SYSTEMATIC N-TUPLE NETWORKS FOR OTHELLO POSITION EVALUATION

Wojciech Jaśkowski[1]

Institute of Computing Science, Poznan University of Technology, Poznań, Poland

Abstract

N-tuple networks have been successfully used as position evaluation functions for board games such as Othello or Connect Four. The effectiveness of such networks depends on their architecture, which is determined by the placement of constituent n-tuples (sequences of board locations) providing input to the network. The most popular method of placing n-tuples consists in randomly generating a small number of long, snake-shaped board location sequences. In this paper, we show that learning n-tuple networks is more effective if it involves, instead, a large number of systematically placed, short, straight n-tuples. In addition, we demonstrate that a simple variant of coevolutionary learning can evolve a systematic n-tuple network with tuples of size just 2 of a comparable performance to the best 1-ply Othello players. Our network consists of only 288 parameters, which is an order of magnitude less than the top published players to date. This indicates a need for more effective learning methods that would be capable of taking a full advantage of larger networks.

Keywords: Othello, Reversi, evolution strategy, n-tuple networks, tabular value functions, strategy representation, board features, agent policy

Preprint. Published in ICGA Journal. Cite as:

Wojciech Jaśkowski, Systematic N-Tuple Networks for Othello Position Evaluation ICGA Journal 37(2), 2014, pp. 85–96

1. INTRODUCTION

Board games have always attracted attention in AI due to their clear rules, mathematical elegance and simplicity. Since the early works of Shannon (1950) on Chess and Samuel (1959) on Checkers, a lot of research have been conducted in the area of board games towards finding either perfect players (e.g., Schaeffer (2007) for Checkers), or stronger than human players (e.g., Buro, 2000 for Othello). The bottom line is that board games still constitute valuable test-beds for improving artificial and computational intelligence game playing methods such as reinforcement learning, Monte Carlo tree search, branch and bound, and (co)evolutionary algorithms.

Most of these techniques employ a position evaluation function to quantify the value of a given game state. In the context of Othello, one of the most successful position evaluation functions is *tabular value function* (Buro, 1997) or *n-tuple network* (Lucas, 2008). It consists of a number of n-tuples, each associated with a look up table, which maps contents of n board fields into a real value. The effectiveness of n-tuple network highly depends on the placement of n-tuples (Szubert, Jaśkowski, and Krawiec, 2013). Typically, n-tuples architectures consist of a small number of long, randomly generated, snake-shaped n-tuples (Manning and Othello, 2010; Szubert *et al.*, 2013; Runarsson and Lucas, 2014).

In this paper, we propose an n-tuple network architecture consisting of a large number of short, straight n-tuples, generated in a systematic way. In the extensive computational experiments, we investigate the effectiveness of our architecture in the context of coevolutionary learning of position evaluation functions for Othello. We compare our architecture with the one involving randomly generated n-tuples. Finally, we evaluate the performance of the best evolved n-tuple network against the top 1-ply Othello players from the literature.

[1] email:wjaskowski@cs.put.poznan.pl

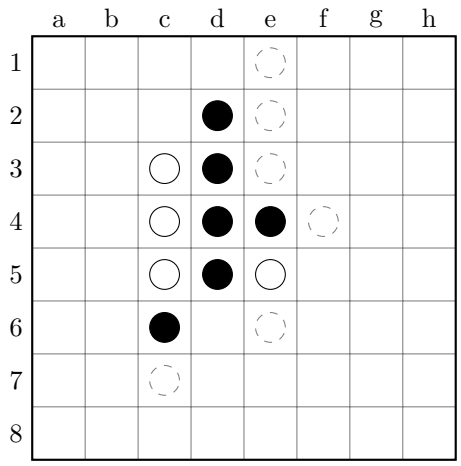


Figure 1: An Othello position, where white has 6 legal moves (dashed gray circles). If white places a piece on e3, the pieces on d3, d4, and e4 are reversed to white.

Despite the importance of the choice of the evaluation function and n-tuple network architecture in particular, to the best of our knowledge, this is the first study that evaluates different ways of placing n-tuples on the Othello board. Therefore, to concentrate only on this issue, instead of trying to design a complete state-of-the-art Othello player that uses advanced game-tree search techniques and opening databases, here we study only position evaluation functions, putting aside other significant elements of a successful computer players (Runarsson and Lucas, 2014).

2. METHODS

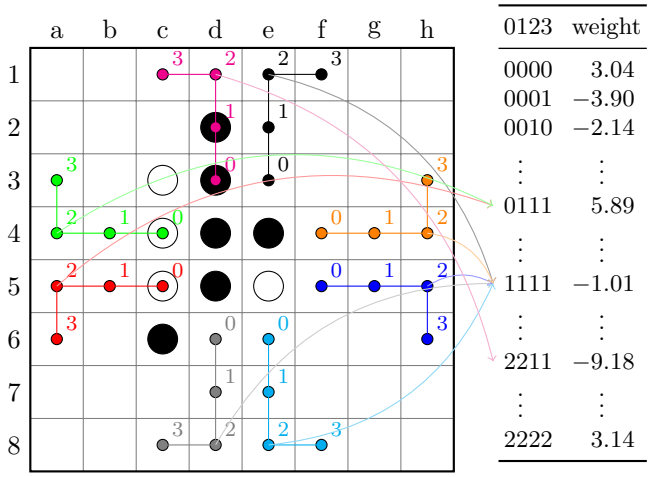
2.1 Othello

Othello (a.k.a. Reversi) is a two player, deterministic, perfect information strategic game played on an 8×8 board. There are 64 pieces being black on one side and white on the other. The game starts with two white and two black pieces forming an askew cross in the center on the board. The players take turns putting one piece on the board with their color facing up. A legal move consists in placing a piece on a field so that it forms a vertical, horizontal, or diagonal line with another player's piece, with a continuous, non-empty sequence of opponent's pieces in between (see Fig. 1), which are reversed after the piece is placed. Player passes if and only if it cannot make a legal move. The game ends when both players passed consecutively. Then, the player having more pieces with their color facing up wins.

Othello has been found to have around 10^{28} legal positions (Allis, 1994) and has not been solved; this is one of the reasons why it has become such a popular domain for computational intelligence methods (Lucas, 2007; Osaki *et al.*, 2008; Manning, 2010b; Chong *et al.*, 2012; van den Dries and Wiering, 2012; Manning and Othello, 2010; Samothrakis *et al.*, 2013; Szubert *et al.*, 2013; Jaśkowski, Szubert, and Liskowski, 2014).

2.2 Position Evaluation Function

In this paper, our goal is not to design state-of-the-art Othello players, but to evaluate different position evaluation functions. This is why our players are simple state evaluators in a 1-ply setup (Szubert *et al.*, 2013; Samothrakis *et al.*, 2013; Runarsson and Lucas, 2014): given the current state of the board, a player generates all legal moves and applies the position evaluation function to the resulting states. The state gauged as the most desirable determines the move to be played. Ties are resolved at random.



0123	weight
0000	3.04
0001	−3.90
0010	−2.14
⋮	⋮
0111	5.89
⋮	⋮
1111	−1.01
⋮	⋮
2211	−9.18
⋮	⋮
2222	3.14

Figure 2: An 4-tuple employed eight times to take advantage of board symmetries (symmetric sampling). The eight symmetric expansions of the 4-tuple return, in total, $5 \times -1.01 + 2 \times 5.89 - 9.18 = -2.45$ for the given board state.

2.3 N-tuple Network

The best performing evaluation function in the Othello League is *n-tuple network* (Samothrakis *et al.*, 2013). N-tuple networks have been first applied to optical character recognition problem by Bledsoe and Browning (1959). For games, it have been used first by Buro (1997) under the name of *tabular value functions*, and later popularized by Lucas (2008). According to Szubert *et al.* (2013) main advantages of n-tuple networks "include conceptual simplicity, speed of operation, and capability of realizing nonlinear mappings to spaces of higher dimensionality".

N-tuple network consists of m n_i-tuples, where n_i is tuple's size. For a given board position $\mathbf{b}$, it returns the sum of values returned by the individual n-tuples. The ith n_i-tuple, for $i = 1 \ldots m$, consists of a predetermined sequence of board locations $(loc_{ij})_{j=1 \ldots n_i}$, and a look up table LUT_i. The latter contains values for each board pattern that can be observed on the sequence of board locations. Thus, n-tuple network is a function

$$f(\mathbf{b}) = \sum_{i=1}^{m} f_i(\mathbf{b}) = \sum_{i=1}^{m} \mathrm{LUT}_i \left[\mathrm{index} \left(b_{loc_{i1}}, \ldots, b_{loc_{in_i}} \right) \right].$$

Among possible ways to map the sequence to an index in the look up table, the following one is arguably convenient and computationally efficient:

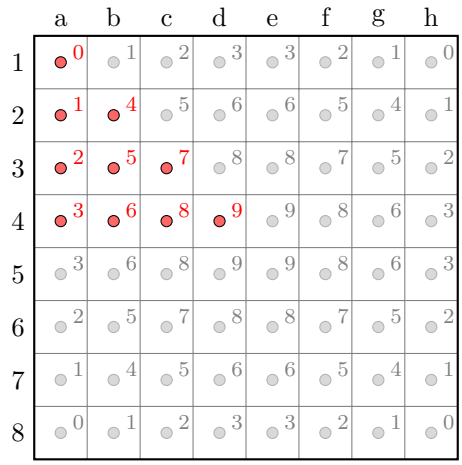
$$\mathrm{index}(\mathbf{v}) = \sum_{j=1}^{|\mathbf{v}|} v_j c^{j-1},$$

where c is a constant denoting the number of possible values on a single board square, and $\mathbf{v}$ is the sequence of board values (the observed pattern) such that $0 \leq v_j < c$ for $j = 1 \ldots |\mathbf{v}|$. In the case of Othello, $c = 3$, and white, empty, and black squares are encoded as 0, 1, and 2, respectively.

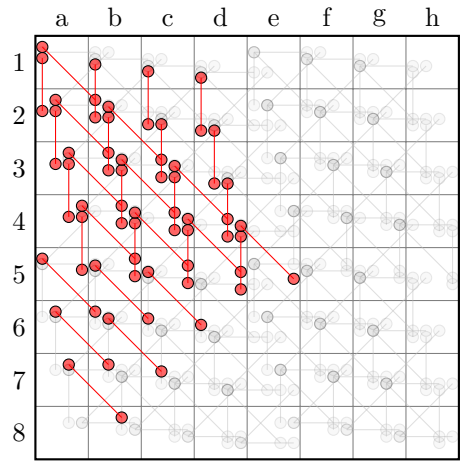
The effectiveness of n-tuple networks is improved by using *symmetric sampling*, which exploits the inherent symmetries of the Othello board (Lucas, 2007). In symmetric sampling, a single n-tuple is employed eight times, returning one value for each possible board rotation and reflection. See Fig. 2 for an illustration.

2.4 N-tuple Network Architecture

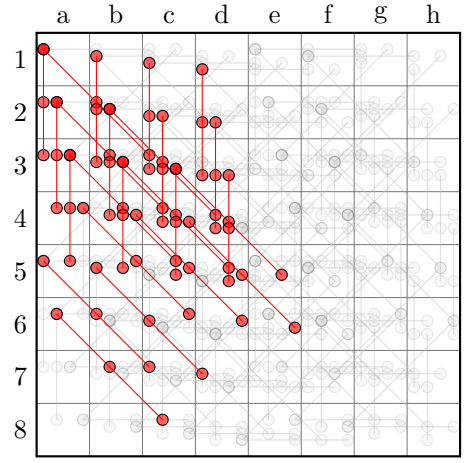
Due to the spatial nature of game boards, n-tuples are usually consecutive snake-shaped sequences of locations, although this is not a formal requirement. If each n-tuple in a network is of the same size, we denote it as $m \times n$-tuple network, having $m \times 3^n$ weights. Apart from choosing n and m, an important design issue of n-tuples network architecture is the location of individual n-tuples on the board (Szubert *et al.*, 2013).



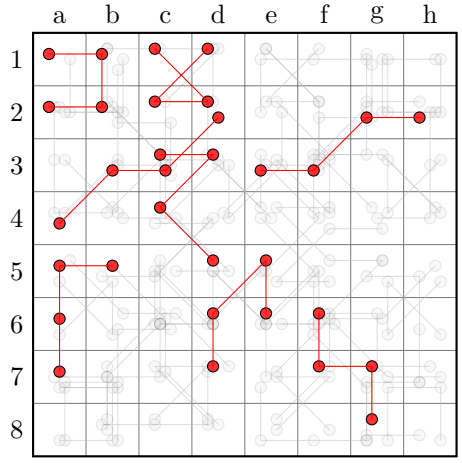
(a) all-1 network consisting of 10 1-tuples (30 weights). The numbers denote symmetric expansions corresponding to the "main" 1-tuples.



(b) all-2 network consisting of all 32 straight 2-tuples (288 weights).



(c) all-3 network consisting of all 24 straight 3-tuples (648 weights).



(d) rand-8×4 network consisting of 8 randomly generated snake-shaped 4-tuples (648 weights).

Figure 3: Rand-* and all-* n-tuple network architectures. "Main" n-tuples have been shown by red, while their symmetric expansions by light gray.

2.4.1 Random Snake-shaped N-tuple Network

In the light of the importance of a proper features selection, it is surprising that many investigations in game strategy learning have involved *randomly generated snake-shaped n-tuple networks*. Lucas (2008) generated individual n-tuples by starting from a random board location, then taking a random walk of 6 steps in any of the eight orthogonal or diagonal directions. The repeated locations were ignored, thus the resulting n-tuples were from 2 to 6 squares long. The same method Krawiec and Szubert (2011) used for generating 7×4, 9×5 and 12×6-tuple networks, and Thill, Koch, and Konen (2012) for generating 70×8 tuple networks playing Connect Four.

An $m \times n$-tuple network generated in this way will be denoted as *rand-*$m \times n$ (see Fig. 3d for an example).

2.4.2 Systematic Straight N-tuple Network

Alternatively, we propose a deterministic method of constructing n-tuple networks. Our *systematic straight n-tuple networks* consist of all possible vertical, horizontal or diagonal n-tuples placed on the board. Its smallest

architecture	weights	architecture	weights
all-2 (32×2)	288	rand-10×3	270
all-3 (24×3)	648	rand-8×4	648
all-4 (21×4)	1701	rand-7×5	1701

Table 1: The number of weights for three pairs of *systematic straight* (all-*) and *random snake-shaped* (rand-*) n-tuple networks architectures.

representative is a network of 1-tuples (see Fig. 3a). Thanks to symmetric sampling, only 10 of them is required for an 8×8 Othello board, and such 10×1-tuple network, which we denote as all-1, involves $3 \times 10^1 = 30$ weights. Fig. 3b and Fig. 3c show all-2 and all-3 networks, respectively. Table 1 contains the number of weights in selected architectures of rand-* and all-* networks.

2.4.3 Other Approaches

Logistello (Buro, 2000), computer player, which beat the human Othello world champion in 1997, used 11 n-tuples of $n \in \{3, 10\}$, hand-crafted by an expert. External knowledge has also been used by Manning and Othello (2010), who generated a diverse 12×6-tuple network using random inputs method from Breiman's Random Forests basing on a set of 10 000 labeled random games.

2.5 Learning to Play Both Sides

When a single player defined by its evaluation function is meant to play both as black and white, it must interpret the result of the evaluation function complementary depending on the color it plays. There are three methods serving this purpose.

The first one is *double function* (e.g., Thill *et al.*, 2012), which simply employs two separate functions: one for playing white and the other for playing black. It allows to fully separate the strategy for white and black players. However, its disadvantage consists in that two times more weights must be learned, and the experience learned when playing black is not used when playing white and vice versa.

Output negation and *board inversion* (e.g., Manning, 2010b; Runarsson and Lucas, 2014) are alternatives to double function. They use only single set of weights, reducing the search space and allowing to transfer the experience between the white and black player. When using output negation, black selects the move leading to a position with the maximal value of the evaluation function whereas white selects the move leading to a position with the minimal value.

Player using *board inversion* learns only to play black. As the best black move it selects the one leading to the position of the maximum value. If it has to play white, it temporarily flips all the pieces on the board interpreting the board as if it played black. Then it selects the best 'black' move, flips all the pieces back, and plays the white piece in the selected location.

3. EXPERIMENTS AND RESULTS

3.1 Common Experimental Settings

3.1.1 Othello Interaction

Othello is a deterministic game and here we study only deterministic players (with an exception of rare situations when at least two considered positions have the same evaluation value). Therefore, following previous research (Lucas and Runarsson, 2006; Samothrakis *et al.*, 2013; Szubert *et al.*, 2013), to provide more continuous performance measure, we introduce more variability to Othello interactions by forcing both players to make random moves with the probability $\epsilon = 0.1$. As a consequence the players no longer play (deterministic) Othello, but

stochastic ϵ-Othello. However, the ability to play ϵ-Othello is highly correlated with the ability to play the original Othello (Lucas and Runarsson, 2006).

In a single game, we assume that the winner scores 1 point, the loser 0 and, in case of a draw, players score 0.5 points each. Since the game is asymmetrical, an interaction of two players in this study involves a *double game*, where both individuals play one game as black and one game as white player.

3.1.2 Coevolutionary Learning

In order to compare different n-tuple network architectures, we performed several computational experiments involving a simple variant of coevolutionary learning (Axelrod, 1987). The algorithm maintains a single population of n-tuple networks, which are evolved for 10000 generations by $(10 + 90)$ Evolution Strategy (Beyer and Schwefel, 2002) using Gaussian mutation with $\sigma = 1.0$. The weights of individuals in the initial population are drawn from the $[-0.1, 0.1]$ interval. Individual's fitness is calculated as an average score in a round-robin tournament played by the population members, where each match between two individuals consists of 10 double games.

3.1.3 Performance Measure

In order to objectively measure the performance of the players obtained in the experiments, we used its average score obtained in 1000-double-games-match against a pool of 13 state-of-the art 1-ply players. The pool consists of networks of two different evaluation functions: either weighted piece counter (WPC) or n-tuple networks (NTN):

- SWH (WPC) hand-crafted by Yoshioka, Ishii, and Ito (1999),
- LR06 (WPC) co-evolved by Lucas and Runarsson (2006),
- SJK09 (WPC, Szubert, Jaśkowski, and Krawiec, 2009), SJK11 (WPC, Szubert, Jaśkowski, and Krawiec, 2011), SJK13_CTDL (NTN, Szubert *et al.*, 2013) obtained using coevolutionary temporal difference learning,
- SJK13_ETDL (NTN, Szubert *et al.*, 2013) learned using evolutionary temporal difference learning,
- EM10_GECCO, EM10_Nash70 (Manning, 2010a), EM10_TCIAG1 and EM10_TCIAG2 (Manning, 2010b) obtained using resource-limited Nash memory, which involved both coevolution and temporal difference learning (all NTN),
- RL14_iPrefN and RL14_iPref1 (NTN) computed using preference learning by Runarsson and Lucas (2014), andPB11_ETDL (NTN) obtained by a combination of evolution and temporal difference learning by Burrow (2011).

Note that the performance measure was used only externally to monitor the training progress and not exploited in any way to drive the (coevolutionary) learning.

3.1.4 Statistical Analysis

We repeated each coevolutionary run 10 times. Every 100 generations, we measured the performance of the fittest individual in the population. The performance of the fittest individual from the last generation is identified with method's performance. For statistical analysis in all of the experiments, we used non-parametric Wilcoxon rank sum test (a.k.a. the Mann-Whitney U test) with the significance level $\alpha = 0.05$.

3.2 Preliminary: Board Inversion vs. Output Negation

Figure 4 presents the results of learning with board inversion and output negation for representatives of two types of n-tuple networks architectures: rand-8×4 having $8 \times 4^3 = 648$, and all-1 with $10 \times 3^1 = 30$ weights.

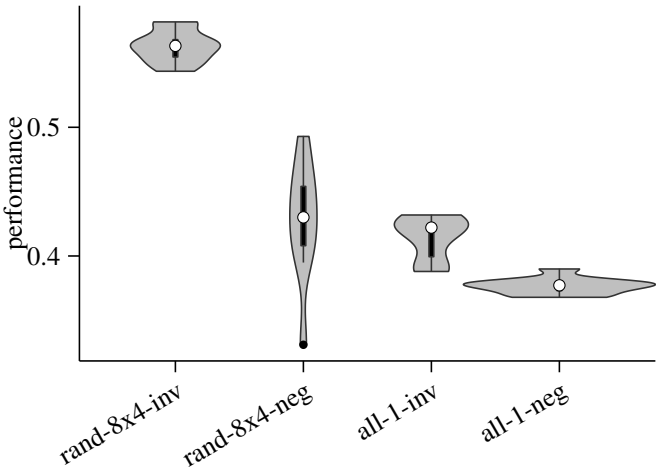


Figure 4: The comparison of output negation against board inversion for two n-tuples architectures. The performance is measured as the average score obtained in 1000 double games against a pool of state-of-the art 1-ply players $\epsilon = 0.1$. In each violin shape, the white dot marks the median, the black boxes range from the lower to the upper quartile, while the thin black lines represent 1.5 interquartile range. Outliers beyond this range are denoted by black dots. The outer shape shows the probability density of the data.

The figure shows that board inversion surpasses output negation regardless of the player architecture, which confirms the findings of Runarsson and Lucas (2014) in the context of preference learning. The differences between the methods are statistically significant (see also the detailed results in Table 3 in Appendix).

Therefore, in the following experiments we employ exclusively board inversion.

3.3 All Short Straight vs. Random Long Snake-shaped N-tuples

In the main experiment, we compare n-tuple networks consisting of all possible short straight n-tuples (all-2, all-3, and all-4) with the long random snake-shaped ones (rand-10×3, rand-8×4, and rand-7×5). We chose the number of n-tuples and their sizes in such a way as to make the number of weights in corresponding architectures equal, or, if impossible, similar at least (see Table 1).

The results of the experiment are shown in Fig. 5 as violin plots. A statistical analysis of the three corresponding pairs of networks reveals that:

- all-2 is better than rand-10×3 (p-value = 0.019),
- all-3 is better than rand-8×4 (p-value = 0.001), and
- no difference between all-4 and rand-7×5 can be claimed.

Although, we found no difference between the networks with the highest number of parameters, for the other two compared pairs, the networks consisting of a large number of short, systematically placed n-tuples (all-*) are shown to work better than the ones consisting of a small number of random long snake-shaped ones (rand-*).

Those findings are further confirmed in Fig. 6, which shows the pace of learning for each of the six analyzed architectures. The figure plots methods' average performance as a function of number of generations, which is proportional to the computational effort. We can see that all-2 and all-3 exhibit higher average performance throughout the learning process than their random counterparts. All-4, on the other hand, learns slower than rand-7×5, but the performance gap between them slowly decreases, which suggests that all-4 could overtake its random counterpart if it is allowed to learn beyond 10000 generations.

Visual inspection of the violin plots reveals that all-* architectures are also generally slightly more robust (except all-3 vs. rand-4×8) due to lower variances than rand-* architectures (cf. Fig. 5). This is because the variance of

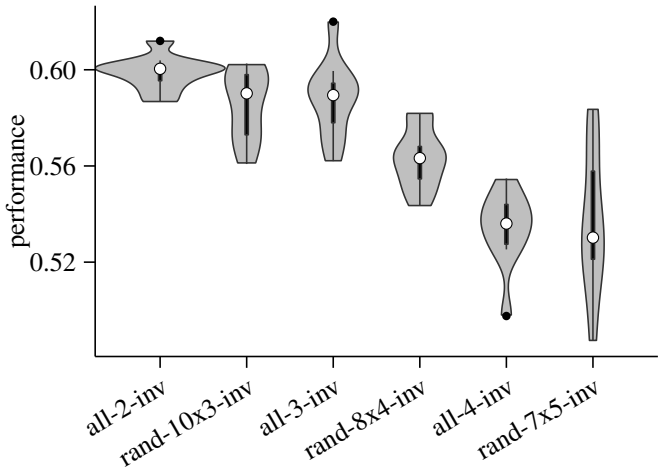


Figure 5: The comparison of all short straight n-tuple networks (all-*) with random long snake-shaped n-tuple networks (rand-*). The distribution of performances is presented as violin plots (see Fig. 4 for explanation).

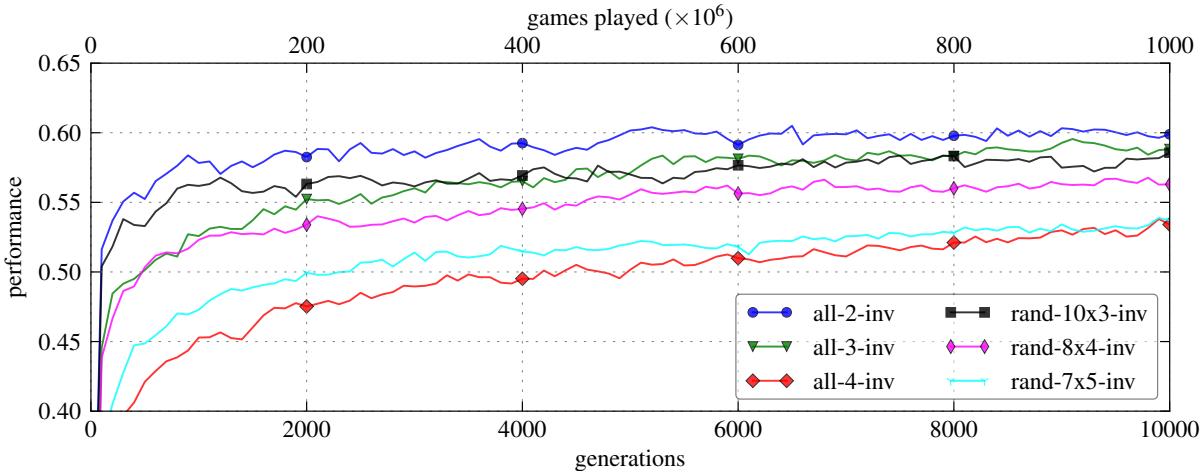


Figure 6: Pace of learning of six analyzed n-tuple networks architectures. Each point on the plot denotes the average performance of method's fittest individual in a given generation.

rand-* architectures is attributed to both its random initialization and non-deterministic learning process, while the variance of all-* is only due to the latter.

3.4 Coevolutionary Learning Works Best for 2-Tuple Networks

Longer n-tuples should, in principle, lead to higher network's performance, since they can 'react' to patterns that the shorter ones cannot. However, the results presented in Fig. 5 show it is not the case for coevolutionary learning. Despite having two times more weights, all-3 does not provide better performance than all-2 (no statistical difference). Furthermore, all-4 is significantly worse than both all-2 and all-3.

Figure 6 suggests that all-2 is not only the best in the long run, but it is also the method that learns fastest. Note, however, that all-3 catches up all-2 eventually. Observe also that all-4 learns even slower than all-3 and although the gap between all-3 and all-4 decreases over time, it is still large after 10000 generations.

Thus, we demonstrated that for Othello, all-2 with just 288 weights, the smallest among the six considered n-tuple network architectures, is also the best of them, for the employed coevolutionary setup, at least. We must, however, note that, e.g., temporal difference learning or coevolutionary temporal difference learning scale better than the purely evolutionary methods (Szubert *et al.*, 2013), thus we are far from claiming that the all-2 network is better *in general* than all-n, where $n > 2$.

weights	player	RL14_iPrefN	all-2	SJK13_CTDL	PB11_ETDL	EM10_GECCO	SJK13_ETDL	EM10_TCIAG1	EM10_Nash70	EM10_TCIAG2	SWH	LR06	SJK11	SJK09	RL14_iPref1	**Total**
6561	RL14_iPrefN	-	**54.8**	54.7	57.2	**54.9**	59.5	**61.8**	65.8	64.9	63.4	78.1	79.0	77.0	84.0	**65.8**
288	all-2	45.3	-	**60.2**	53.4	51.6	**62.2**	58.7	**68.8**	67.9	52.6	81.6	83.9	80.0	**84.0**	65.4
4698	SJK13_CTDL	45.4	39.8	-	**58.2**	51.7	58.9	55.8	58.9	63.1	67.7	**86.8**	**85.7**	**85.6**	83.2	64.7
6561	PB11_ETDL	42.8	46.6	41.8	-	46.8	56.3	56.1	57.8	58.5	**84.8**	83.2	83.3	84.0	81.5	63.3
8748	EM10_GECCO	45.2	48.4	48.3	53.2	-	61.1	60.5	62.1	**68.7**	61.8	76.9	74.2	77.5	79.7	62.9
3240	SJK13_ETDL	40.5	37.8	41.1	43.7	38.9	-	48.9	54.5	55.2	84.1	73.3	73.3	77.5	76.0	57.3
8748	EM10_TCIAG1	38.2	41.3	44.2	44.0	39.6	51.1	-	51.0	56.8	63.3	77.7	80.3	78.3	78.3	57.2
8748	EM10_Nash70	34.2	31.2	41.1	42.2	37.9	45.5	49.0	-	53.8	73.0	76.9	77.1	76.8	75.3	54.9
8748	EM10_TCIAG2	35.1	32.2	36.9	41.5	31.3	44.8	43.3	46.2	-	67.7	74.2	72.3	73.5	72.4	51.6
64	SWH	36.6	47.4	32.3	15.2	38.2	16.0	36.7	27.0	32.3	-	46.1	52.3	54.6	77.6	39.4
64	LR06	22.0	18.4	13.2	16.9	23.2	26.7	22.3	23.2	25.8	53.9	-	49.2	53.7	57.7	31.2
64	SJK11	21.0	16.2	14.3	16.7	25.9	26.7	19.8	22.9	27.8	47.7	50.8	-	49.0	57.5	30.5
64	SJK09	23.1	20.0	14.5	16.1	22.5	22.6	21.7	23.2	26.6	45.4	46.3	51.0	-	57.5	30.0
192	RL14_iPref1	16.0	16.0	16.8	18.5	20.3	24.0	21.7	24.8	27.6	22.4	42.3	42.5	42.5	-	25.8

Table 2: The results of a round robin tournament between Othello 1-ply players: 13 published ones and the all-2 player introduced in this study. Each value is a percent of maximum score possible to obtain in a 1000-double-games-match of ϵ-Othello, where $\epsilon = 0.1$. Best scores against each player have been marked bold.

3.5 Comparison with State-of-the-art Players

In order to see what is the limit of the performance of a 288-weight player we co-evolved all-2 architecture with a larger population of size 600. The performance of the best agent obtained in this way is 65.4%. Table 2 shows the round-robin tournament results between our player and the 13 published players used for the performance measure. all-2 player placed second in this tournament losing only to RL14_iPrefN obtained by Runarsson and Lucas (2014). Let us note, however, that the differences between the five best players are small and their ranking might change if new players are added to the tournament.

Nevertheless, basing on the tournament results, we can claim that all-2 is one of the best players in 1-ply Othello[2].

It is important to notice that our player is over a dozen times ‘smaller‘ than the other top players in terms of the number of parameters used (cf. column ‘weights’ in the table). This suggests that there is a lot of untapped potential in the larger networks to improve.

4. CONCLUSIONS

In this paper, we analyzed different n-tuple network architectures for Othello position evaluation function. Our main contribution is showing that a network consisting of all possible, systematically generated, short n-tuples leads to a better play than long random snake-shaped tuples originally used by Lucas (2007) provided the same number of weights. Such an important task as feature selection should apparently not be left to a random process.

The results of our experiments also show that tuples longer than 2 give no advantage for (co)evolutionary learning, causing slower learning rate, at the same time.

Finally, we demonstrated the effectiveness of systematic n-tuple networks by coevolving a small network consisting of all possible straight 2-tuples, which has a comparable performance to the top 1-ply Othello players from the literature. Importantly, our network involves only 288 weights, while the number of parameters of the other competitors is in the range of $[3240, 8748]$. Since larger networks have, in principle, larger potential, our results

[2]The player is available at `http://www.cs.put.poznan.pl/wjaskowski/othello-league-players`

suggest that the learning algorithms used to learn those networks were unable to take full advantage of it. Thus there is a need for better self-learning algorithms.

The computationally intensive experiments we performed involved co-evolutionary learning. It remains to be seen whether our findings hold for different experimental settings. The interesting open questions are: i) whether our systematic short networks are also advantageous when using other self-learning methods such as temporal difference learning, and ii) whether such networks are also profitable for other board games, e.g., Connect Four or Checkers.

ACKNOWLEDGMENT

This work has been supported by the Polish National Science Centre grant no. DEC-2013/09/D/ST6/03932. The computations have been performed in Poznań Supercomputing and Networking Center. The author would like to thank Marcin Szubert for his helpful remarks on an earlier version of this article and Paweł Liskowski for contributing to the software used for running the computational experiments.

References

Allis, V. L. (1994). *Searching for solutions in games and artificial intelligence*. Ph.D. thesis, University of Limburg, Maastricht, The Netherlands.

Axelrod, R. (1987). The Evolution of Strategies in the Iterated Prisoner's Dilemma. *Genetic Algorithms in Simulated Annealing* (ed. L. Davis), pp. 32–41. Pitman, London.

Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies–A comprehensive introduction. *Natural computing*, Vol. 1, No. 1, pp. 3–52.

Bledsoe, W. W. and Browning, I. (1959). Pattern recognition and reading by machine. *Proc. Eastern Joint Comput. Conf.*, pp. 225–232.

Buro, M. (1997). An evaluation function for othello based on statistics. Technical report, NEC, Princeton, NJ, NECI 31.

Buro, M. (2000). Experiments with Multi-ProbCut and a new high-quality evaluation function for Othello. *Games in AI Research* (eds. H. van den Herik and H. Iida), pp. 77–96. University Maastricht. ISBN 90–621–6416–1.

Burrow, P. (2011). *Hybridising evolution and temporal difference learning*. Ph.D. thesis, University of Essex.

Chong, S. Y., Tino, P., Ku, D. C., and Xin, Y. (2012). Improving Generalization Performance in Co-Evolutionary Learning. *IEEE Transactions on Evolutionary Computation*, Vol. 16, No. 1, pp. 70–85.

Dries, S. van den and Wiering, M. A. (2012). Neural-Fitted TD-Leaf Learning for Playing Othello With Structured Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 23, No. 11, pp. 1701–1713. ISSN 2162–237X.

Jaśkowski, W., Szubert, M., and Liskowski, P. (2014). Multi-Criteria Comparison of Coevolution and Temporal Difference Learning on Othello. *EvoApplications 2014* (eds. A. I. Esparcia-Alcazar and A. M. Mora), Vol. 8602 of *Lecture Notes in Computer Science*, pp. 301–312, Springer.

Krawiec, K. and Szubert, M. G. (2011). Learning n-tuple networks for Othello by coevolutionary gradient search. *Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11*, pp. 355–362, ACM Press, New York, New York, USA. ISBN 9781450305570.

Lucas, S. M. (2007). Learning to play Othello with N-tuple systems. *Australian Journal of Intelligent Information Processing Systems, Special Issue on Game Technology*, Vol. 9, No. 4, pp. 01–20.

Lucas, S. M. and Runarsson, T. P. (2006). Temporal difference learning versus co-evolution for acquiring othello position evaluation. *IEEE Symposium on Computational Intelligence and Games*, pp. 52–59.

Lucas, S. (2008). Learning to play Othello with n-tuple systems. *Australian Journal of Intelligent Information Processing*, Vol. 4, pp. 1–20.

Manning, E. P. (2010a). Coevolution in a large search space using resource-limited nash memory. *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 999–1006, ACM.

Manning, E. P. (2010b). Using Resource-Limited Nash Memory to Improve an Othello Evaluation Function. *Computational Intelligence and AI in Games, IEEE Transactions on*, Vol. 2, No. 1, pp. 40 –53. ISSN 1943–068X.

Manning, E. P. and Othello, A. (2010). Using Resource-Limited Nash Memory to Improve an Othello Evaluation Function. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 1, pp. 40–53.

Osaki, Y., Shibahara, K., Tajima, Y., and Kotani, Y. (2008). An Othello evaluation function based on Temporal Difference Learning using probability of winning. *IEEE Symposium On Computational Intelligence and Games*, pp. 205–211, IEEE. ISBN 978–1–4244–2973–8.

Runarsson, T. and Lucas, S. (2014). Preference Learning for Move Prediction and Evaluation Function Approximation in Othello. *Computational Intelligence and AI in Games, IEEE Transactions on*. ISSN 1943–068X.

Samothrakis, S., Lucas, S., Runarsson, T., and Robles, D. (2013). Coevolving Game-Playing Agents: Measuring Performance and Intransitivities. *IEEE Transactions on Evolutionary Computation*, Vol. 17, No. 2, pp. 213–226.

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, Vol. 3, No. 3, pp. 211–229.

Schaeffer, J. (2007). Game Over: Black to Play and Draw in Checkers. *ICGA Journal*, Vol. 30, No. 4, pp. 187–197.

Shannon, C. E. (1950). XXII. Programming a computer for playing chess. *Philosophical magazine*, Vol. 41, No. 314, pp. 256–275.

Szubert, M., Jaśkowski, W., and Krawiec, K. (2009). Coevolutionary Temporal Difference Learning for Othello. *IEEE Symposium on Computational Intelligence and Games*, pp. 104–111, Milano, Italy.

Szubert, M., Jaśkowski, W., and Krawiec, K. (2011). Learning Board Evaluation Function for Othello by Hybridizing Coevolution with Temporal Difference Learning. *Control and Cybernetics*, Vol. 40.

Szubert, M., Jaśkowski, W., and Krawiec, K. (2013). On Scalability, Generalization, and Hybridization of Coevolutionary Learning: a Case Study for Othello. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 5, No. 3, pp. 214–226.

Thill, M., Koch, P., and Konen, W. (2012). Reinforcement Learning with N-tuples on the Game Connect-4. *Parallel Problem Solving from Nature - PPSN XII* (eds. C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone), Vol. 7491 of *Lecture Notes in Computer Science*, pp. 184–194, Springer. ISBN 978–3–642–32936–4.

Yoshioka, T., Ishii, S., and Ito, M. (1999). Strategy acquisition for the game. *Strategy Acquisition for the Game "Othello" Based on Reinforcement Learning*, Vol. 82, No. 12, pp. 1618–1626.

5. APPENDICES

APPENDIX A: DETAILED RESULTS

	mean	median	0	1	2	3	4	5	6	7	8	9
all-2-inv	0.59888	0.60040	0.5911	0.6005	0.6004	0.6120	0.6039	0.6004	0.5971	0.5867	0.5949	0.6018
all-3-inv	0.58823	0.58945	0.5751	0.5866	0.5909	0.5944	0.5714	0.5620	0.5880	0.6200	0.5995	0.5944
rand-10x3-inv	0.58563	0.59020	0.5870	0.5715	0.6026	0.5990	0.5934	0.5773	0.5989	0.5706	0.5952	0.5608
rand-8x4-inv	0.56300	0.56325	0.5495	0.5813	0.5819	0.5536	0.5629	0.5577	0.5435	0.5636	0.5682	0.5678
rand-7x5-inv	0.53696	0.53020	0.5756	0.5439	0.5625	0.5093	0.5836	0.4874	0.5279	0.5303	0.5301	0.5190
all-4-inv	0.53408	0.53605	0.5337	0.5453	0.5379	0.5401	0.5465	0.5342	0.5253	0.5254	0.4976	0.5548
rand-8x4-neg	0.42811	0.43000	0.4232	0.4200	0.3949	0.4933	0.4368	0.4554	0.3310	0.4724	0.4040	0.4501
all-1-inv	0.41398	0.42205	0.4225	0.4293	0.3877	0.3947	0.3886	0.4321	0.4216	0.4136	0.4253	0.4244
all-1-neg	0.37672	0.37720	0.3727	0.3807	0.3764	0.3783	0.3901	0.3677	0.3806	0.3740	0.3780	0.3687

Table 3: Performances obtained in 10 coevolutionary runs of all n-tuple network architectures considered in this study. Each value is an average score in 1000 double games against a pool of 13 state-of-the-art players in ϵ-Othello, where $\epsilon = 0.1$.