a certain number of tuples where the score peaks. With 10 7-tuples, the program achieved the average score 234,136 and the maximum score 504,660. As far as the authors know, these scores are the highest among the TD learning players (without game-tree-search techniques).

Our future work includes the following two tasks. First, we would like to confirm the performance of our 7-tuples with expectimax search. Second, we would like to propose a better way of selecting N-tuples from the ordered list.

References

1. GPCC (games and puzzles competitions on computers) problems for 2015 (2015, in Japanese). http://hp.vector.co.jp/authors/VA003988/gpcc/gpcc15.htm
2. Abdelkader, A., Acharya, A., Dasler, P.: On the complexity of slide-and-merge games, [cs.CC] (2015). arXiv:1501.03837

3. Chabin, T., Elouafi, M., Carvalho, P., Tonda, A.: Using linear genetic programming to evolve a controller for the game 2048 (2015). http://www.cs.put.poznan.pl/wjaskowski/pub/2015-GECCO-2048-Competition/Treecko.pdf

4. Cirulli, G.: 2048 (2014). http://gabrielecirulli.github.io/2048/

5. Jaśkowski, W., Szubert, M.: Game 2048 AI controller competition @ GECCO 2015 (2015). http://www.cs.put.poznan.pl/wjaskowski/pub/2015-GECCO-2048-Competition/GECCO-2015-2048-Competition-Results.pdf

6. Langerman, S., Uno, Y.: Threes!, fives, 1024!, and 2048 are hard. CoRR abs/1505.04274 (2015)

7. Oka, K., Matsuzaki, K., Haraguchi, K.: Exhaustive analysis and Monte-Carlo tree search player for two-player 2048. Kochi Univ. Technol. Res. Bull. **12**(1), 123–130 (2015, in Japanese)

8. Oka, K., Matsuzaki, K.: An evaluation function for 2048 players: evaluation for the original game and for the two-player variant. In: Proceedings of the 57th Programming Symposium, pp. 9–18 (2016, in Japanese)

9. van der Ree, M., Wiering, M.: Reinforcement learning in the game of Othello: learning against a fixed opponent and learning from self-play. In: IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), pp. 108–115 (2013)

10. Rodgers, P., Levine, J.: An investigation into 2048 AI strategies. In: 2014 IEEE Conference on Computational Intelligence and Games, pp. 1–2 (2014)

11. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM J. Res. Dev. **44**(1), 206–227 (1959)

12. Schraudolph, N.N., Dayan, P., Sejnowski, T.J.: Learning to evaluate go positions via temporal difference methods. In: Computational Intelligence in Games, pp. 77–98 (2001)

13. Sutton, R.S.: Learning to predict by the methods of temporal differences. Mach. Learn. **3**(1), 9–44 (1988)

14. Szubert, M., Jaśkowski, W.: Temporal difference learning of N-tuple networks for the game 2048. In: 2014 IEEE Conference on Computational Intelligence and Games, pp. 1–8 (2014)

15. Tesauro, G.: TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural Comput. **6**(2), 215–219 (1994)

16. Wu, I.C., Yeh, K.H., Liang, C.C., Chang, C.C., Chiang, H.: Multi-stage temporal difference learning for 2048. In: Cheng, S.-M., Day, M.-Y. (eds.) Technologies and Applications of Artificial Intelligence. LNCS, vol. 8916, pp. 366–378. Springer, Cham (2014). doi:10.1007/978-3-319-13987-6_34

17. Xiao, R.: nneonneo/2048-ai (2015). https://github.com/nneonneo/2048-ai

18. Yeh, K.H., Liang, C.C., Wu, K.C., Wu, I.C.: 2048-bot tournament in Taiwan (2014). https://icga.leidenuniv.nl/wp-content/uploads/2015/04/2048-bot-tournament-report-1104.pdf

19. Yeh, K.H., Wu, I.C., Hsueh, C.H., Chang, C.C., Liang, C.C., Chiang, H.: Multi-stage temporal difference learning for 2048-like games, [cs.LG] (2016). arXiv:1606.07374

20. Zaky, A.: Minimax and expectimax algorithm to solve 2048 (2014). http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2013-2014-genap/Makalah2014/MakalahIF2211-2014-037.pdf