Coevolutionary CMA-ES for Knowledge-Free Learning of Game Position Evaluation

***Abstract*—One weakness of coevolutionary algorithms observed in knowledge-free learning of strategies for adversarial games has been their poor scalability with respect to the number of parameters to learn. In this paper, we investigate to what extent this problem can be mitigated by using Covariance Matrix Adaptation Evolution Strategy, a powerful continuous optimization algorithm. In particular, we employ this algorithm in a competitive coevolutionary setup denoting this setting as Co-CMA-ES. We apply it to learn position evaluation functions for the game of Othello and find out that, in contrast to plain (co)evolution strategies, Co-CMA-ES learns faster, finds superior game-playing strategies and scales better. Its advantages come out into the open especially for large parameter spaces of tens of hundreds of dimensions. For Othello, combining Co-CMA-ES together with an experimentally-tuned derandomized systematic n-tuple networks significantly improved the current state of the art. Our best strategy outperforms all the other Othello 1-ply players published to date by a large margin regardless of whether the round-robin tournament among them involves a fixed set of initial positions or the standard initial position but randomized opponents. These results show a large potential of CMA-ES-driven coevolution, which could be, presumably, exploited also in other games.**

***Index Terms*—competitive coevolution, CMA-ES, n-tuple system, reinforcement learning, large parameter optimization, continuous optimization, numerical optimization, Reversi**

I. Introduction

A great deal of adversarial game playing computer programs involve some form of knowledge elaborated by human experts. For example, Deep Blue, the famous computer chess program, relied on a position evaluation function that was fine-tuned by a chess grandmaster [20]. Moreover, the program took advantage of an opening book consisting of several thousands of professional games [7]. Also in Go, another classic game, the knowledge extracted from the expert games was successfully used to guide the tree policy of the Monte Carlo Tree Search algorithm [11]. More recently, the authors of the best Shogi[1] player optimized the parameters of its evaluation function on the basis of a large game database [19].

Although the successes of such *expert-knowledge-based* methods cannot be questioned, they are limited to domains where the expert knowledge already exists or is affordable to obtain. In the era of digital entertainment only a small fraction of enjoyable games can be ascribed to this category. From a broader perspective, the long-term challenge of artificial intelligence is to design methods that are *knowledge-free* [34] or at least *expert-knowledge-free*. Such methods are able to autonomously find game-playing strategies without (or with very little) knowledge about the game at hand.

The authors are with the Institute of Computing Science, Poznan University of Technology, Poznan, Poland, e-mails: {wjaskowski, mszubert}@cs.put.poznan.pl.

[1]Also known as Japanese chess or the Generals' Game

One knowledge-free approach to games consists in learning strategies (typically in a form of parametrized position evaluation functions) from *self-play*, which means that learning agents gather experience solely from games played against themselves. Two families of methods that are capable of such self-learning include temporal difference learning (TDL, [46], [52]) and coevolutionary learning [1], [18], [37]. While their relative effectiveness is a subject of intensive research [29], [27], it is generally believed that temporal difference learning outperforms coevolutionary algorithms with respect to the learning speed [33], [32]. On the other hand, evolutionary-based methods have been found to achieve higher performance in deterministic tasks [56] and can surpass temporal difference learning in the long run when function approximation is employed [32], [29].

One of the main weaknesses of (co)evolutionary algorithms in the context of learning position evaluation for games is that they scale poorly with the number of parameters of the evaluation function [50]. For instance, in Othello, it was found that increasing the number of parameters above a certain threshold inhibits the evolution process, which, in turn, leads to low performance of the evolved strategies [22].

In this paper, we attempt to mitigate the scalability problems observed in coevolutionary learning of game-playing strategies. To this aim, we harness Covariance Matrix Adaptation Evolution Strategy (CMA-ES), a state of the art continuous optimization method. We demonstrate the superiority of *coevolutionary CMA-ES* (Co-CMA-ES) over plain (co)evolution strategies and show that the advantages of the former come out into the open especially for large parameter spaces of several hundreds of dimensions. To the best of our knowledge, CMA-ES has never been used before in coevolutionary learning of such highly dimensional game position evaluation functions and this is why its benefits have not yet been revealed in this respect.

In the experiments, we apply Co-CMA-ES to Othello, a game which has been a popular benchmark recently [33], [36], [54], [45], [43], [41]. One of the main reasons why Othello is so interesting is that the game is full of dramatic reversals caused by the rapid changes in dominance on the board. Finding a precise evaluation function for such highly volatile states is definitely a non-trivial task.

As a position evaluation function we adopt n-tuple networks [31]. In contrast to their original applications, instead of assigning board patterns to network's inputs randomly, we do it systematically. Extending our earlier results [22], here we further improve such *systematic n-tuple networks* to find out which feature patterns provide the strongest players. Combining the efficiency of coevolutionary CMA-ES and experimentally-tuned systematic n-tuple networks, allows us to produce Othello players that are significantly better at 1-ply than all the other players published to date.

Overall, the major contributions of this paper include: i) experimental evidence that employing CMA-ES to coevolve position evaluation functions with a large number of parameters alleviates previously reported scalability problems; ii) demonstration that, when compared to random input assignment, placing n-tuple networks systematically results in more robust learning and higher performance of produced strategies; iii) investigation into the impact of the size and shape of n-tuple networks on the performance of produced game-playing strategies; iv) producing, without explicit use of expert knowledge, effective Othello position evaluation functions and thoroughly comparing their performance with over a dozen recently published players.

II. CMA-ES for Learning Evaluation Function

Learning real-valued parameters of a position evaluation function can be framed as a black-box continuous optimization problem. Covariance Matrix Adaptation Evolution Strategy (CMA-ES [15]) is a state-of-the-art algorithm for solving such problems. In this section, we describe it and review its past applications to learning strategies in games.

A. CMA-ES

CMA-ES belongs to the class of evolution strategies (ES [2])—evolutionary algorithms that are particularly suitable for continuous optimization. Following the principles of natural evolution, they implement an iterative process of selection, mutation and recombination within a maintained population of individuals. An individual represents a candidate solution while its fitness reflects the value of the objective function being optimized. In each iteration of the algorithm, the fittest individuals undergo recombination and (usually Gaussian) mutation to give rise to a new generation.

The main distinctive features of ES among evolutionary algorithms include deterministic truncation selection, unbiased variation operators and self-adaptation, which allows change of the behavior of variation operators during the course of an optimization process.

CMA-ES (see Algorithm 1) is a modern powerful representative of ES, which has proven successful in a number of difficult, noisy or non-separable continuous optimization problems [14]. In contrast to conventional (plain) ES, instead of an explicit population, CMA-ES maintains a multivariate Gaussian probability distribution $\mathcal{N}(\mathbf{m}, \sigma^2\mathbf{C})$, where $\mathbf{m}$ and $\mathbf{C}$ are, respectively, its mean and covariance matrix, while σ is a step-size. In each iteration, the distribution is used to sample λ new candidate solutions, which are then evaluated. Afterwards, the three parameters of the distribution are updated basing on the observed ranking of the candidate solutions. The new mean $\mathbf{m}$ is calculated as a weighted recombination of the best μ candidate solutions[2]. The update procedures of covariance matrix $\mathbf{C}$ and step-size σ are more complex, and their analysis is beyond the scope of this paper, thus, for their description and analysis the reader is referred to the original work of Hansen [13].

[2]There are multiple ways to define the weights used in the weighted recombination. Here, we use superlinear weights defined as $w_i = \frac{\ln(\mu+1)-\ln(i+1)}{\sum_{j=1}^{\mu}(\ln(\mu+1)-\ln(j+1))}$. Note that those weights should not be confused with the weights of the n-tuple network.

Algorithm 1 General outline of the CMA-ES algorithm.

Require: population size λ, initial distribution mean $\mathbf{m}$, initial step size σ, fitness function f

```
INITIALIZE(μ, w, C = I)                          ▷ Σ_{i=1}^{μ} w_i = 1
repeat
    for k = 1 to λ do x_k ~ N(m, σ²C)
    for k = 1 to λ do EVALUATE(x_k, f)
    m ← Σ_{i=1}^{μ} w_i x_{i:λ}                  ▷ f(x_{i:λ}) > f(x_{i+1:λ})
    UPDATE STEP SIZE(σ)
    UPDATE COVARIANCE MATRIX(C)
until TERMINATION CONDITION
```

The advantages of CMA-ES include its ability to explicitly model dependencies between variables (covariance matrix $\mathbf{C}$), invariance against the translation and rotation of the search space as well as against a monotonic transformation of the fitness function. Let us also note that for all of the parameters of the algorithm (e.g., μ) carefully derived default values have been proposed. The only exception is the sample (population) size λ, which is problem-specific and thus its adjustment is left to the user.

B. CMA-ES in Games

Although in recent years CMA-ES has been a popular choice for tackling various optimization problems, its applications to adversarial games have been rather occasional.

In the area of board games, the first study of CMA-ES was carried out by Konen and Bartz-Beielstein [30], who applied this method to learn strategies for the simple game of Tic-tac-toe. They used CMA-ES with four different fitness functions and found out that it can achieve a comparable performance as temporal difference learning (TDL) only when the fitness function involves a perfect tic-tac-toe player.

Another application of CMA-ES in the context of learning against a fixed game-playing opponent is reported by Schaul and Schmidhuber [44], who considered three Go-inspired games (Atari-Go, Go-Moku and Pente) and investigated the scalability of the proposed neural network architecture with respect to the board size. Remarkably, in their follow-up study [12], they claim that the CMA-ES algorithm "does not scale well to larger number of weights" of a neural network. In particular, other methods, including plain ES, were found to achieve significantly higher performance.

To the best of our knowledge, the scalability issues of the CMA-ES algorithm were not further investigated. As a matter of fact, the vast majority of the reported successes achieved by this algorithm in evolving game-playing strategies are limited to simple representations involving only a small number of parameters to learn. For example, the aforementioned work on Tic-tac-toe [30] employed 19 learning parameters. Likewise, Boumaza [4] obtained one of the best Tetris players to date by optimizing only 8 parameters using CMA-ES. The only exception in this respect is the recent work of Hausknecht et al. [16] on general Atari game playing. The authors used a fixed-topology networks with 580–3560 weights, but CMA-ES was outperformed by the NEAT algorithm.

C. CMA-ES in Reinforcement Learning Problems

Compared with infrequent applications in games, more research has been conducted on applying CMA-ES to a wider class of reinforcement learning problems [46]. In particular, several studies investigated CMA-ES in the context of pole balancing. In one of them, Igel [21] demonstrated that it outperforms other evolutionary methods. In later works [17], CMA-ES was compared also to temporal difference learning methods and, generally, has been found more robust with respect to the choice of parameters and less susceptible to noise, which inherently arises in many control problems.

Besides pole balancing problems, the performance of CMA-ES was evaluated on grid world tasks, which constitute another popular reinforcement learning benchmark. Lucas [32] compared CMA-ES to TDL and found out that the evolutionary approach struggles when using direct table-based strategy representation. However, when using a function approximator in the form of a multi-layer perceptron, CMA-ES got much better results than TDL. In parametrized grid world tasks [29], CMA-ES is reported to be more robust than other evolutionary methods, but also significantly slower than methods based on value functions, such as TDL.

D. Coevolutionary CMA-ES

Most applications of CMA-ES concern problems in which a well-defined objective function is available and can be easily employed to evaluate the fitness of candidate solutions (cf. line 4 of Algorithm 1). However, in the case of learning game-playing strategies, which is an example of a co-optimization problem [38], designing a computationally tractable fitness function is challenging. Indeed, evaluating an absolute performance of a strategy would require playing against all possible opponent strategies. Due to the huge number of possible opponents in nontrivial games, a feasible alternative is to consider only a limited set of opponents, and thus, lessen the computational burden. In this case, the set of opponents used for evaluation could be formed by the predefined expert players or a sample of random opponents.

Instead of using manually or randomly selected opponents, an appealing idea is to employ *round-robin competitive coevolution* [1], [38], which has been applied to many two-player games, including Backgammon [37], Checkers [10] and Othello [33]. In coevolution, the evolving strategies are evaluated by playing against each other. Formally, the fitness of each candidate solution $x \in \mathbf{x}$ is calculated as:

$$f(x) = \sum_{x' \in \mathbf{x} \setminus \{x\}} g(x, x'), \tag{1}$$

where $g(x, x')$ denotes the result of a single game played between two population members: x and x'.

In this work, we employ the competitive fitness function (1) in CMA-ES (Algorithm 1), denoting the resulting algorithm as Co-CMA-ES. Interestingly, only a few previous studies investigated such a setup. Specifically, in the aforementioned work on Tic-tac-toe, despite trying different, albeit small, feature sets, the coevolutionary variant of CMA-ES resulted in slow learning and "turned out to be a failure" [30]. In another work, coevolutionary CMA-ES was applied to Othello [43], but only with a basic strategy representation of a weighted piece counter containing 64 weights.

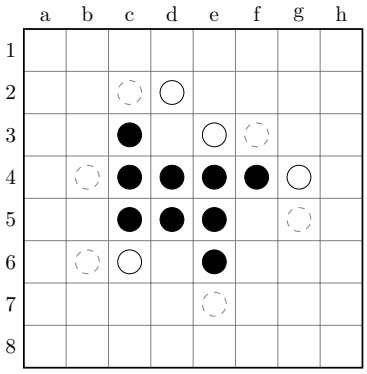


Figure 1: An Othello position, in which the white player has 6 possible moves (dashed gray circles). Playing b4 will make the following pieces white: c3, c4, d4, e4, f4.

Collectively, previous studies provide a few important insights about empirical characteristics of the CMA-ES method. First of all, it was successfully applied to several different reinforcement learning problems (including games), in which, usually, it outperformed other evolutionary approaches. However, most of the reported experiments were conducted in simple tasks with straightforward (even table-based) representations based on a small number of features. Consequently, there is an apparent need of further investigation on CMA-ES and its scalability in non-trivial tasks with much larger strategy spaces. Moreover, due to ambiguous results obtained thus far by combining CMA-ES with coevolution [30], we also attempt to reach better understanding of the empirical performance of this combination.

III. Position Evaluation for Othello

A. Othello

In recent years, the game of Othello has been frequently employed for evaluating both evolutionary [33], [36], [54], [41], [8], [26], [51] and temporal difference learning methods [45], and for comparing their empirical results [27], [41], [50]. The game of Othello is a deterministic, sequential, zero-sum board game played by two players using double-sided pieces with white and black face, each face assigned to one player. The players make moves alternately by placing pieces on the board, with their colors face up. A legal move consists of placing a piece on an empty square and flipping certain opponent's pieces. The location to place a new piece must: 1) be adjacent to one of the opponent's pieces, and 2) form a vertical, horizontal, or diagonal line with another piece of the player in such a way that at least one opponent's piece is surrounded on both sides. All such surrounded opponent's pieces are then flipped to the other color; if multiple lines exist, flipping affects all of them. Figure 1 shows an example of an Othello position with six possible moves of the white player. The game ends when no player has a legal move. The goal of each player is to maximize the number of its pieces at the end of the game.

Since Othello is asymmetrical, in this study, we conveniently consider *double game*s, in which each of the two players play two games in a row: one game as black and one game as white.

B. Position Evaluation using n-Tuple Network

Due to a huge number of possible states in Othello, position evaluation is typically implemented by a function approximator. *n-tuple networks* are effective function approximators, originally proposed by Bledsoe and Browning [3] for optical character recognition. In the context of Othello, they were employed for the first time under the name of *tabular value functions* by Buro [5] in his famous Logistello program. More recently, they were popularized by Lucas [31] and successfully applied in other games such as Connect 4 [53],the puzzle game of *2048* [47] and Tetris [28]

1) n-Tuple Network: An n-tuple network consists of m components, each being a tuple of different board locations. For a given board state $\mathbf{b}$, the network outputs the sum of values returned by the individual tuples. The ith tuple, where $i = 1, \ldots, m$, includes a predetermined sequence of n_i board locations $(loc_{ij})_{j=1,\ldots,n_i}$, and an associated look-up table LUT_i. The table contains weights for each possible board pattern that can be observed on the considered sequence of locations. Thus, an n-tuple network implements a position evaluation function p:

$$p(\mathbf{b}) = \sum_{i=1}^{m} p_i(\mathbf{b}) = \sum_{i=1}^{m} \mathrm{LUT}_i \left[idx\left(\mathbf{b}_{loc_{i1}}, \ldots, \mathbf{b}_{loc_{in_i}}\right)\right]$$

$$idx\left(\mathbf{v}\right) = \sum_{j=1}^{|\mathbf{v}|} v_j c^{j-1},$$

where $\mathbf{b}_{loc_{ij}}$ is a board value at location loc_{ij}, $\mathbf{v}$ is a sequence of board values (the observed pattern) such that $0 \leq v_k < c$, for $k = 1, \ldots, |\mathbf{v}|$, and c is a constant denoting the number of possible board values. In the case of Othello, $c = 3$, and squares occupied by white, empty, and black squares are encoded as 0, 1, and 2, respectively. Consequently one look-up table contains 3^{n_i} weights—one weight for each possible configuration of pieces placed on the given n_i locations.

The effectiveness of n-tuple networks can be improved by using *symmetric sampling*, which exploits the inherent symmetries of a game board [31]. In symmetric sampling, a single tuple is employed eight times, returning one value for each possible board rotation and reflection (see Fig. 2).

2) Systematic n-Tuple Network: To create an n-tuple network, board locations must be first assigned to particular tuples. Since for many board games the spatial neighborhood of chosen locations is essential, tuples are usually consecutive snake-shaped sequences of locations, albeit this is not a formal requirement. In most previous studies, the board locations for such sequences were chosen randomly [31], [53]. Only recently, Jaśkowski [23] proposed a *systematic n-tuple network*. It consists of all possible vertical, horizontal and diagonal n-tuples of the same length (see Fig. 5a). Its smallest representative is a network of 1-tuples. Thanks to symmetric sampling, only 10 of them are required to cover an 8×8 Othello board, and such 10×1-tuple network, which we denote as all-1, contains $10 \times 3^1 = 30$ weights.

3) Evaluation Function Interpretation: Another design issue concerns the way the position evaluation function is used by both players to make moves. We employ *board inversion,* a method first used for Othello by Manning [35] and later studied by Runarsson and Lucas [41]. In this method, the black player selects the move leading to the most valuable positions. If the white player is to move, it temporarily flips all the pieces on the board in order to interpret the board from the black player's perspective. Then it selects the best move, flips all the pieces back, and plays the white piece in the selected location.

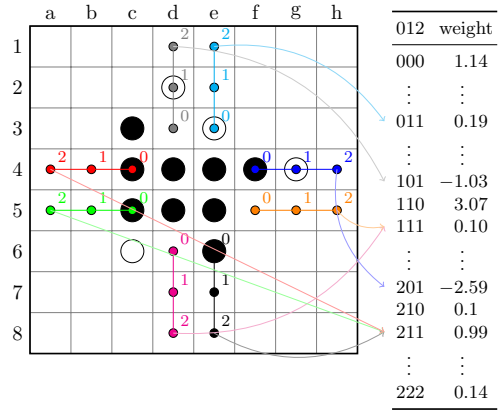


Figure 2: A single straight 3-tuple employed eight times (symmetric sampling) for the given board position. In this example, the eight symmetric expansions of the 3-tuple return $0.19 - 1.03 + 2 \times 0.1 - 2.59 + 3 \times 0.99 = -0.26$.

IV. Experiments and Results

The objectives of experiments presented in this section are twofold. Firstly, we attempt to verify how CMA-ES scales when compared to plain ES. Our second goal is to find out the best n-tuple system architecture.

A. Experimental Setup

In the following experiments, we evolve individuals (candidate solutions), which are real-valued vectors interpreted as the weights of an n-tuple network. To this aim, we consider two algorithms: plain (μ, λ) evolutionary strategy (ES) with $\mu = \lambda/2$ and CMA-ES (see Section II-A). Plain ES involves uniform Gaussian mutation with standard deviation of 1. Similarly, the step-size σ of CMA-ES is initialized[3] to 1. The initial population (or a single point in the case of CMA-ES) is generated by sampling the weights uniformly from the range $[-0.1, 0.1]$.

Each run was repeated 5 times. All runs were stopped after achieving 2000 generations.

1) Coevolutionary Fitness Evaluation: Competitive coevolution, which we utilize here, is driven by the relative fitness of individuals in the population. During the evaluation phase, each individual plays one double game with each other in the population (a round robin tournament). The individual fitness is the sum of scores obtained in these games, where in a single game the win, loss or draw counts as 1, 0.5 or 0 points, respectively. In order to prepare the strategies for different situations, games start from different opening positions. Specifically, each game starts from a random opening position chosen from 1 000 unique 6-ply positions prepared by Runarsson and Lucas [41].

[3]Standard CMA-ES adapts σ on-line, but we have found out that for this application constant sigma works equally well.

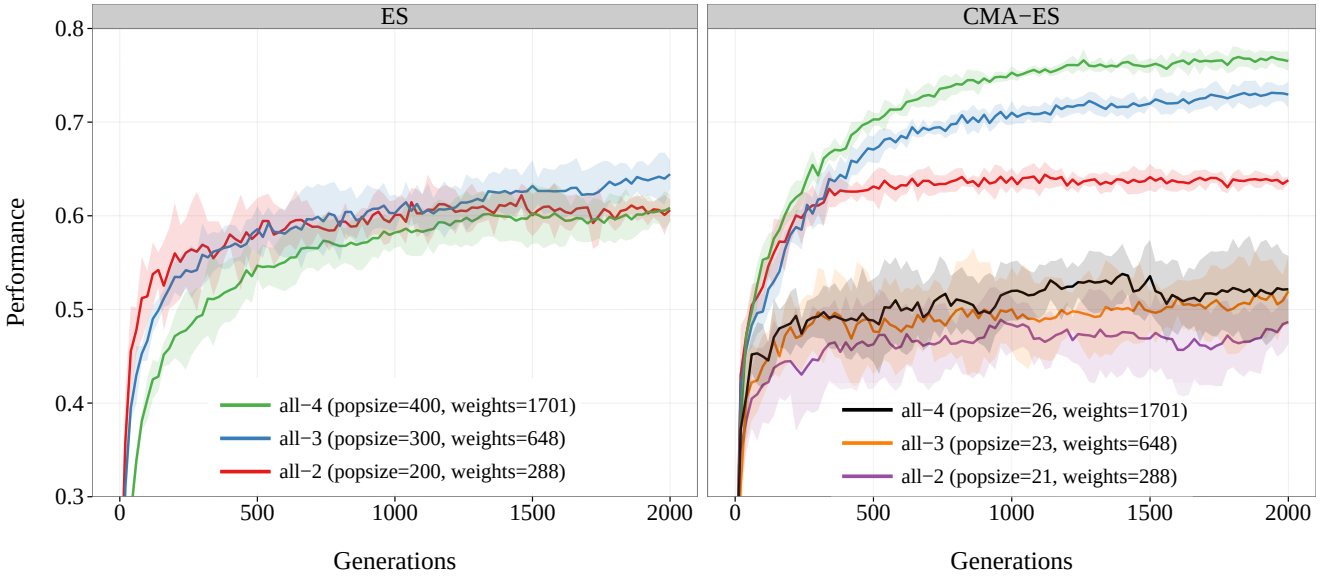


Figure 3: Scalability of ES and CMA-ES. Semi-transparent ribbons visualize 95% confidence bands using Student's *t*-values.

2) External Objective Performance Measure for Monitoring Progress: Because the fitness of coevolving individuals is relative, in order to objectively measure the learning progress, we use an external performance measure. To this aim, we employ 11 previously published players: SWH, LR06, SJK09, SJK11, SJK13_CTDL, SJK13_ETDL, BP11, EM10_Nash70, EM10_TCIAIG2, RL14_iPref1 and RL14_iPrefN, which are described in Section IV-C3. Every 50 generations we measure the performance of the best-of-generation individual (chosen according to the fitness) against those players by taking the average game score over all 1000 opening positions.

3) Population Size: The suggested population size λ for CMA-ES is $4 + 3\ln(d)$, where d denotes dimensionality of the search space [15]. Since in the following experiments d is determined by the number of weights in an n-tuple network and varies in the range from ca. 300 to 3000, the suggested population size would be in the range of $[20, 30]$. However, in the preliminary experiments, we found out that larger populations make the learning more effective, especially for larger search spaces. This lead us to use $\lambda = 200$, $\lambda = 300$, and $\lambda = 400$ for, respectively, $d \in [0, 500)$, $d \in [500, 1000)$, $d \in [1000, 3000)$. The positive effect of larger λ values is demonstrated in the right inset of Fig. 3, which shows learning curves for three systematic n-tuple networks. Clearly, the baseline $\lambda = 4 + 3\ln(d)$ performs poorly on this problem. We have also verified that increasing the population size beyond $\lambda = 400$ provides negligible benefits.

4) Implementation: All the methods studied in this paper were implemented[4] in Java. As for CMA-ES, we used the original implementation authored by Nikolaus Hansen[5]. The experiments were executed using 6 core (12 threads) AMD Opteron™ 6234 and lasted 3h to 30h per run depending on the population size involved and the algorithm used.

[4]The source code required to run all the experiments along with the best obtained Othello players is available at https://github.com/wjaskowski/TCIAIG-2015-Co-CMA-ES

[5]https://code.google.com/p/cma-es

B. Scalability: CMA-ES vs. ES

In the first experiment we compare the scalability of CMA-ES and plain ES with respect to the dimensionality of the search space. For this purpose, we employ three n-tuple network architectures with different number of weights: all-2 (288 weights), all-3 (648 weights), and all-4 (1701 weights).

The results, shown in Fig. 3, confirm our earlier findings [50], [22] that, despite exceeding performance of 0.6, plain ES scales poorly, being unable to take advantage of more features available in larger networks. Although all-3 achieves eventually better performance than all-2, the gain is minor and it is achieved at the cost of slower learning. Enhancing the architecture to tuples of size 4 provides no further advantage as, this time, the all-4 curve does not approach all-3. Moreover, out of the three considered architectures, only all-2 converged in 2000 generations, but the speed of learning of the other two decreases over time.

Being aware of the scalability issues exhibited by ES, we were surprised to see that CMA-ES experiences no such problems, not only significantly improving the performance when the number of weights grows, but also trading-off no learning speed for that.

Fig. 4, which shows data from the same experiment, compares ES and CMA-ES directly for particular network architectures. Clearly, CMA-ES is a better learning technique for this problem. Not only does it achieve significantly higher performance than ES, but it also learns much faster, converges faster, and exhibits less variance. The gap between ES and CMA-ES grows with the architecture size, being the widest for the largest architecture (all-4).

Finally, let us note that the computational overhead related to the CMA-ES is negligible for all-2 and all-3 network architectures. Only for the largest networks, CMA-ES runs take 25% longer than ES runs (ca. 16h vs. 20h), which is caused by the covariance matrix update overhead. Nevertheless, the performance gain obtained by CMA-ES compensates for this negative effect. Therefore, in the rest of this work, we use only coevolutionary CMA-ES.

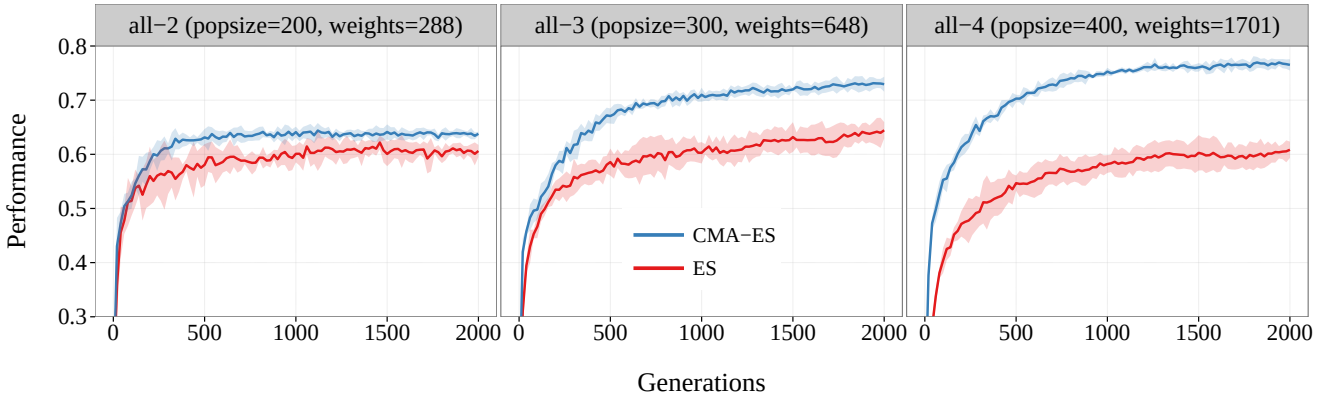
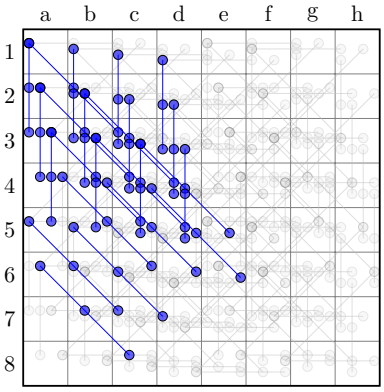
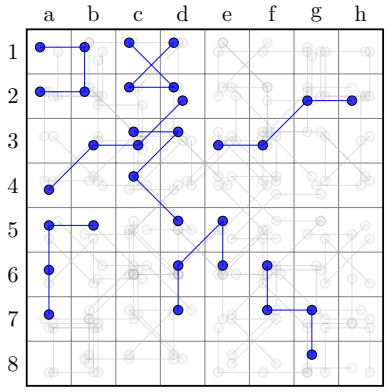


Figure 4: Comparison of ES and CMA-ES for architectures of different weight categories.



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



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

Figure 5: Examples of systematic (all-n) and random (rand-$m\times n$) n-tuple network architectures. "Primary" n-tuples were drawn with blue, while their symmetric expansions with light gray.

C. Which Features are Most Effective?

1) Systematic vs. Random Snake-Like n-Tuples: The most popular method of placing n-tuples on the board consists in randomly generating a small number of long, snake-shaped sequences [36], [50], [41]. Lucas, who introduced this method [31], created tuples by starting from a random location and taking a random walk of 6 steps in any of the 8 orthogonal or diagonal directions. By ignoring repeated locations the resulting tuples had from 2 to 6 elements.

Only in a recent work, Jaśkowski [22] found out that learning n-tuple networks can be more effective if it involves a large number of systematically placed, short, straight n-tuples (see Fig. 5a). These results were obtained for n-tuple networks with a moderate number of weights (< 1000) which were optimized using plain ES. Having found that coevolutionary CMA-ES is a more scalable learning method, we employ it to extend the previous results.

To fairly compare systematic networks with the random ones, we assumed that, regardless of the method of placing n-tuples on the board, the network should involve the same (or at least similar) number of weights. To this aim, we extended the Lucas's method by repeatedly generating a random snake-like sequence until it has length n. A network consisting of m n-tuples generated in this way, denoted as rand-$m\times n$, has $m\times 3^n$ weights (see Fig. 5b for an example).

We performed the comparison in three categories: 288 weights, 648 weights, and 1701 weights, which correspond to all-2, all-3 and all-4 architectures, respectively (see Table I). For each of them, we chose two types of rand-$m\times n$ networks, with the same number of weights. The first type of random networks contained n-tuples of equal length as the corresponding ones in the systematic all-n network. The second type involved a smaller number of longer tuples. The only exception is the category of 288 weights, in which we used only one type of the random network, since systematic and random generation of 32 2-tuples must produce the same network. Also, there is no m such that rand-$m\times 3$ could have 288 weights, which is why we had to fall back on the closest possible rand-10×3 architecture with 270 weights.

#	architecture	weights	architecture	weights	popsize
1	all-2 (32×2)	288	rand-10×3	270	200
2	all-3 (24×3)	648	rand-24×3 rand-8×4	648	300
3	all-4 (21×4)	1701	rand-21×4 rand-7×5	1701	400

Table I: Three pairs of *systematic straight* (all-n) and *random snake-shaped* (rand-$m\times n$) n-tuple network architectures.

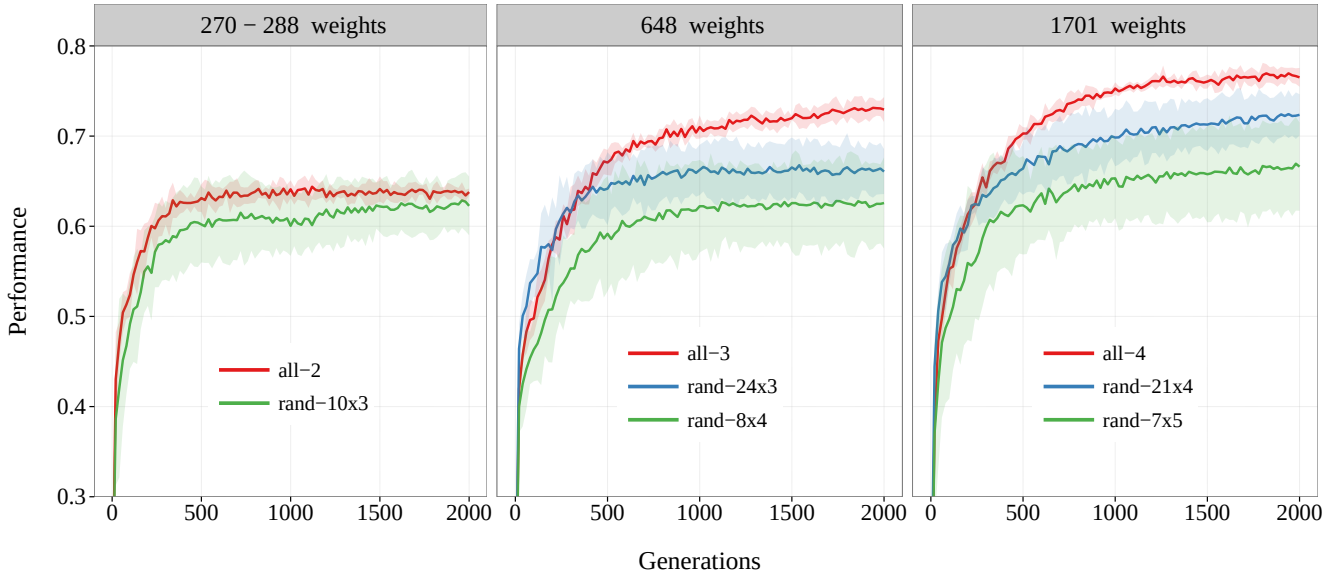


Figure 6: A comparison of systematic (all-n) and random (rand-$m{\times}n$) networks for three different categories: 288 weights, 648 weights, and 1701 weights. Notice that for the two latter ones, rand-$m{\times}n$ networks differ in m, the number of n-tuples.

The experimental results shown in Fig. 6 confirm our past observations [22] by clearly demonstrating that systematic architectures allow achievement higher performance than the random ones. What is new here, however, is that CMA-ES amplifies this effect for larger architectures. In particular, for 288 weights there is no statistical difference between systematic and random networks, but there is one for 648 and 1701 weights as the confidence intervals do not overlap. The largest performance gap can be observed between all-4 and rand-7×5.

Moreover, not only can we conclude that systematic networks are more effective than the random ones, but also that it is more profitable to employ a large number of short tuples instead of a small number of larger ones (see rand-24×3 vs. rand-8×4 and rand-21×4 vs. rand-7×5).

Finally, notice that the systematic networks are also significantly more robust than the random ones (compare the confidence interval ribbons in Fig. 6).

2) Improving Performance with Larger Weight Space: Encouraged by the success of the systematic networks learned by the CMA-ES algorithm, in a quest for even higher performance, we made two attempts to improve our, currently best, all-4 architecture. First, we simply tried to lengthen the tuples by trying all-5 network, consisting of 3402 weights. Figure 7 shows that all-5 brings only a slight improvement over all-4. Noteworthy, the figure illustrates decreasing performance differences between consecutive all-n ($n = 2, \ldots, 5$) architectures suggesting that there is very little to gain by trying $n > 5$.

What, however, resulted in a major performance gain was the addition of square-shaped 4-tuples to all-4, which resulted in a network of 2511 weights denoted as all-4+2×2 (see Fig. 7). Notice also that all-4+2×2 learns significantly faster than other architectures.

Finally, we also tried all-5+2×2, but, again, we observed only a minor average performance improvement over all-4+2×2.

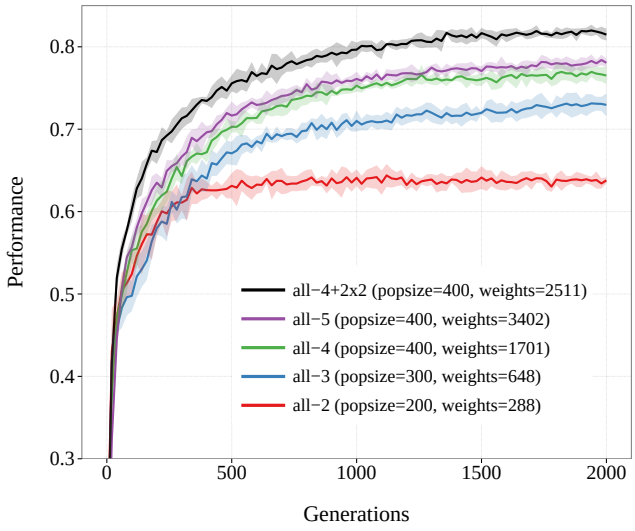


Figure 7: Comparison of systematic n-tuple networks.

3) Computational Efficiency Considerations: Intuitively, the efficiency of function evaluations depends on the number of components they involve. It is also true for n-tuple networks whose evaluation time depends on the number of n-tuples (m) and their size (n). Systematic n-tuple networks can play respectively, ca. 1200, 1100, 1000 and 1000 games per second[6] for all-2, all-3, all-4 and all-4+2×2 architectures. rand-$*$ architectures: rand-10×3, rand-24×3 and rand-21×4 were able to play 1900, 1800 and 1600 games per second, respectively.

The differences among systematic n-tuple networks are minor, but rand-$*$ networks are 50–60% more efficient than all-$*$ networks, since they consist of fewer n-tuples (m) making the number of table lookups lower.Comparison with State of the Art Players

In this section, we compare the performance of our best

[6]A single game involves usually 30 moves per player.

player	eval. function	interpretation	weights	knowledge-free	method
SWH [57]	WPC	neg	64	no	hand-crafted
LR06 [33]	WPC	neg	64	yes	(co)evolution strategy
SJK09 [48]	WPC	neg	64	yes	coevolutionary temporal difference learning (CTDL)
SJK11 [49]	WPC	neg	64	yes	coevolutionary temporal difference learning (CTDL)
SJK13_CTDL [50]	n-tuples	neg	4698	yes	CTDL with a network topology mutation
SJK13_ETDL [50]	n-tuples	neg	4698	no	evolution against SWH hybridized with temporal difference learning
EM10_TCIAIG1 [36]	n-tuples	neg	8748	yes	coevolution with temporal difference learning and resource-limited Nash memory
EM10_TCIAIG2 [36]	n-tuples	neg	8748	yes	coevolution with temporal difference learning and resource-limited Nash memory
EM10_GECCO [35]	n-tuples	inv	8748	yes	coevolution with temporal difference learning and resource-limited Nash memory
EM10_Nash70 [36]	n-tuples	neg	6561	yes	a preliminary version of EM10_TCIAIG1 submitted to Othello League
RL14_iPrefN [41]	n-tuples	inv	6561	no	preference learning on a database of games played by experts
RL14_iPref1 [41]	n-tuples	inv	192	no	preference learning on a database of games played by experts
PB11_ETDL [6]	n-tuples	neg	8748	no	combination of evolution against SWH and temporal difference learning
CoCMAES-4+2×2	n-tuples	inv	2511	yes	Co-CMA-ES with systematic n-tuple network

Table II: A list of 14 state-of-the-art Othello 1-ply players. WPC is a linear representation consisting of 64 weights, one for each field. To interpret board evaluation players use either output negation (neg) or board inversion (inv, see Section III-B3). PB11_ETDL and SJK13_ETDL are knowledge-based as they were trained against the hand-crafted SWH player.

# weights	player	**CoCMAES-4+2x2**	EM10_GECCO	RL14_iPrefN	SJK13_CTDL	EM10_Nash70	PB11_ETDL	EM10_TCIAIG1	EM10_TCIAIG2	SJK13_ETDL	SWH	SJK09	SJK11	LR06	RL14_iPref1	**Total**
2511	**CoCMAES-4+2x2**	**-**	**71.3**	**63.5**	**80.2**	**79.5**	**83.1**	**79.0**	**80.5**	**80.9**	79.0	**93.1**	**93.9**	**93.4**	**95.4**	**82.5**
8748	EM10_GECCO	28.7	-	45.5	67.6	73.4	70.7	75.5	71.6	77.4	63.6	89.6	89.4	87.5	89.6	71.5
6561	RL14_iPrefN	36.5	54.6	-	57.2	67.8	62.3	64.4	66.6	69.3	59.6	82.8	83.2	83.1	89.7	67.5
4698	SJK13_CTDL	19.8	32.4	42.8	-	55.2	60.6	51.9	56.6	67.1	76.8	91.5	92.4	92.0	90.1	63.8
6561	EM10_Nash70	20.5	26.6	32.2	44.8	-	54.5	49.3	50.8	56.8	79.8	87.1	87.0	87.9	83.6	58.5
8748	PB11_ETDL	17.0	29.3	37.7	39.4	45.5	-	48.3	47.8	58.0	82.3	85.5	88.3	87.9	87.6	58.0
8748	EM10_TCIAIG1	21.0	24.5	35.6	48.1	50.7	51.7	-	52.6	59.0	64.7	85.4	86.6	86.4	82.4	57.6
8748	EM10_TCIAIG2	19.5	28.4	33.4	43.5	49.2	52.3	47.4	-	53.6	73.4	85.4	83.6	86.0	78.3	56.4
4698	SJK13_ETDL	19.1	22.6	30.7	32.9	43.2	42.0	41.0	46.5	-	**83.6**	76.4	73.9	74.6	77.1	51.0
64	SWH	21.1	36.4	40.4	23.2	20.2	17.8	35.3	26.7	16.5	-	51.2	51.2	50.0	78.0	36.0
64	SJK09	6.9	10.4	17.2	8.5	12.9	14.5	14.6	14.6	23.6	48.8	-	52.4	48.7	62.8	25.8
64	SJK11	6.1	10.7	16.9	7.6	13.0	11.8	13.4	16.5	26.1	48.8	47.6	-	50.9	63.6	25.6
64	LR06	6.6	12.6	16.9	8.0	12.1	12.1	13.6	14.1	25.4	50.0	51.3	49.1	-	60.3	25.5
192	RL14_iPref1	4.6	10.4	10.3	9.9	16.5	12.4	17.6	21.8	23.0	22.0	37.2	36.4	39.7	-	20.1

Table III: Results of the round robin tournament between 14 selected Othello players. Each value denotes the score obtained in a head to head match between two players expressed in percent points (relative to the maximum possible score in a match). Each match has been played starting from a fixed set of 1000 game positions (see Section IV-A), thus involving 2000 single games. The whole tournament involved 182 000 games. Best scores in each column are marked in bold.

player, named CoCMAES-4+2×2 obtained in the 2000th generation of all-4+2×2 experiment, against 13 other Othello 1-ply strategies published by different researchers over the past dozen or so years (see Table II[7]).

4) Round Robin Tournament: The results of the round robin tournament between the 14 considered players are shown in Table III. CoCMAES-4+2×2 finished the tournament first with the total score of 82.5%, defeating the runner-up by a large margin of 11%. Moreover, our player won all head-to-head matches beating the second-best EM10_GECCO 71.3% to 28.7% and the third-best RL14_iPrefN 63.5% to 36.5%.

Notice also that considering results against particular opponents (in each column), CoCMAES-4+2×2 scores better than any other player. The only exception is SWH against which CoCMAES-4+2×2 is weaker than SJK13_ETDL, PB11_ETDL and (slightly) EM10_Nash70. The reason why SJK13_ETDL and PB11_ETDL perform well against it is that SWH has been involved in their learning process [50], [6]. Therefore, they are overfitted to this particular player.

5) Performance Profiles: The results of the round robin tournament can be conveniently visualized as a series of

[7]Note that 11 of the players in the table were used for monitoring the learning progress (see Section IV-A2). We did not use the other two players for this purpose (EM10_Nash70, and EM10_TCIAIG1), since we obtained them from their author only after performing the experiments.

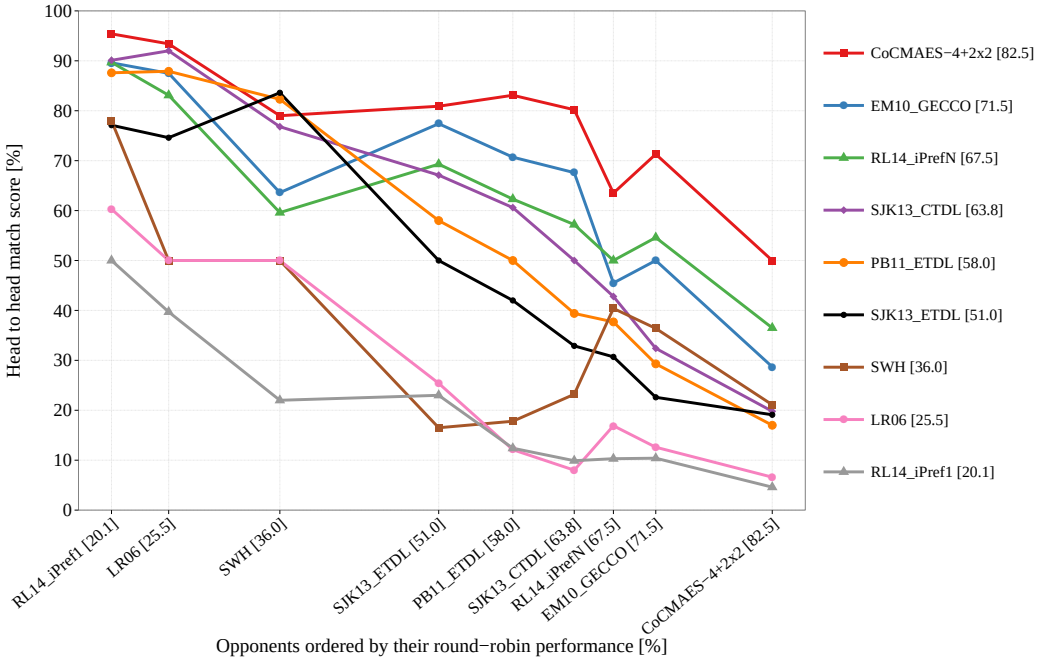


Figure 8: The round robin results visualized as a set of performance profiles. Each plot represents a performance profile of a given player. A player's profile is a vector of points (opponent performance, head to head match score) ordered by the opponent performance. Here, as player's performance we use the round robin total score. For clarity of presentation, only the nine most interesting players have been shown.

performance profiles [26], [25], shown in Fig. 8. Each performance profile, represented by a polyline in the figure, shows scores of a given player against opponents of increasing strength. Not surprisingly, a (roughly) downward trend is common to all the profiles, since it is harder to get a higher score against stronger opponents.

Importantly, a profile can reveal the bias of the learning method used to produce a particular player. For instance, we can see the aforementioned overfitting of SJK13_ETDL and PB11_ETDL to SWH, at which point the two profiles have locally higher values. Another observation concerns RL14_iPrefN, which plays well against strong opponents (e.g., CoCMAES-4+2×2), but relatively badly against the weak ones (e.g., SWH, LR06). The probable reason for this behavior is that RL14_iPrefN has been obtained by supervised learning on a set of games played by expert (human) Othello players. This is why RL14_iPrefN might be biased towards playing against strong players. This hypothesis is supported by a similar effect in the profile of RL14_iPref1, which was learned in the same way as RL14_iPrefN, but represented only by tuples of size 1.

To further investigate this issue, we compared the best players against the random player, which selects each move uniformly at random. The average results over $100\,000$ double games accompanied with 95% confidence intervals for CoCMAES-4+2×2, RL14_iPrefN and SJK13_CTDL are 99.26 ± 0.04, 98.72 ± 0.05, and, 97.73 ± 0.06, 99.53 ± 0.03, respectively. The random player is a poor opponent, but RL14_iPrefN still loses against it in nearly 2.5% of cases. We suspect that this poor result is because of RL14_iPrefN has been trained on an expert game database [41]. Thus, it learned to play well only against strong players, while never experiencing the weak ones.

Finally, notice that the performance profile of CoCMAES-4+2×2 dominates all other profiles (when excluding SWH from the consideration). This stands in sharp contrast with other strong players (EM10_GECCO, RL14_iPrefN and SJK13_CTDL), which are mutually incomparable when interpreting the nine considered strategies as separate objectives.

6) Half-Random Tournament: Recall that during coevolution the individuals begin the game from a position drawn from a fixed set of 1000 initial positions. In order to exclude the possibility that our players specifically evolved to play on the given set of 1000 positions, we perform a *half-random tournament*. This time all games start from the original Othello initial position. Since the players to be evaluated are deterministic[8], in order, to differentiate the conditions under which they are tested, their opponents (but only them) are forced to make a random move with probability $\epsilon = 0.1$. The results of the tournament are presented in Table. IV.

We can observe that CoCMAES-4+2×2 is also the unquestionable winner of this tournament surpassing the runner-up by nearly 8%. Interestingly, although the runner-up has not changed compared to the first tournament, SJK13_CTDL now slightly surpasses RL14_iPrefN. This is because forcing the opponents to make random moves from time to time makes them weaker. Thus, the low score of RL14_iPrefN can be explained by its tendency of relatively poor performance against weaker players, observed already in the previous round-robin tournament.

7) Number of Weights vs. Performance: So far we have compared players only in terms of their relative performance. However, it is interesting to analyze them also with respect

[8]Except the rare situation when the network returns the same value for multiple moves considered.

# weights	player	**CoCMAES-4+2x2**	EM10_GECCO	SJK13_CTDL	RL14_iPrefN	PB11_ETDL	EM10_Nash70	EM10_TCIAIG1	EM10_TCIAIG2	SJK13_ETDL	SWH	SJK11	LR06	SJK09	RL14_iPref1	**Total**
2511	**CoCMAES-4+2x2**	**-**	**87.1**	**83.0**	**74.7**	**91.6**	**93.4**	**89.1**	78.3	**88.7**	93.4	**97.4**	**98.0**	**97.7**	**96.4**	**89.9**
8748	EM10_GECCO	59.1	-	78.4	64.1	83.6	88.7	85.1	**88.9**	83.2	77.2	78.3	93.1	93.0	94.7	82.1
4698	SJK13_CTDL	58.1	69.4	-	66.6	81.6	70.4	76.1	80.7	83.5	70.4	95.5	94.5	96.4	93.9	79.8
6561	RL14_iPrefN	69.6	77.6	71.3	-	75.8	83.4	79.7	79.7	73.8	63.6	87.1	91.1	87.9	95.5	79.7
6561	PB11_ETDL	50.5	59.0	63.1	60.1	-	74.9	69.0	68.9	77.7	96.5	92.5	94.3	96.2	94.2	76.7
8748	EM10_Nash70	38.7	55.1	67.3	47.2	67.6	-	71.5	69.1	65.9	87.8	93.7	92.2	92.7	90.8	72.3
8748	EM10_TCIAIG1	47.8	56.1	64.3	56.1	71.4	66.6	-	69.3	69.4	76.9	91.7	87.5	90.8	88.2	72.0
8748	EM10_TCIAIG2	54.9	48.6	54.4	51.4	67.8	72.8	68.6	-	66.2	85.1	85.8	90.9	90.6	86.5	71.1
4698	SJK13_ETDL	46.9	52.6	54.2	58.5	60.8	74.4	66.3	69.4	-	**96.8**	84.9	81.3	89.7	87.5	71.0
64	SWH	29.8	44.4	44.0	45.9	19.2	31.0	42.9	38.2	17.1	-	62.9	55.2	66.3	87.6	45.0
64	SJK11	19.5	42.4	21.4	25.9	28.4	31.1	26.5	36.4	39.7	59.2	-	67.8	63.3	66.8	40.6
64	LR06	16.9	28.4	21.9	25.2	20.3	30.6	30.2	29.6	42.4	68.1	58.8	-	67.2	68.1	39.0
64	SJK09	19.6	27.3	23.8	25.6	25.1	28.5	25.9	32.4	32.8	55.1	63.2	64.1	-	62.5	37.4
192	RL14_iPref1	18.3	34.8	27.7	22.1	27.6	31.5	31.1	37.3	35.6	23.3	55.0	52.2	57.3	-	34.9

Table IV: The results of a half-random tournament between 14 published Othello players. Each game starts deterministically from the standard initial Othello board. The "row" player plays according to its strategy, but the "column" player (ϵ-opponent) is forced to make random moves with probability $\epsilon = 0.1$. This is why the scores does not sum to 100%. Each score is a percent of points obtained in 1000 double games between a player and a ϵ-opponent.

to the number of weights (parameters) they consist of. When taking into consideration the top 9 players from Table III, we can notice that CoCMAES-4+2×2, apart from outperforming the other strategies, has been obtained by searching the smallest parameter space. It has only 2511 weights compared to, e.g., 8748 weights of the runner-up (see Table IV). This implies that the effectiveness of our methodology is due to the proper feature space, an effective learning method, or, most probably, a combination of both.

The latter claim can be further supported by analyzing the players of simpler architectures coevolved by Co-CMA-ES (see Section 3). To this aim, we conducted a new round-robin tournament among a set of 17 players including the 14 ones described in Section IV-C3 and, additionally, CoCMAES-2, CoCMAES-3 and CoCMAES-4. The latter ones are the best players obtained in the previous experiments involving the following architectures, respectively: all-2 (288 weights), all-3 (768 weights) and all-4 (1711 weights).

Figure 9 compares the players on two criteria: i) relative performance in the tournament, and ii) the number of parameters of its architecture (assuming that the smaller the number of parameters, the better). Observe that the four players learned by Co-CMA-ES are non-dominated, and, together with SWH form a Pareto-front.

As a side note, let us observe that, despite several efforts [33], [48], [49], among the players using the simplest WPC encoding, the hand-designed SWH is still the best one.

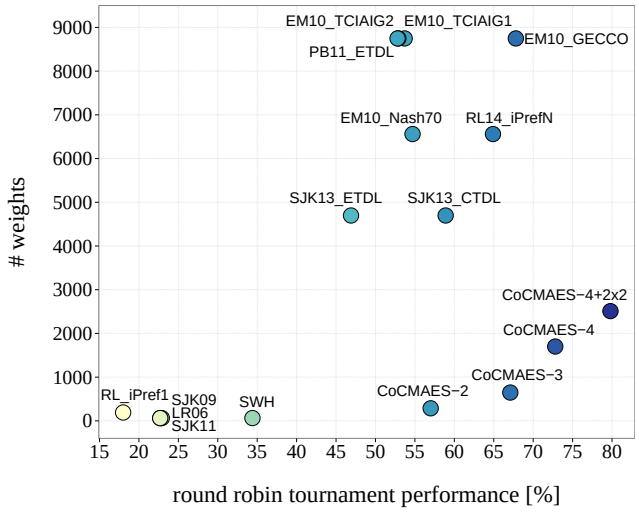


Figure 9: A comparison of 17 Othello players on two objectives: i) a round robin tournament performance (more is preferred) and number of weights (less is preferred). Co-CMA-ES-evolved players lie on the Pareto front.

V. Discussion and Conclusions

In this paper, we have improved over the state-of-the-art methodology for knowledge-free learning of position evaluation functions for the game of Othello. This result has been achieved by a synergy of two ingredients: i) Co-CMA-ES, a coupling of competitive coevolution and CMA-ES, and ii) an experimentally-refined n-tuple network consisting of a large number of relatively short tuples, systematically placed on the board. In a comprehensive comparison, we demonstrated that our best player outperforms by a large margin all Othello 1-ply players published to date. In particular, the player obtained by Co-CMA-ES achieves the highest score in a round robin tournament, regardless of whether the tournament involved a fixed set of initial positions or the standard initial position but with randomized opponents. It also wins against all its competitors in head-to-head matches and dominates them in terms of performance profiles.

Interestingly, the best n-tuple network obtained in this study is relatively small in terms of the number of weights (2511) compared to the top previously published players (4698–8748 weights). What is more, using Co-CMA-ES we were also able to coevolve even smaller networks (288–1701

weights), which can still be regarded as remarkably effective solutions to the considered benchmark problem.

One of the main findings of this study is that Co-CMA-ES does not suffer from scalability issues reported in the previous works on coevolutionary game strategy learning [50]. In contrast to plain (co)evolution strategies, CMA-ES-driven coevolution benefits from increasing the size of the parameter search space by consistently improving the final performance of produced Othello players.

Does it mean that Co-CMA-ES for Othello will profit from even larger networks than those considered in this work? In its current form, probably not, because of two reasons. First, we have not found a way to significantly enhance the performance of all-4+2×2 architecture. For example, moving from tuples of size 4 to tuples of size 5 provides only a tiny performance gain at the expense of doubling the number of weights required to learn (1701 vs. 3402). Second, the covariance matrix and the time CMA-ES requires to update it grow proportionally to the square of the number of parameters to optimize. In our implementation, already for 3402 weights, the time required to update the covariance matrix starts to exceed the time needed for the round-robin fitness evaluation of the population of 400 individuals. Although there exists a version of CMA-ES that achieve linear time and space complexity [39], our preliminary experiments have shown that the price for the lower computational cost are much weaker optimization results than these of the "standard" CMA-ES.

From a broader perspective, it is also important to note that our experiments showed no visible coevolutionary pathologies [9] or hard-to-understand and unexpected dynamics [55] often observed in coevolutionary learning. Our learning curves are roughly monotonic (if we ignore the fluctuations caused by the noisy performance measure). Notice that Co-CMA-ES achieved this without involving any techniques explicitly sustaining the learning progress such as coevolutionary archives [40], [24], [35]. We speculate that it may be due to employing relatively large populations (we employed $\lambda = 400$) compared to some other coevolutionary studies, in which small population sizes ($\lambda < 100$) dominate [30], [8], [43]. Employing large population is also the likely reason why Co-CMA-ES proved successful in contrast to some of the previously considered combinations of coevolution and CMA-ES (mentioned in Section II-D).

One limitation of our method is that, despite the agent learned from scratch without any expert player knowledge, it is also a product of a human-driven experimentation with sets of features (n-tuples). Thus, a future direction consists in letting the learning algorithm extract the features by itself. Some attempts of this kind has been already made in the context of Othello by Manning [36], Burrow [6] and Szubert et al. [50], but neither of them involved CMA-ES, because it is suitable for numerical optimization only. Effectively hybridizing CMA-ES with algorithms capable of handling discrete spaces could free the human designer of manually experimenting with feature extracting methods.

Other interesting future research directions include further improving the performance of the Co-CMA-ES. Apart from trying other variants of CMA-ES, one could employ a two-population coevolution in place of the one-population setup. This might open the possibility to separate two roles played by the individuals of the population: i) search space explorers, and ii) fitness evaluators. In the one-population setup and round-robin fitness evaluation, increasing the population size λ results in increasing both the exploration capacities, and precision of the fitness evaluation. Both effects are positive but the latter one comes at a computational cost of $O(\lambda^2)$. In a two-population coevolution, the two populations could have different sizes. Thus, we could separately tune the search space exploration capabilities, and the precision of fitness evaluation. In this way, we could improve the learning speed, or at least maintain it while using less function evaluations, thus saving computational resources.

Finally, although our best 1-ply player already defeats the authors of this study (who play Othello at amateur level), its strength could be further improved by employing game tree search. The impact of the look-ahead search depth on the performance of 1-ply-trained Othello players has been investigated by Runarsson and Jonsson [42], but only for a simple WPC representation.

Acknowledgments

We would like to thank T. Runarsson and S. Lucas for providing us the set of 1000 initial boards [41], and S. Lucas and E. Manning for providing their Othello players. This work has been supported by the Polish National Science Centre grant no. DEC-2013/09/D/ST6/03932. The computations were performed in Poznan Supercomputing and Networking Center.

REFERENCES

[1] Peter J. Angeline and Jordan B. Pollack. Competitive Environments Evolve Better Solutions for Complex Tasks. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 264–270, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[2] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[3] Woodrow Wilson Bledsoe and Iben Browning. Pattern recognition and reading by machine. In *Proc. Eastern Joint Comput. Conf.*, pages 225–232, 1959.

[4] Amine Boumaza. On the evolution of artificial Tetris players. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 387–393. IEEE, 2009.

[5] Michael Buro. Experiments with Multi-ProbCut and a new high-quality evaluation function for Othello. In H. J. van den Herik et al., editor, *Games in AI Research*, pages 77–96. Univ. Maastricht, 2000.

[6] Peter Burrow. *Hybridising evolution and temporal difference learning*. PhD thesis, University of Essex, 2011.

[7] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1):57–83, 2002.

[8] Siang Yew Chong, Peter Tino, Day Chyi Ku, and Xin Yao. Improving Generalization Performance in Co-Evolutionary Learning. *IEEE Transactions on Evolutionary Computation*, 16(1):70–85, 2012.

[9] Sevan G. Ficici and Jordan B. Pollack. Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states. In *Proceedings of the Sixth International Conference on Artificial Life*, pages 238–247. MIT Press, 1998.

[10] David B. Fogel. *Blondie24: Playing at the Edge of AI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[11] Sylvain Gelly, Levente Kocsis, Marc Schoenauer, Michele Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. The grand challenge of computer Go: Monte Carlo tree search and extensions. *Communications of the ACM*, 55(3):106–113, 2012.

[12] Mandy Grüttner, Frank Sehnke, Tom Schaul, and Jürgen Schmidhuber. Multi-dimensional deep memory atari-go players for parameter exploring policy gradients. In *Proceedings of the 20th International Conference on Artificial Neural Networks: Part II*, ICANN'10, pages 114–123, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] Nikolaus Hansen. The CMA evolution strategy: a comparing review. In J.A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006.

[14] Nikolaus Hansen. Benchmarking a bi-population cma-es on the bbob-2009 function testbed. In *Proc. of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conf.: Late Breaking Papers*, pages 2389–2396, NY, USA, 2009. ACM.

[15] Nikolaus Hansen and Andreas Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[16] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014.

[17] Verena Heidrich-Meisner and Christian Igel. Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms*, 64(4):152–168, 2009.

[18] W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In C. G. Langton et al., editor, *Artificial life II*, volume 10 of *Sante Fe Institute Studies in the Sciences of Complexity*, pages 313–324, Redwood City, 1992. Addison-Wesley.

[19] Kunihito Hoki and Tomoyuki Kaneko. Large-Scale Optimization for Evaluation Functions with Minimax Search. *J. Artif. Intell. Res.(JAIR)*, 49:527–568, 2014.

[20] Feng-hsiung Hsu. IBM's Deep Blue chess grandmaster chips. *IEEE Micro*, 19(2):70–81, 1999.

[21] Christian Igel. Neuroevolution for Reinforcement Learning using Evolution Strategies. In *Proceedings of the 2003 Congress on Evolutionary Computation*, volume 4, pages 2588–2595, 2003.

[22] Wojciech Jaśkowski. Systematic n-tuple networks for othello position evaluation. *ICGA Journal*, 37(2):85–96, June 2014.

[23] Wojciech Jaśkowski. Systematic n-tuple networks for position evaluation: Exceeding 90% in the othello league. Technical Report RA-06/2014, arXiv:1406.1509, Poznan University of Technology, 2014.

[24] Wojciech Jaśkowski and Krzysztof Krawiec. Coordinate system archive for coevolution. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–10, Barcelona, 2010. IEEE.

[25] Wojciech Jaśkowski, PawełLiskowski, Marcin Szubert, and Krzysztof Krawiec. Performance profile: a multi-criteria performance evaluation method for test-based problems. *International Journal of Applied Mathematics and Computer Science*, to appear.

[26] Wojciech Jaśkowski, Paweł Liskowski, Marcin G. Szubert, and Krzysztof Krawiec. Improving Coevolution by Random Sampling. In *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO '13, pages 1141–1148, New York, NY, USA, 2013. ACM.

[27] Wojciech Jaśkowski, Marcin Szubert, and Paweł Liskowski. Multi-criteria comparison of coevolution and temporal difference learning on othello. In A. I. Esparcia-Alcazar and A. M. Mora, editors, *EvoApplications 2014*, volume 8602 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2014.

[28] Wojciech Jaśkowski, Marcin Szubert, Paweł Liskowski, and Krzysztof Krawiec. High-dimensional function approximation for knowledge-free reinforcement learning: a case study in sz-tetris. In *GECCO'15: Proceedings of the 17th annual conference on Genetic and Evolutionary Computation*, pages 567–574, Madrid, Spain, July 2015. ACM.

[29] Shivaram Kalyanakrishnan and Peter Stone. Characterizing reinforcement learning methods through parameterized learning problems. *Machine Learning*, 84(1-2):205–247, 2011.

[30] Wolfgang Konen and Thomas Bartz-Beielstein. Reinforcement learning for games: failures and successes. In *Proc. of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conf.: Late Breaking Papers*, pages 2641–2648. ACM, 2009.

[31] Simon M. Lucas. Learning to play Othello with N-tuple systems. *Australian Journal of Intelligent Information Processing Systems, Special Issue on Game Technology*, 9(4):01–20, 2007.

[32] Simon M Lucas. Estimating learning rates in evolution and TDL: Results on a simple grid-world problem. In *Computational Intelligence and Games, 2010 IEEE Symposium on*, pages 372–379. IEEE, 2010.

[33] Simon M. Lucas and Thomas P. Runarsson. Temporal difference learning versus co-evolution for acquiring othello position evaluation. In *IEEE Symposium on Computational Intelligence and Games*, pages 52–59. IEEE, 2006.

[34] Jacek Mańdziuk. *Knowledge-Free and Learning-Based Methods in Intelligent Game Playing*, volume 276 of *Studies in Computational Intelligence*. Springer, 2010.

[35] Edward P. Manning. Coevolution in a large search space using resource-limited nash memory. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, pages 999–1006, New York, New York, USA, July 2010. ACM Press.

[36] Edward P. Manning. Using Resource-Limited Nash Memory to Improve an Othello Evaluation Function. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):40–53, 2010.

[37] Jordan B. Pollack and Alan D. Blair. Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32(3):225–240, 1998.

[38] Elena Popovici, Anthony Bucci, R. Paul Wiegand, and Edwin D. de Jong. Coevolutionary Principles. In Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors, *Handbook of Natural Computing*, pages 987–1033. Springer, 2012.

[39] Raymond Ros and Nikolaus Hansen. A simple modification in CMA-ES achieving linear time and space complexity. In *Parallel Problem Solving from Nature–PPSN X*, pages 296–305. Springer, 2008.

[40] Chrisotpher D. Rosin and Richard K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

[41] Thomas Runarsson and Simon Lucas. Preference Learning for Move Prediction and Evaluation Function Approximation in Othello. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(3):300–313, 2014.

[42] Thomas Philip Runarsson and Egill O. Jonsson. Effect of look-ahead search depth in learning position evaluation functions for Othello using e-greedy exploration. In *IEEE Symposium on Computational Intelligence and Games, CIG 2007*, pages 210–215. IEEE, 2007.

[43] Spyridon Samothrakis, S Lucas, TP Runarsson, and David Robles. Coevolving Game-Playing Agents: Measuring Performance and Intransitivities. *IEEE Transactions on Evolutionary Computation*, 17(2):213–226, 2013.

[44] Tom Schaul and Jürgen Schmidhuber. Scalable neural networks for board games. In *Proceedings of the 19th International Conference on Artificial Neural Networks: Part I*, ICANN '09, pages 1005–1014, Berlin, Heidelberg, 2009. Springer-Verlag.

[45] Ioannis E. Skoulakis and Michail G. Lagoudakis. Efficient Reinforcement Learning in Adversarial Games. In *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, pages 704–711. IEEE, November 2012.

[46] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1998.

[47] Marcin Szubert and Wojciech Jaśkowski. Temporal difference learning of n-tuple networks for the game 2048. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.

[48] Marcin Szubert, Wojciech Jaśkowski, and Krzysztof Krawiec. Coevolutionary temporal difference learning for othello. In *IEEE Symposium on Computational Intelligence and Games*, pages 104–111, Milano, Italy, 2009.

[49] Marcin Szubert, Wojciech Jaśkowski, and Krzysztof Krawiec. Learning board evaluation function for othello by hybridizing coevolution with temporal difference learning. *Control and Cybernetics*, 40(3):805–831, 2011.

[50] Marcin Szubert, Wojciech Jaśkowski, and Krzysztof Krawiec. On scalability, generalization, and hybridization of coevolutionary learning: a case study for othello. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(3):214–226, 2013.

[51] Marcin Szubert, Wojciech Jaśkowski, Paweł Liskowski, and Krzysztof Krawiec. Shaping Fitness Function for Evolutionary Learning of Game Strategies. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1149–1156, New York, NY, USA, 2013. ACM.

[52] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[53] Markus Thill, Patrick Koch, and Wolfgang Konen. Reinforcement Learning with N-tuples on the Game Connect-4. In *Proc. of the 12th International Conference on Parallel Problem Solving from Nature*, pages 184–194, Berlin, Heidelberg, 2012. Springer.

[54] Sjoerd van den Dries and Marco A. Wiering. Neural-Fitted TD-Leaf Learning for Playing Othello With Structured Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 23(11):1701–1713, November 2012.

[55] Richard A. Watson and Jordan B. Pollack. Coevolutionary Dynamics in a Minimal Substrate. In *Proc. of the Genetic and Evolutionary Computation Conference*, pages 702–709. Morgan Kaufmann, 2001.

[56] Shimon Whiteson, Matthew E Taylor, and Peter Stone. Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 21(1):1–35, 2010.

[57] Taku Yoshioka, Shin Ishii, and Minoru Ito. Strategy acquisition for the game. *Strategy Acquisition for the Game "Othello" Based on Reinforcement Learning*, 82(12):1618–1626, 1999.