On Scalability, Generalization, and Hybridization of Coevolutionary Learning: A Case Study for *Othello*

Marcin Szubert, Wojciech Jaśkowski, and Krzysztof Krawiec, *Member, IEEE*

***Abstract*—This study investigates different methods of learning to play the game of *Othello*. The main questions posed concern scalability of algorithms with respect to the search space size and their capability to generalize and produce players that fare well against various opponents. The considered algorithms represent strategies as n-tuple networks, and employ self-play temporal difference learning (TDL), evolutionary learning (EL) and coevolutionary learning (CEL), and hybrids thereof. To assess the performance, three different measures are used: score against an *a priori* given opponent (a fixed heuristic strategy), against opponents trained by other methods (round-robin tournament), and against the top-ranked players from the online *Othello* League. We demonstrate that although evolutionary-based methods yield players that fare best against a fixed heuristic player, it is the coevolutionary temporal difference learning (CTDL), a hybrid of coevolution and TDL, that generalizes better and proves superior when confronted with a pool of previously unseen opponents. Moreover, CTDL scales well with the size of representation, attaining better results for larger n-tuple networks. By showing that a strategy learned in this way wins against the top entries from the *Othello* League, we conclude that it is one of the best 1-ply *Othello* players obtained to date without explicit use of human knowledge.**

***Index Terms*—Coevolution, n-tuple systems, *Othello*, temporal difference learning (TDL).**

I. Introduction

LEARNING a game-playing strategy can be naturally viewed as searching through a space of all possible strategies. Like in every other search problem, two questions have to be answered in order to design an efficient search algorithm. First, what is the search domain, i.e., how is the space of candidate solutions defined? Second, what is the goal of the search problem, i.e., what are the properties of the desired outcome of the search?

These questions are particularly important in the domain of games, where the learning problem alone typically does not provide obvious answers to them. Regarding the search space, for most nontrivial games, it is impossible to represent a candidate solution (i.e., a strategy) directly as a mapping from states to actions, due to a huge number of states. Thus, typically, a more concise way of storing strategies is employed, for instance, a position evaluation function approximated by a neural network. Importantly, the choice of strategy representation determines the size and characteristics of the search space of the learning problem.

When it comes to search goals, in contrast to many combinatorial optimization problems, there are no predefined objective functions for learning game-playing strategies. Instead, the goal of search can be defined using a *solution concept* [9] that implicitly divides the search space into solutions and nonsolutions. Among several solution concepts applicable to the interactive domain of games, like Nash equilibrium and Pareto optimality, the most intuitive one is *maximization of expected utility*, i.e., maximization of the expected score against a randomly selected opponent [8]. This concept corresponds to the measure of *generalization performance* [7], which, in turn, follows the notion of generalization in machine learning. For this concept, the learning task becomes an optimization problem, in which the objective function is the performance achieved in games with *all* possible rival strategies. Elegant as it is, such formulation raises a serious difficulty caused by the cost of objective function calculation. Indeed, an explicit construction of such a function is intractable for most games due to a vast number of possible opponents.

This difficulty is typically overcome by dropping the objective function in favor of an evaluation function that is computationally cheaper and steers the search algorithm toward the assumed goal. In the case of games, such an evaluation function limits the number of opponents and can be static or dynamic. A static evaluation function employs a fixed set of predefined expert players and might be used as a driving force for, e.g., evolutionary learning (EL). A dynamic function uses a set of opponents that changes together with the learning players, and is typically employed in self-learning methods such as *coevolutionary algorithms*. By exposing a learner to more different opponents, learning algorithms that use dynamic evaluation functions can be expected to produce players that win against a wider range of opponents. On the other hand, dynamic evaluation functions can be confusing for an algorithm and lead to undesired phenomena such as coevolutionary pathologies [27], [43].

In this work, we explore the key issues pertaining to the above questions about search space and search goal in the context of learning strategies for the board game of *Othello*. In particular, we investigate how the size of the search space and the type of evaluation function influence the performance of evolutionary, coevolutionary, temporal difference learning (TDL) algorithms, and their hybrids introduced in our previous works

Manuscript received February 04, 2012; revised August 10, 2012 and December 20, 2012; accepted April 11, 2013. Date of publication April 18, 2013; date of current version September 11, 2013. The work of M. Szubert and K. Krawiec was supported by the Ministry of Science and Higher Education under Grant N N519 441939. The work of W. Jaśkowski was supported by the Ministry of Science and Higher Education under Grant 91-531/DS.

The authors are with the Institute of Computing Science, Poznań University of Technology, Poznań 60965, Poland (e-mail: mszubert@cs.put.poznan.pl; kkrawiec@cs.put.poznan.pl; wjaskowski@cs.put.poznan.pl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2013.2258919

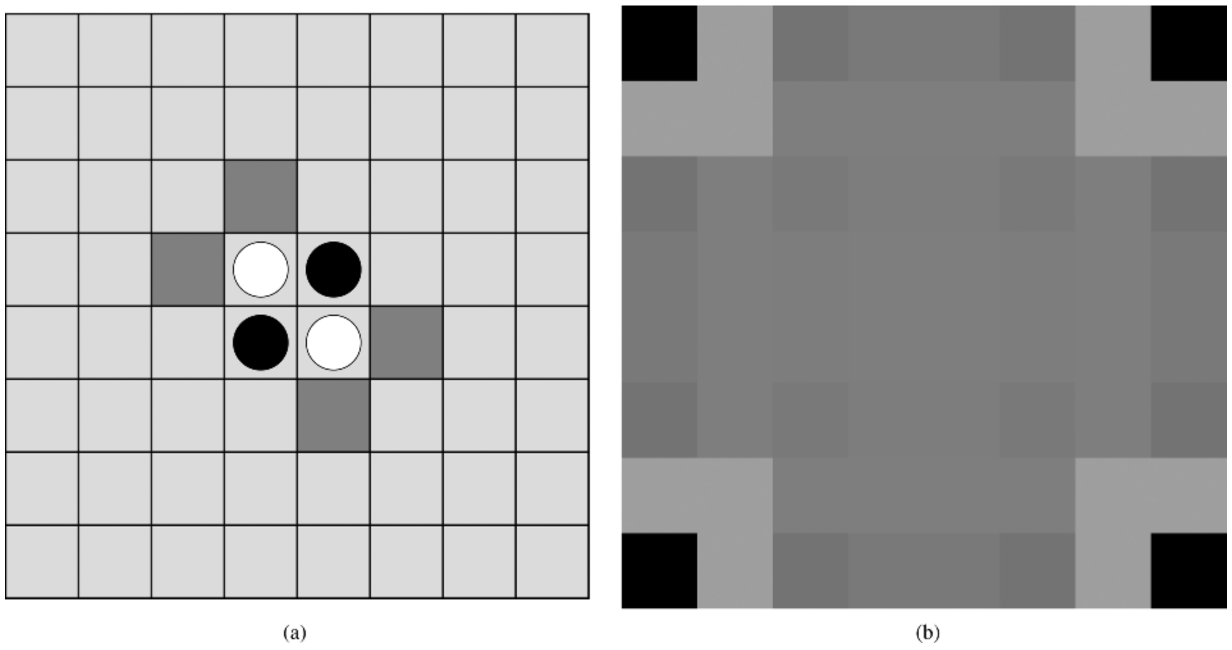


Fig. 1. *Othello* board and its coloring according to heuristic player weights (darker color/greater weight). (a) *Othello* initial board state. (b) Heuristic WPC weights.

[36], [15]. This allows us also to discuss the benefits of global and local search hybridization. We demonstrate that changing the size of representation, which for the sake of this study is an n-tuple network, has a significant impact on learning results achieved by particular methods. Some algorithms scale up well and easily utilize the opportunities offered by the larger search space, while the performance of others tends to deteriorate. Most importantly, by gauging the results by means of different performance measures that involve different samples of opponent strategies, we can verify which of the considered approaches generalizes better and is more capable of producing strategies that can defeat a variety of strong opponents.

Overall, the major contributions of this paper include: 1) experimental evidence that a coevolutionary learning (CEL) algorithm can provide itself with a sample of opponents/trainers that are diversified enough to make the resulting strategies superior to all other state-of-art learning algorithms considered in this study; 2) demonstration of the synergy of combinatorial search performed by coevolution with the gradient-based search carried out by TDL, in particular TDL's capability to support coevolution in highly dimensional search spaces; 3) investigation into the impact of representation size; and, last but not least; and 4) demonstration of discrepancies between players' assessments obtained using various performance measures.

II. The Game of *Othello*

Othello is played by two players on an 8×8 board. Typically, pieces are disks with white and black faces, each color representing one player. Fig. 1(a) shows the initial state of the board; each player starts with two pieces in the middle of the grid. The black player moves first, placing a piece, black face up, on one of four shaded locations. Players make moves alternately until no legal moves are possible.

A legal move consists of placing a piece on an empty square and flipping appropriate pieces. To place a new piece, two conditions must be fulfilled. First, the position of the piece must be adjacent to an opponent's piece. Second, the new piece and some other piece of the current player must form vertical, horizontal, or diagonal lines with a contiguous sequence of opponent's pieces in between. After placing the piece, all such opponent's pieces are flipped; if multiple lines exist, flipping affects all of them. A legal move requires flipping at least one of the opponent's pieces. Making a move in each turn is obligatory, unless there are no legal moves. The game ends when both players have no legal moves. The player who has more disks at the end of the game wins; the game can also end with a draw.

A. Strategy Representation

There are two common ways in which game-playing strategies can be represented, namely, as *a move selector* or as a *state evaluator* (also referred to as a *position evaluator*). A move selector takes the current game state as an input and returns a move to be made. A state evaluator, on the other hand, is used to estimate how beneficial a given state is for the player. With the help of a game tree search algorithm, this allows for selecting the move that leads to the most favorable afterstate.

Most recent works on learning *Othello* strategies have focused on creating board evaluation functions [24], [26], and we follow that trend in this study. Moreover, we focus our research on comparison between learning procedures rather than developing efficient tree search algorithms. For this reason, to select a move during the game, we evaluate all states at 1-ply—when a player is to make a move, it expands the current game state to all possible direct afterstates and evaluates each of them using player's evaluation function. The move that yields the highest return value is selected.

B. The Othello League

A good overview of different *Othello* player architectures and their estimated performance is provided by the *Othello* Position Evaluation Function League [22]. The player's rank in the league is based on the score obtained in 100 games played at 1-ply (in which 10% of moves are forced to be random) against the standard heuristic weighted piece counter (WPC) player. WPC is a simple architecture, which may be viewed as an artificial neural network comprising a single linear neuron with inputs connected to all board locations. It assigns a single weight to each location and calculates the utility of a given board state by multiplying weights by color-based values of the pieces occupying corresponding locations. The standard heuristic player developed by Yoshioka *et al.* [44] is illustrated in Fig. 1(b). We also use it in our experiments as one of the opponents to measure the performance of evolved strategies.

Regarding the league results, all the best players submitted to the competition are based on more complex architectures than WPC. Examples of such architectures, which often operate in a nonlinear fashion and involve numerous parameters are: a symmetric n-tuple network, a multilayer perceptron (MLP), and a spatial MLP. At the time of writing, the league all-time leader was an n-tuple network with a winning percentage of just below 80%. Such results can be achieved by a self-play training with TDL (see Section III-A) [20]. In our research, we use the same architecture to compare performance acquired by different learning methods. To make the comparison more informative, we used the best players from the league as opponents in a series of round-robin tournaments.

C. *The n-Tuple Network Architecture*

The idea of n-tuple systems was originated by Bledsoe and Browning [3] for use in character recognition. Since then it has been successfully applied to both classification [30] and function approximation tasks [14]. Their main advantages include conceptual simplicity, speed of operation, and capability of realizing nonlinear mappings to spaces of higher dimensionality. Following significant research on using n-tuple classifiers for handwritten digits [21] and face recognition problems [23], recently, Lucas proposed employing the n-tuple architecture also for game-playing purposes [20].

An n-tuple system expects as input some compound entity (matrix, tensor, image) $\mathbf{x}$, which elements (usually scalar variables) can be retrieved using some form of coordinates. An n-tuple network operates by sampling that input object with m n-tuples. The ith n-tuple t_i is a sequence of n variables a_{ij}, $j = 0 \ldots n-1$, $i = 0 \ldots m-1$, each corresponding to predetermined coordinates in the input. Assuming that each variable takes on one of v possible values, an n-tuple can be viewed as a template for an n-digit number in base-v numeral system. When a specific input $\mathbf{x}$ is given, it assumes one of v^n possible values. The number represented by the n-tuple t_i is used as an index in an associated lookup table LUT_i, which contains parameters equivalent to weights in standard neural networks. For a given input $\mathbf{x}$, the output of the n-tuple network can be calculated as

$$f(\mathbf{x}) = \sum_{i=0}^{m-1} f_i(\mathbf{x}) = \sum_{i=0}^{m-1} \mathrm{LUT}_i \left[\sum_{j=0}^{n-1} \mathbf{x}(a_{ij}) v^j \right] \tag{1}$$

where $\mathbf{x}(a_{ij})$ denotes retrieving from $\mathbf{x}$ the element located at the position indicated by a_{ij}.

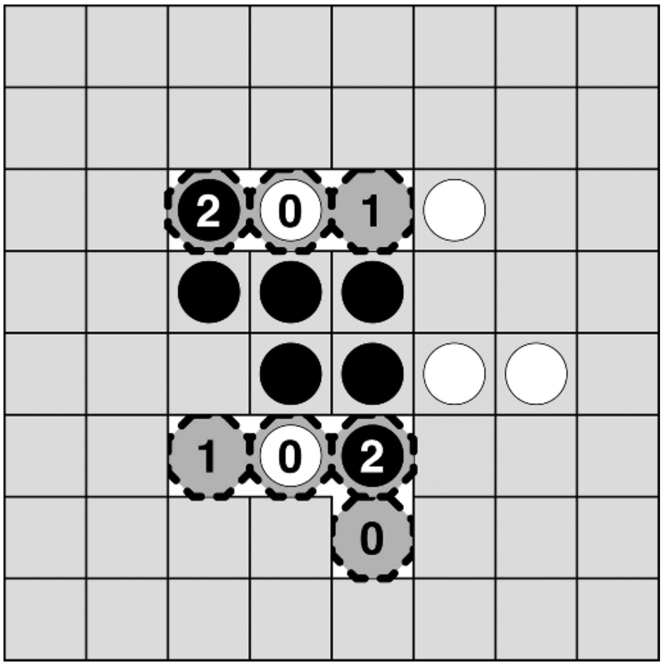


Fig. 2. Two sample n-tuples viewed as templates for base-3 numbers. Each input location represents a ternary digit. Multiplying them by successive powers of 3 leads to decimal values of $2 \times 3^2 + 1 \times 3^0 = 19$ and $1 \times 3^3 + 2 \times 3^1 = 33$, which are used as indexes in the associated lookup tables.

In the context of *Othello*, an n-tuple network acts as a state evaluation function. It takes a board state as an input $\mathbf{x}$ and returns its utility. Input variables are identified with coordinates on the board, and the value retrieved from a single location is 0, 2, or 1 if, respectively, it is occupied by a white piece, a black piece, or is empty. Consequently, an n-tuple represents a ternary number which is used as an index for the associated lookup table containing 3^n entries (see Fig. 2). Additionally, symmetric sampling (introduced in [20]) can be incorporated; a single n-tuple is employed eight times, once for each possible board reflection and rotation. LUT values indexed by all such equivalents are summed together to form the output of the particular n-tuple. The final value of a board is simply the sum of all n-tuple outputs [see (1)].

The number of possible n-tuple instances is exponential in function of the size of $\mathbf{x}$ and n, so assigning the initial input variables (board locations) to n-tuples is an important design issue. Typically, in pattern recognition applications, the simplest approach of random selection is commonly used. However, in the context of games, the spatial neighborhood of chosen locations is intuitively appealing. For this reason, and particularly for *Othello*, connected sets of locations like a straight line or a rectangle area are usually chosen. In our implementation, we allowed for more flexible assignments in the form of *snake* shapes, proposed by Lucas [20]. For each n-tuple, we choose a random square on the board from which a random walk of $n-1$ steps in any of the maximum eight possible directions is taken. Other implementation details concerning n-tuples are presented in Section IV-A.

D. *Previous Research on Computer Othello*

The game of *Othello* has been a subject of artificial intelligence research for more than 20 years. The significant interest in this game may be explained by its simple rules, large state-space cardinality (around 10^{28}), and high divergence rate causing that it remains unsolved—a perfect *Othello* player has not been developed yet. For all these reasons, it remains an excellent benchmark for learning algorithms and player architectures.

Conventional *Othello*-playing programs are based on a thorough human analysis of the game, leading to sophisticated handcrafted evaluation functions. They often incorporate supervised learning techniques that use large expert-labeled game databases and efficient look-ahead game tree search. One of the first examples representing such an approach was BILL [18]. Besides using precomputed tables of board patterns, it employed Bayesian learning to build in so-called *features* into an evaluation function. Today, one of the strongest *Othello* programs is Logistello [4], which makes use of advanced search techniques and applies several methods to learn from previous games. Its evaluation function is based on a predefined pattern set including horizontal, vertical, and diagonal lines, as well as special patterns covering edges and corners of the board. Pattern configurations correspond to binary features and have associated values. Evaluating a board consists in summing values of occurring features, and thus, is very similar to calculating the value of an n-tuple network.

Recently, the mainstream research on *Othello* has moved toward better understanding of what types of learning algorithms and player architectures work best. The Congress on Evolutionary Computation (CEC) *Othello* Competitions [22] pursued this direction by limiting the ply depth to one, effectively disqualifying the algorithms that employ a brute-force game tree search.

The most challenging scenario of elaborating a game strategy is learning without any support of human knowledge or opponent strategies given *a priori*. This task formulation is addressed by, among others, TDL and CEL, which were applied to *Othello* by Lucas and Runarsson [24]. Other examples of using self-learning approaches for *Othello* include coevolution of spatially aware MLPs [5], TD-leaf learning of structured neural networks [41], coevolutionary TDL (CTDL) [36], and Nash memory applied for coevolved n-tuple networks [26]. That study inspired our previous paper [36], in which we compare these methods with their direct hybridization called CTDL.

III. Methods

A. Temporal Difference Learning

Since the influential work of Tesauro [39] and the success of his *TD-Gammon* player trained through self-play, TDL [33] has become a well-known approach for elaborating game strategies without help from human knowledge or expert strategies given *a priori*.

The use of reinforcement learning techniques for such applications stems from modeling a game as a sequential decision problem, where the task of the learner is to maximize the expected reward in the long run (game outcome). The essential feature of this scenario is that the actual (true) reward is typically not known before the end of the game so some means are necessary to propagate that information backwards through the series of states, assign credit to particular decisions, and guide the intragame learning.

The TD(0) algorithm solves prediction learning problems that consist in estimating the future behavior using the past experience. Its goal is to make the preceding prediction match more closely to the current prediction (taking into account distinct system states observed in the corresponding time steps). Technically, the prediction at a certain time step t can be considered as a function of two arguments: the outcome of system observation and the vector of modifiable weights $\mathbf{w}$, which are updated by the following rule:

$$\Delta \mathbf{w}_t = \alpha(P_{t+1} - P_t)\nabla_w P_t. \tag{2}$$

In our case, P_t is realized by an n-tuple network (1) whose outputs are squeezed to the interval $(-1, 1)$ by hyperbolic tangent. Using such a prediction function within TD(0) update rule (2) results in changing LUT weights according to

$$\Delta\text{LUT}_i \left[\sum_{j=0}^{n-1} \mathbf{x}(a_{ij})v^j\right]_t = \alpha(P_{t+1} - P_t)\left(1 - P_t^2\right).$$

This formula modifies only those LUT entries that correspond to the elements of the board state selected by the n-tuple, i.e., with indices generated by the n-tuple. If the state observed at time $t + 1$ is terminal, the exact outcome of the game is used instead of the prediction P_{t+1}. The outcome is $+1$ if the winner is black, -1 if the winner is white, and 0 when the game ends in a draw.

The process of learning consists in applying the above formula to the LUT entries after each move, i.e., with P_t and P_{t+1} being, respectively, the network output before and after a move. The training data for that process, i.e., a collection of games, each of them being a sequence of states $\mathbf{x}_1, \mathbf{x}_2, \ldots$, are acquired in a method-specific way (e.g., via self-play). During training games, moves are selected on the basis of the most recent evaluation function.

Othello is a deterministic game, thus the course of the game between a particular pair of deterministic players is always the same. This feature reduces the number of possible game trees that a learner interacts with and explores, which makes learning ineffective. To remedy this situation, at each turn, a random move is forced with a probability ϵ (this is known as ϵ-greedy policy [34]). After such a random move, the learning algorithm does not update weights. Thanks to random moves, players are confronted with a wide spectrum of possible behaviors of their opponents.

B. Evolutionary and Coevolutionary Learning

The TDL approach is a gradient-based local search method that maintains a single solution and as such may not be able to escape from local optima [38]. Evolutionary computation [2], a global search neo-Darwinian methodology of solving learning and optimization problems, has completely opposite characteristics; it maintains a population of candidate solutions (individuals), but has no means for calculating individually adjusted corrections for each solution parameter. It lessens the problem of local optima by its implicit parallelism and random modification of candidate solutions. Consequently, evolutionary computation seems to be an attractive complementary alternative for TDL for learning game strategies.

The fitness function is an indispensable component of evolutionary computation that drives the search process by assigning fitness values to candidate solutions. It is also the fitness function that constitutes the major difference between *evolutionary* and *coevolutionary* algorithms. While in evolutionary algorithms this function is expected to be *static* and reflect an individual's absolute performance, coevolutionary algorithms employ *dynamic* fitness functions that assess the relative performance of individuals with respect to other individuals in the population.

However, one faces substantial difficulty when designing an absolute fitness function for the task of learning game strategies. A truly objective assessment of an individual's utility in case of games can be done only by playing against *all* possible opponent strategies. For the majority of games, this is computationally intractable. An alternative is to consider only a limited number of opponents, and thus, lessen the computational burden. In this case, the sample of opponents used for evaluation could be formed by a predefined expert player(s) or a sample of random opponents; such an approach was recently found successful [6].

In this context, coevolution is an appealing alternative that offers a natural way of designing fitness function. Indeed, relative performance of individuals is calculated on the basis of the results of their interactions with other population members. In learning game strategies, an interaction consists in playing a game and increasing the fitness of the winner while decreasing the fitness of the loser. It means that individuals just play games with each other in a round-robin fashion, and the outcomes of these interactions determine their fitness values. This evaluation scheme is termed as competitive coevolution [1].

EL and CEL of game strategies used in this study follow the above ideas and typically start with generating a random initial population of player individuals. Individuals are evaluated with a static or dynamic fitness function, respectively. The best performing strategies are selected, undergo genetic modifications such as mutation or crossover, and their offspring replace some of (or all) former individuals. In practice, this generic scheme is supplemented with various details, which causes EL and CEL to embrace a broad class of algorithms that have been successfully applied to many two-person games, including *Backgammon* [29], *Checkers* [11], and a small version of *Go* [31]. In particular, Lucas and Runarsson used $(1+\lambda)$ and $(1,\lambda)$ evolution strategies in a competitive environment to learn a strategy for the game of *Othello* [24].

C. Hybrid Learning

The past results of learning WPC strategies for *Othello* [24] and small-board *Go* [31] demonstrate that TDL and CEL exhibit complementary features. CEL progresses slower, but, if properly tuned, eventually outperforms TDL. With respect to learning n-tuple networks, though, CEL is reported to be less successful, while TDL confirms its strength [20]. Still, it sounds reasonable to combine these approaches into a hybrid algorithm exploiting different characteristics of the search process performed by each method. In our previous works [36], [16], a method termed CTDL was proposed and applied to learn WPC strategies. CTDL maintains a population of players and alternately performs TDL and CEL. In the TDL phase, each player is subject to TD(0) training. Then, in the CEL phase, individuals are evaluated on the basis of a round-robin tournament. Finally, a new generation of individuals is obtained using selection and variation operators, and the cycle repeats. The idea realized by this method can be called coevolutionary gradient search [17]. The overall conclusion was positive for CTDL, which produced strategies that, on average, defeated those learned by TDL and CEL. Encouraged by these results, we wonder whether CTDL would prove beneficial also for more complex n-tuple network architectures. Additionally, for the sake of completeness, we introduce also an analogous hybrid approach of evolutionary temporal difference learning (ETDL), which in an analogous way combines EL and TDL.

It is worth noting that hybridization of EL and TDL can be considered as a form of memetic algorithm. Memetic algorithms [28] are hybrid approaches coupling a population-based global search method with some form of local improvement. Since these algorithms usually employ evolutionary search, they are often referred to as Lamarckian evolution, to commemorate Jean-Baptiste Lamarck who hypothesized, incorrectly in the view of today's neo-Darwinism, that the traits acquired by an individual during its lifetime can be passed on to its offspring. Technically, memetic algorithms typically alternate genetic search for the population and local search for individual solutions.

Other hybrids of TDL and CEL or EL have been occasionally examined in the past. Kim *et al.* [13] trained a population of neural networks with TD(0) and used the resulting strategies as an input for the standard genetic algorithm with mutation as the only variation operator. In [26], a coevolutionary algorithm is combined with TDL used as a weight mutation operator and applied to the game of *Othello*. Contrary to the approach presented here that uses straightforward coevolution with no long-term memory mechanism, Manning [26] employed the Nash memory algorithm [10] with bounded archives.

IV. Experimental Setup

All algorithms presented above were implemented using our coevolutionary algorithms library called cECJ [35], built upon evolutionary computation in Java (ECJ) framework [25]. Our unit of computational effort is a single game, and the time of other operations is neglected. To provide a fair comparison, all runs were stopped when the number of games played reached 3 000 000. Each experiment was repeated 24 times.

A. Player Architecture

We rely on n-tuple networks because of its appealing potential demonstrated in recent studies [26], [20] and promising results in the *Othello* League [22]. We start from small networks formed by seven instances of 4-tuples (7×4), which include 567 weights. Later, we move to 9×5 networks (2187 weights on aggregate) to end up with the largest 12×6 architecture (8748 weights) that has recently been successfully applied to

Othello by Manning [26]. This progression enables us to observe how particular methods cope with the growing dimensionality of the search space.

We decided to employ the input assignment procedure that results in randomly placed snake-shaped tuples (see Section II-C). Regarding the LUT weights, their initial values depend on the particular learning algorithm. As previous research shows [37], TDL learns faster when it is 0-initialized. Evolutionary methods, on the other hand, assume that the population is randomly dispersed in the search space. For this reason, in the purely coevolutionary algorithm (i.e., without TDL), we start from weights initialized randomly in the $[-1, 1]$ range.

B. Search Operators

The considered search heuristics operate in two spaces: a discrete network topology space and a continuous weight space. Dimensions of the topology space are: the number of tuples, their size, and input connections. Dimensionality of the weight space depends directly on the number of weights and grows exponentially with tuple length. We search both spaces in parallel as it gives the learner more flexibility than searching only one of them. However, to avoid excessive complexification, we limit topology changes just to input assignment; the number of n-tuples and their length stay the same throughout learning. Although the majority of methods applied to train neural networks are based on a fixed structure and search only the weight space, there are some exceptions which explore topology space as well [32], [40].

In accordance with the twofold nature of this search space, we employ two types of operators: genetic and gradient based. Let us note that the former ones rely on the direct encoding of strategies, i.e., the individual's genome is a concatenation of LUT weights associated with its n-tuples. Overall, we use the following genetic operators:

- *weight mutation*: each weight (LUT entry) with probability $p_{mw} = 0.05$ undergoes Gaussian mutation ($\sigma = 0.25$);
- *topology mutation*: each input (board location) is replaced, with probability $p_{mt} = 0.01$, by another input from its neighborhood;
- *topology crossover*: sexual reproduction with probability $p_x = 1.00$; two individuals mate and exchange genes, i.e., entire tuples with LUTs; an offspring inherits $m/2$ randomly selected tuples from each.

The only gradient-based operator works in the weight space and consists in running a single self-play game incorporating TD(0) algorithm (see Section III-A). We use learning rate $\alpha = 0.001$ and force random moves with probability $\varepsilon = 0.1$.

C. Learning Algorithms

TDL searches only the weight space using a single network and self-play TD(0) as the only search operator.

EL is a generational evolutionary algorithm with a population of 50 individuals. The algorithm operates in a well-recognized loop of: 1) evaluation: the fitness of each individual is calculated as a sum of points obtained in 50 randomized games against the WPC-heuristic player; 2) selection: evaluated individuals are subject to tournament selection [12] with tournament size 5; 3) recombination: individuals undergo topology crossover; and 4) mutation: individuals are modified by weight and topology mutation.

CEL is a generational coevolutionary algorithm with a population of 50 individuals. The algorithm operates in a similar fashion to EL, except for the evaluation phase, where a round-robin tournament is played between all individuals, with wins, draws, and losses rewarded by 3, 1, and 0 points, respectively, and the total number of points becomes the individual's competitive fitness. For each pair of individuals, two games are played, with players swapping the roles of the black and white players.

ETDL combines EL and TDL, as described in Section III-C. Similarly to EL, it uses topology mutation and topology crossover, but instead of weight mutation, it employs self-play TDL training. By default, in each TDL phase, a budget of 5000 training games is allocated to the players in the population. Thus, each individual plays 100 games during this learning phase.

CTDL combines CEL and TDL, as described in Section III-C. The algorithm operates as ETDL but uses competitive fitness like CEL.

Notice that CTDL extends CEL in the same way as ETDL extends EL. Moreover, CEL and CTDL (also, EL and ETDL) differ only in the way they search the weight space (weight mutation versus TDL).

Furthermore, where possible, the parameters for the above algorithms were taken directly from our previous research [36], [16] or related works [20], [24], [26]. In some cases, the parameters were determined by preliminary experiments. This includes the value of σ for weight mutation and the number of TDL games in a single phase of hybrid algorithms. It should be emphasized though that our goal was not to find the best parameters for this particular problem, but to compare learning methods using reasonable settings.

D. Performance Measures

To monitor the progress of learning in our experiments, 50 times per run (approximately every 60 000 games), we appoint the individual with the highest fitness as the best-of-generation individual (for TDL, the single strategy maintained by the method is the best-of-generation by definition). The best-of-generation players from all runs of a method form a team. In particular experiments, the performance of a team, and indirectly of the method it represents, is calculated using the following measures (see Section V for details):

1) performance against a heuristic player, i.e., percentage score against a predefined, human-designed WPC strategy (the opponent used to rank the players in the *Othello* League; Section V-A);
2) the number of points in a round-robin tournament between the teams of best-of-generation players produced by algorithms (Section V-B);
3) the place taken in a round-robin tournament involving the best entries from the online *Othello* League (Section V-C1).

It should be emphasized that the outcomes of performance assessments are unavailable to learning algorithms and thus do not

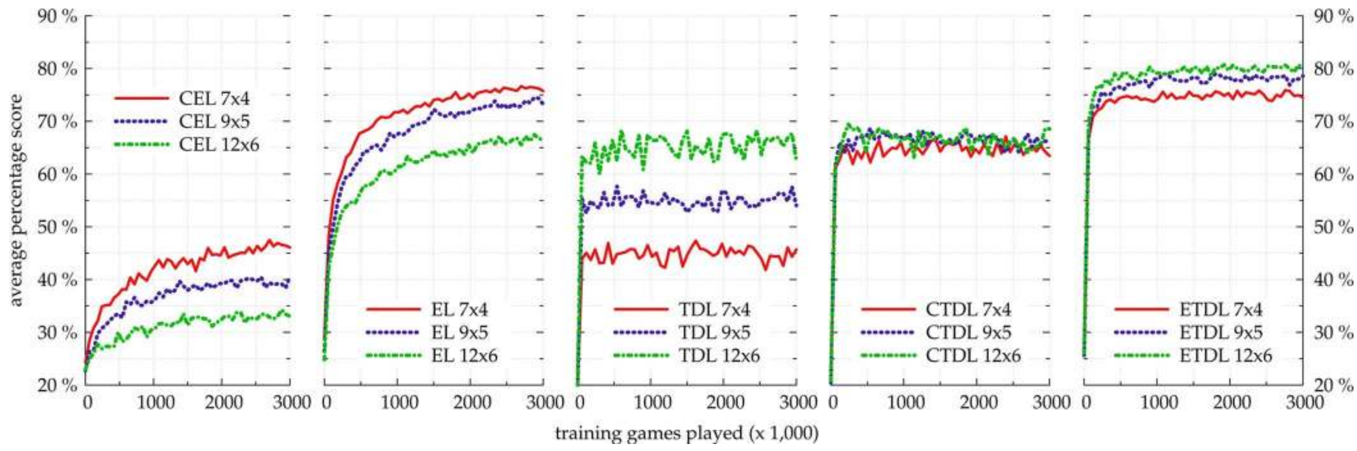


Fig. 3. Comparison of learning methods for three network sizes. Average performance of the best-of-generation individuals measured as a percentage score against the WPC heuristic.

influence the learning process. In a machine learning perspective, the opponents used in the above measures form a testing set and are intended to verify the generalization capability. The only exceptions to this rule are EL and ETDL, where fitness assessment uses the same opponent as the first performance measure.

More details on designing performance measures for game strategies can be found in the recent work of Li *et al.* on the iterated prisoner's dilemma [19].

V. RESULTS

We conducted several experiments focused on comparing how particular learning methods (Section IV-C) fare for different sizes of strategy representation (Section IV-A) with respect to particular performance measures (Section IV-D). In particular, we aimed to answer the following questions: How do the algorithms scale with the size of strategy representation (Section V-A1)? Is the performance against a heuristic player a good predictor of a player's likelihood to beat other opponents? What is the ability of the players trained using particular methods to play against new, previously unseen opponents (Section V-C1)?

A. Performance Against a Heuristic Player

This performance measure, used in previous works [26], [36] and employed in the *Othello* League to rank the contestants [22], is the percentage of points (1.0 point awarded for a win, 0.5 for a draw, calculated with respect to the maximum possible total score) obtained in 1000 games (500 as black and 500 as white) played against the WPC heuristic, a fixed player using the WPC architecture with weights graphically illustrated in Fig. 1(b). All players in our experiments are deterministic, as well as the game of *Othello* itself. Thus, in order to have a more precise estimation of the relative strength of a given trained player versus the WPC heuristic, following Lucas and Runarsson [24], we force both players to make random moves with probability $\varepsilon = 0.1$.

1) Scalability: In the first experiments, we focus on the scalability with respect to the representation size. Fig. 3 illustrates how the various methods' performance against a heuristic player changes when moving from 7×4 to 9×5 and to 12×6 n-tuple networks. The plots show the performance as a function of the total number of training games played, i.e., the games required by fitness calculation (either absolute or relative) as well as the games played in the TDL phase (where applicable). As game playing is the most costly component of all considered algorithms, this comparison is fair in terms of computational effort.

Interestingly, increasing the network size is not necessarily beneficial for all tested methods. Only TDL is able to significantly improve its performance by utilizing the possibilities offered by larger networks. On the contrary, EL and CEL perform even worse with larger networks than with the smaller ones. For the largest 12×6 networks, CEL gains barely a few percent within the entire learning process. We hypothesize that the weight mutation operator is not sufficiently efficient to elaborate fast progress in the larger (higher dimensional) weight search space. This hypothesis is supported by all plots; only the methods involving weight mutation (EL and CEL) have such problems.

To make sure that this is not due to possibly unfavorable settings of weight mutation, we performed another experiment with different standard deviations (σ) of weight mutation. Results for EL with network sizes 7×4 and 12×6 presented in Fig. 4 show that our choice $(\sigma = 0.25)$ is among the best values of deviation. Importantly, no matter what value of σ is used, the performance is lower with the larger networks. Conversely, when no weight mutation is used $(\sigma = 0)$, larger networks allow for achieving better results. In this case, the weights remain unchanged, and the evolutionary process modifies only the topologies of networks. Although this implies that weights remain fixed for an entire evolutionary run, and, therefore, the total number of strategies that can be represented by individuals is more limited, the resulting search problem is easier and evolution eventually benefits from the larger network size.

Finally, the hybrid methods work either on a par with this performance measure (CTDL) or slightly better with larger networks (ETDL). Apparently, using the TDL method to search the weight space of n-tuple networks is a better idea than applying random mutations, especially when the search space is larger. Hybridization allows evolutionary components to focus

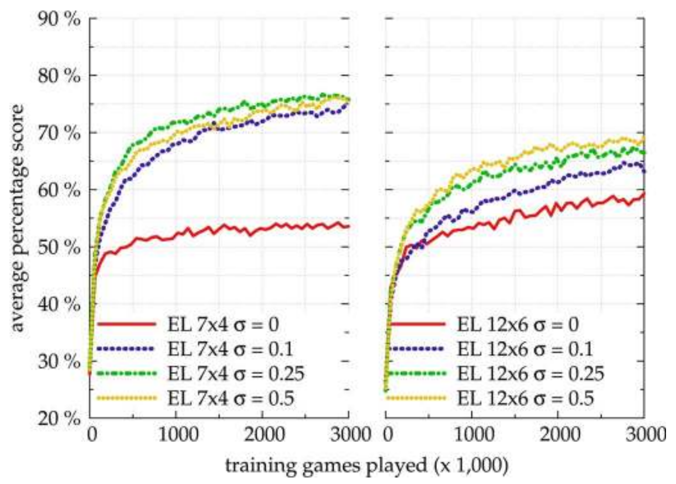


Fig. 4. EL with different σ of weight mutation for 7×4 and 12×6 networks.

entirely on searching the topology space while leaving the continuous weight space to a dedicated gradient-based algorithm, which works well on this problem when applied separately.

2) Method Comparison: After answering the question whether larger representations pay off, we ask which method works best. Fig. 5 compares all the methods for the three considered representation sizes. The results for the smallest 7×4 network confirm our previous findings [36] for the WPC strategy representation: the CTDL hybrid in the long run significantly outperforms the nonhybrid algorithms, TDL and CEL. Moreover, as also observed in previous research [24], TDL learns rapidly, whereas CEL advances slower but eventually reaches a similar or even slightly higher performance level.

However, while the superiority of CTDL is still observable for 9×5 networks, for the 12×6 ones, there is no difference between TDL and CTDL, which both score between 65% and 70%. This level is similar to that reported by Manning [26] for 12×6 networks trained by a complex Nash memory approach (between 66% and 68%). This indicates a ceiling effect [42] in the evaluation of self-learning methods with the WPC-heuristic performance measure. We hypothesize that the randomized WPC-heuristic player does not offer sufficiently diversified challenge to differentiate the strategies produced by these algorithms. To verify this claim and to differentiate the algorithms in terms of their performance, we conducted a series of performance assessments on a pool of opponents, detailed in Sections V-B and V-C1.

Let us note that the above ceiling effect should not be interpreted in absolute terms. The plots clearly show that the WPC heuristic managed to differentiate the performance of evolutionary algorithms (EL and ETDL) versus the other ones (TDL and CTDL), with the former performing better or equal. This is, however, not surprising, given that the former methods have been guided by the WPC heuristic. As we will demonstrate in subsequent sections, these observations tell us very little about the performance of the trained players on another, more sophisticated, sample of opponents.

Last but not least, let us note that ETDL performs better than EL for larger representations. This is further evidence that supports our claim that TDL mutation is much more efficient than weight mutation.

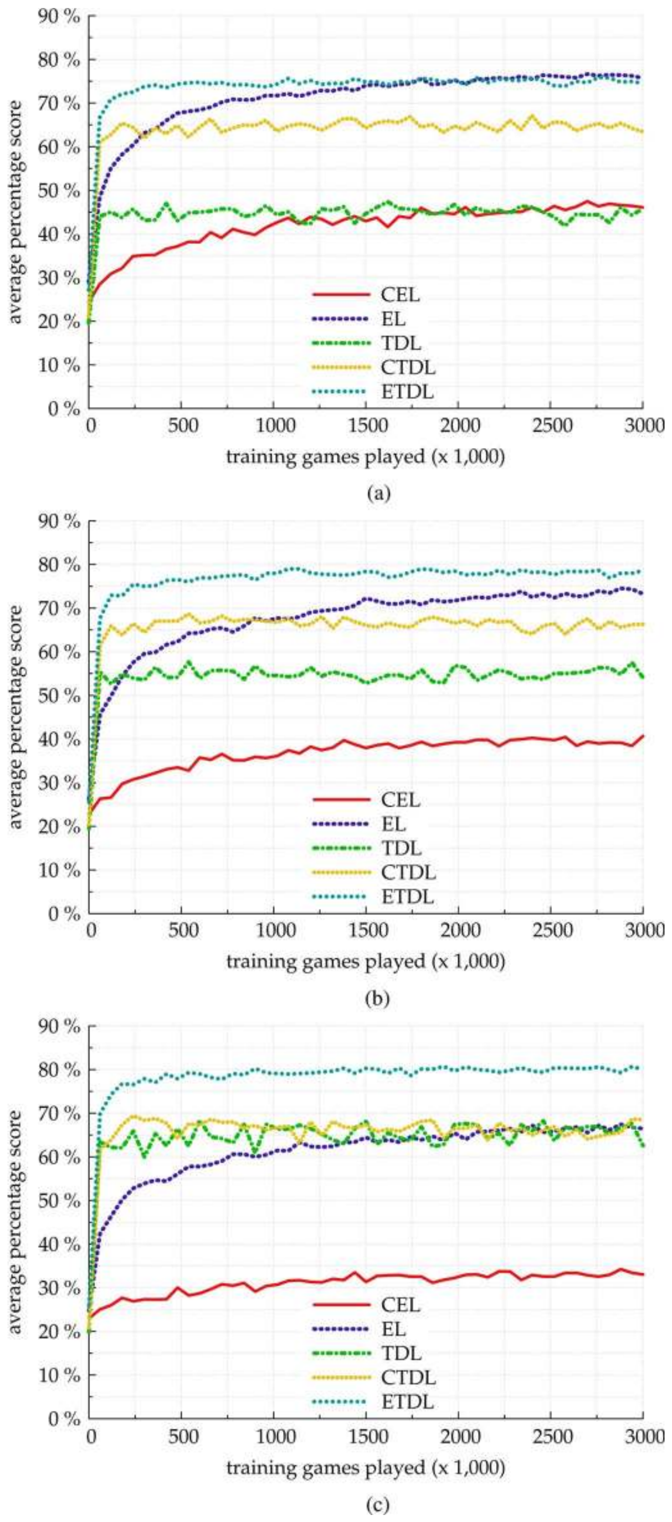


Fig. 5. Comparison of learning methods for three network sizes. Average performance of the best-of-generation individuals measured as a percentage score against the WPC heuristic: (a) 7×4 n-tuple network; (b) 9×5 n-tuple network; and (c) 12×6 n-tuple network.

B. Generational Round-Robin Tournament

The handcrafted heuristic strategy, even when randomized, cannot be expected to represent in full the richness of possible

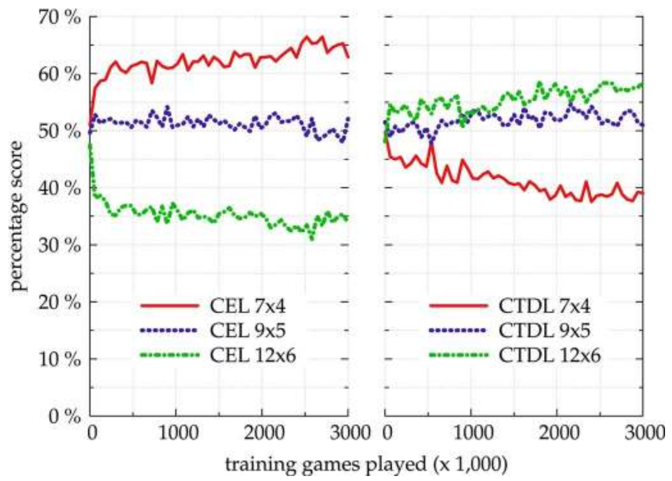


Fig. 6. Generational round-robin tournament for CEL and CTDL using different network sizes.

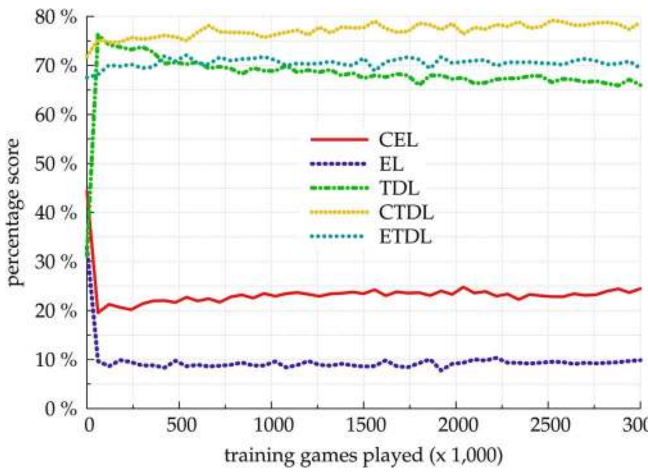


Fig. 7. Generational round-robin tournament for all methods using 12×6 networks.

behaviors of *Othello* strategies. To avoid the ceiling effect caused by a too narrow range of assessment opponents, we let the considered methods generate assessment opponents for each other. Technically, every 60 000 games played we create teams that embrace all the best-of-generation strategies found by 24 runs of particular methods. Next, we play a round-robin tournament between the teams representing particular methods, where each team member plays against all members from the opponent teams. The score of a team is the overall sum of points obtained by its players. As the tournaments are played multiple times during learning, we call this method the generational round-robin tournament.

Note that this assessment scheme is relative: gain for one team implies loss for others. A team can be judged good due to its virtues, but also due to the weaknesses of other teams. As another advantage, the generational round-robin tournament allows us to drop the randomization of moves, since the presence of multiple strategies in the opponent team provides enough behavioral variability.

Fig. 6 plots the relative performance of the CEL and CTDL algorithms for different network sizes. The score is given in percents; a method can maximally obtain 100%, which means that it wins all games. In this confrontation, CEL 7×4 not only beats CEL with larger networks, but its advantage even increases with learning time. This confirms our results obtained for the WPC-heuristic performance measure: CEL has difficulties in coping with larger search spaces. On the other hand, when weight mutation is replaced by a TDL operator (CTDL), larger representations enable achieving better strategies. It is interesting to compare the latter figure with Fig. 3, in which CTDL seems indifferent to network size. This supports the ceiling effect hypothesis; differences between certain methods cannot be uncovered using the WPC-heuristic performance measure.

Fig. 7 plots the relative performance of all the algorithms using the 12×6 network. Again, the players produced by TDL, CTDL, and EL, which played at the same level against the WPC heuristic [cf. Fig. 5(c)] turn out to generate players of diametrically different competence when compared on a different pool of assessment opponents. As these opponents uncovered previously unobserved differences between methods and have been trained using diametrically different algorithms (as opposed to WPC-heuristic opponents that differ only in the randomized moves), we hypothesize that they are behaviorally more diversified. Verifying this claim would, however, require an additional analysis that is beyond the scope of this paper. Let us emphasize that the teams confronted here are composed of the same best-of-generation individuals that produced the results reported in Fig. 5(c), i.e., we assess here the outcomes of the same runs of learning algorithms.

In the tournament confrontation, the CTDL hybrid is clearly the winner and beats its constituent methods, TDL and CEL. Also, its advantage over the competitors increases over time. What is, however, more interesting, is that CTDL defeats ETDL, which supports our intuition expressed in Section V-A2 that ETDL tends to overfit: it performs best against the WPC heuristic, but fails when faced with another set of players that is likely to be behaviorally more diversified. On the other hand, ETDL is still quite good, and, in particular, better than TDL. We cannot say the same about EL, which wins only around 10% of games. The self-learning TDL component of ETDL reduces the negative effects of overfitting.

C. *Othello League*

1) Generational Othello League Tournament: One of the goals we were heading toward in this study was to create a strategy that would win in a direct confrontation with the best entries in the *Othello* League [22]. Thanks to the courtesy of the league organizers, we were provided with strategies submitted to the league by anonymous contestants. We selected the top 14 strategies (from several hundreds submitted to the league) to form a pool of opponents. Each of our best-of-generation players was assessed by adding it to the pool and playing a deterministic round-robin tournament among all 15 strategies, with wins, draws, and losses rewarded by 3, 1, and 0 points, respectively. Note that each player faces every other player twice: once as black and once as white.

Fig. 8 shows the performance of our players expressed as a percentage of the maximum score possible to attain when confronted with the league pool. Each method (represented by 24 best-of-generation players) could obtain maximally $14 \times 24 \times$

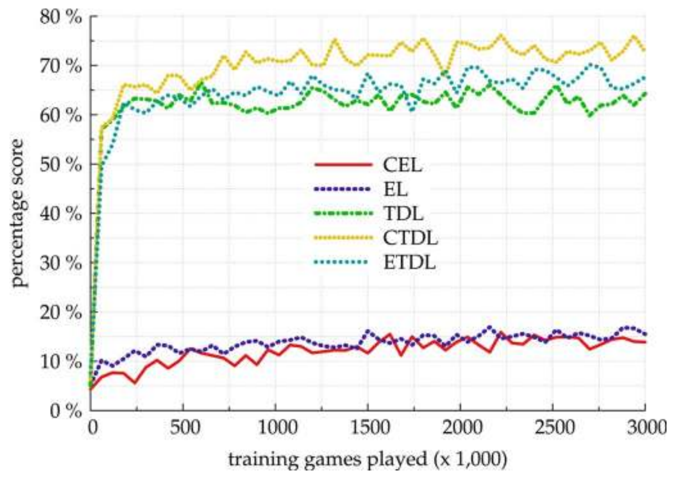


Fig. 8. Generational *Othello* League tournament. Percentage score is the score of a team of best-of-generation individuals obtained against the team formed by the league players, normalized by the maximum possible score.

$3 \times 2 = 2016$ points (100%). It is easily noticeable that this assessment ranks the methods roughly in the same order as in the generational round-robin tournament (cf. Fig. 5). Once again we can observe that ETDL turns out inferior to CTDL when the opponents are different from the strategy it was taught with. CTDL scores approximately 5%–10% more points. Methods that use weight mutation instead of TDL do not perform well.

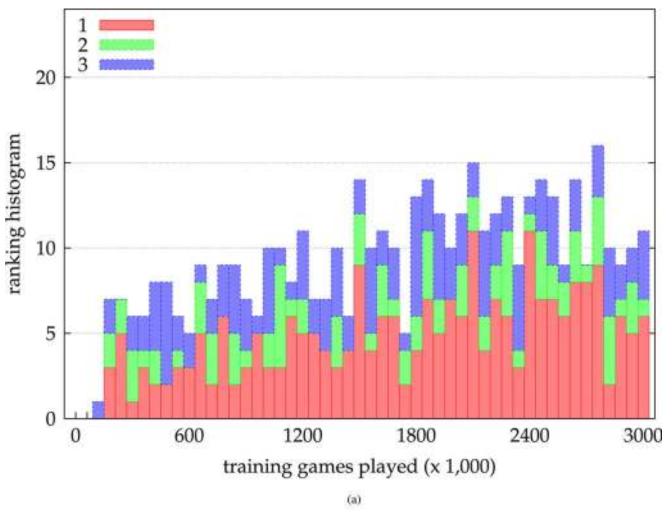
The total number of points is a valuable relative performance measure, but it does not inform us about the absolute places taken in the tournament by our best-of-generation players. Fig. 9(a) and (b) shows how many times the 24 players produced by, respectively, ETDL and CTDL rank among the top three players of the league. Clearly, the coevolutionary approach leads to winning the tournament much more often than the evolutionary method.

2) ETDL Players in Othello League: The above experiments have shown that the players produced by ETDL are less versatile than the ones produced by CTDL. However, when evaluated against the WPC heuristic, ETDL appears remarkably successful. As we could see in Fig. 5(c), in an average run, it attained a performance level of 80%. Moreover, one of the runs produced a player that reached 87.1% and took the lead when submitted to the online *Othello* League [22] under the name epTDLmpx_12 × 6. Table I shows the results of the top ten entries in the league at the time of writing.[1] All players in the table are based on the same n-tuple network architecture, but of various sizes.

D. Analysis of Network Topology

Besides measuring and comparing the performance of the learning algorithms, we were also interested in the internal representation of the best strategies. For this purpose, we examined the topologies of produced n-tuple networks and gathered statistics on the best-of-run players evolved by the CTDL method. Fig. 10 demonstrates how many times a particular field of the *Othello* board was covered by the tuples of the best players. The shades reflect the number of times a field appeared in networks. Note that the frequency pattern is reminiscent of the configuration of weights in the WPC-heuristic player illustrated in Fig. 1(b). Certainly, tuples cumulate around corners, which appear to be the most important fields on the board. Also, topology mutations pressured networks to abandon central fields which, on the contrary, have less influence on the board evaluation, and the central four of them are already occupied at the beginning of the game.

[1]In the online league, players play only 100 games, which is the difference between our estimation of epTDLmpx_12 × 6 performance (87.1%) and 89.5 obtained in the league [89 points (for wins) +0.5 points (for a draw)].



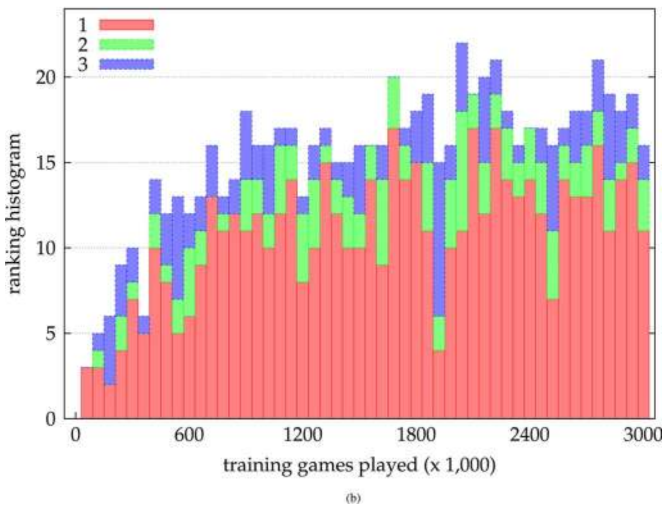
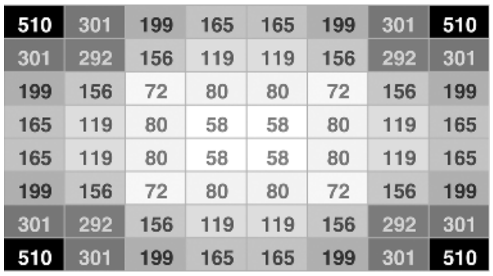


Fig. 9. Frequency distribution of ranks obtained by ETDL and CTDL best-of-generation individuals in a round-robin competition with *Othello* League players. Bar height indicates how many times a particular rank was obtained by the 24 evolved players. (a) Ranks obtained by ETDL. (b) Ranks obtained by CTDL.

TABLE I
OTHELLO LEAGUE RANKING

Name	Size	Played	Won	Drawn	Lost
epTDLmpx_12x6	12 × 6	100	89	1	10
prb_nt30_001	30 × 6	100	84	0	16
prb_nt15_001	15 × 6	100	83	3	14
epTDLxover	12 × 6	100	81	4	15
t15x6x8	15 × 6	100	79	3	18
SelfPlay15	12 × 6	100	77	0	23
tz278_2	278 × 2	100	76	3	21
Nash70	12 × 6	100	72	4	24
x30x6x8	30 × 6	100	71	4	25
pruned-pairs-56t	56 × 2	100	71	1	28



510	301	199	165	165	199	301	510
301	292	156	119	119	156	292	301
199	156	72	80	80	72	156	199
165	119	80	58	58	80	119	165
165	119	80	58	58	80	119	165
199	156	72	80	80	72	156	199
301	292	156	119	119	156	292	301
510	301	199	165	165	199	301	510

Fig. 10. Frequency of board field occurrences in n-tuples.

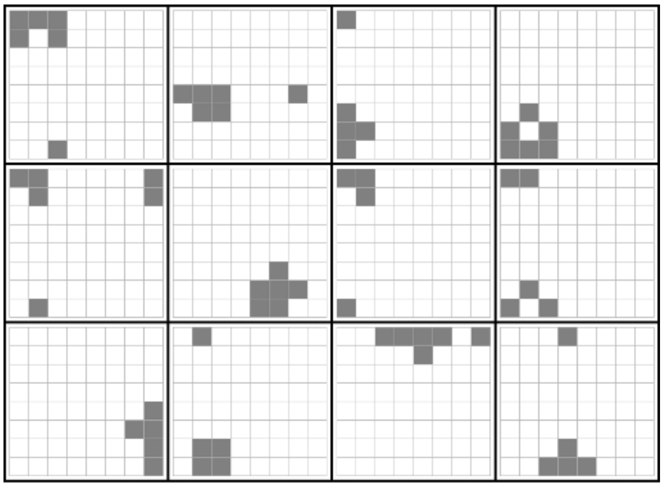


Fig. 11. The tuples of the best CTDL player superimposed on the *Othello* board.

Fig. 11 presents the topology of the best CTDL 12×6-tuple player. The arrangement may seem sparse, but due to the later eightfold mirroring (not shown here), this strategy, in fact, covers almost all board fields. Most tuples watch the combinations of fields that are known to be strategically important in *Othello*: neighboring fields close to corners, or the corners on two opposite sides of the board.

VI. Discussion

Although our best ETDL-evolved player currently leads the *Othello* League, it fares much worse when facing head-to-head other players from the League and the players evolved by means of coevolutionary algorithms. This phenomenon may be explained in terms of solution concepts [9]. ETDL uses the evaluation function based on the WPC-heuristic player, and so optimizes the players' behavior against this specific opponent. The strategies it trains do not have a chance to play with different opponents and learn from such experience. Clearly, randomization of the WPC heuristic, intended to increase behavioral diversity, does not help in this regard. Formally, ETDL implements the specific solution concept of maximization of expected score against the WPC heuristic,[2] which, at least for the game of *Othello*, does not seem to be a good approximation of the general maximal expected utility solution concept. This observation applies also to the way the *Othello* League ranks strategies, and limits the conclusions that may be drawn from that ranking.

[2]In *Othello* with randomized moves.

In contrast, CTDL, a self-learning method equipped with dynamic evaluation function and based on coevolution and TDL, yields players that generalize much better and successfully compete with a variety of opponents: evolved, coevolved, trained by TDL, and the top strategies submitted to the *Othello* League. In particular the last ones, by implementing various approaches and submitted by different researchers, can be claimed to represent a richer repertoire of behaviors. Having said that, we do not argue that CTDL implements any named solution concept. However, the results of extensive round-robin tournaments indicate that it is closer to the solution concept of maximization of expected utility for 1-ply *Othello* than any other method used in this paper, in particular, the top-ranked strategies from the *Othello* League.

Lucas and Runarsson [24] have found that coevolution applied to strategies represented as WPCs learns much slower than TDL, but eventually converges to solutions of similar quality. The results reported in Section V-A1 shed new light on this issue. The performance gap between coevolutionary algorithms and TDL strongly depends on the dimensionality of the search space. For 7×4-tuple networks (567 weights), the coevolutionary algorithm (CEL) in the long run indeed achieves results comparable to TDL, but TDL proves far better for larger search spaces of 9×5 and 12×6 networks (2187 and 8748 weights, respectively). Its gradient-based learning rule is relatively insensitive to the number of variables of consideration, while coevolution does not seem to be able to catch up, even in the long run.

The evolutionary algorithm (EL), despite obtaining higher absolute scores against the WPC heuristic, also tends to attain worse performance for larger networks. The common factor that appears to be responsible for these difficulties is the weight mutation operator, which seems to work reasonably well only in smaller search spaces (cf. Fig. 4). On the other hand, some form of mutation is necessary for the evolutionary approach (Fig. 4 shows that without mutation the score is even worse). Indeed, even random mutation proved effective in high-dimensional spaces in some previous studies [11], [13].

However, given the virtues of gradient-based search methods, it seems natural to couple them with coevolution, as we did here in the CTDL algorithm. This hybridization turns out truly advantageous when coevolution operates exclusively in the network topology search space, leaving the search in the space of weight values entirely to TDL. This approach is an interesting mixture that can be considered as a realization of Lamarckian coevolution, since players pass on to the offspring the traits acquired in their lifetime. Finally, combining two completely different search operators for neuroevolution seems to be especially appealing.

VII. Conclusion

This study demonstrates that, at least for the game of *Othello* and strategies represented as n-tuple networks, the CEL algorithm can autonomously select and maintain a dynamic sample of opponents/trainers that make the resulting strategies generalize better than the strategies trained by an evolutionary approach. The samples of opponents used by the latter method, obtained by randomization of a fixed strategy (WPC), are clearly inferior in that respect. At this stage of research, we can only hypothesize that the major reason for this is greater behavioral diversity of coevolutionary opponents. What nevertheless follows from the experimental results is that the coevolutionary opponents are diversified "in the right way," i.e., they guide the learning process toward more versatile strategies.

However, to find the candidates for a sample of opponents in the first place, an effective search operator is indispensable, particularly when the dimensionality of representation is high. The gradient-based search operator (TDL) proved most useful in this respect, in contrast to purely random mutation. The resulting hybrid, CTDL, may then be seen as a successful combination of an effective learning mechanism (TDL) with an appropriate method for filtering out the right opponents (coevolution). Interestingly, this hybrid seems to scale well with the dimensionality of the search space, i.e., the strategies it yields generalize better for larger representations (n-tuple networks). Future research could investigate in more detail the interplay between the combinatorial, evolutionary search in the space of n-tuple topologies, and continuous, gradient-based search in the space of weights performed by TDL or other variants of reinforcement learning (e.g., by trying to find out the most efficient proportions of usage of these search operators).

Another lesson learned from this work is that assessing players using various performance measures can lead to qualitatively different outcomes, even if all of them take care to ensure that the opponents are diverse. Thus, great caution should be taken when drawing conclusions from such results.

In a broader perspective, the results presented here show that solution concepts, which define an ultimate goal to be achieved in a learning process, are not purely theoretical formalisms of interactive domains (of which games are a special case), but also essential tools of practical relevance. They help to understand the behavior of algorithms, in particular, why they fail or succeed. They can serve as guidelines for designing better learning methods, particularly for determining the choice of opponents and the desired structure of interactions between learners. Finally, they suggest how algorithms should be externally and objectively evaluated, so that an assessment reflects the true usefulness of a strategy.

VIII. Additional Material

The accompanying material for this study (software implementation, parameter files, and the best evolved players) is available online at http://www.cs.put.poznan.pl/mszubert/projects/cecj.html.

Acknowledgment

The authors would like to thank S. M. Lucas and his team for maintaining the *Othello* League and making the submitted players available.

References

[1] P. J. Angeline and J. B. Pollack, "Competitive environments evolve better solutions for complex tasks," in *Proc. 5th Int. Conf. Genetic Algorithms*, S. Forrest, Ed., 1993, pp. 264–270.

[2] T. Bäck, U. Hammel, and H. P. Schwefel, "Evolutionary computation: Comments on the history and current state," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 3–17, Apr. 1997.

[3] W. W. Bledsoe and I. Browning, "Pattern recognition and reading by machine," in *Proc. Eastern Joint IRE-AIEE-ACM Comput. Conf.*, 1959, pp. 225–232.

[4] M. Buro, "Logistello: A strong learning Othello program," presented at the 19th Annu. Conf. Gesellschaft für Klassifikation e.V., Basel, Switzerland, Mar. 8–10, 1995.

[5] S. Y. Chong, M. K. Tan, and J. D. White, "Observing the evolution of neural networks learning to play the game of Othello," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 240–251, Jun. 2005.

[6] S. Y. Chong, P. Tino, D. C. Ku, and Y. Xin, "Improving generalization performance in co-evolutionary learning," *IEEE Trans. Evol. Comput.*, vol. 16, no. 1, pp. 70–85, Feb. 2012.

[7] S. Y. Chong, P. Tino, and X. Yao, "Measuring generalization performance in coevolutionary learning," *IEEE Trans. Evol. Comput.*, vol. 12, no. 4, pp. 479–505, Aug. 2008.

[8] E. D. de Jong, "The maxsolve algorithm for coevolution," in *Proc. Conf. Genetic Evol. Comput.*, 2005, pp. 483–489.

[9] S. G. Ficici, "Solution concepts in coevolutionary algorithms," Ph.D. dissertation, Dept. Comput. Sci., Brandeis Univ., Waltham, MA, USA, 2004.

[10] S. G. Ficici and J. B. Pollack, "A game-theoretic memory mechanism for coevolution," in *Proc. Int. Conf. Genetic Evol. Comput. I*, 2003, pp. 286–297.

[11] D. B. Fogel, *Blondie24: Playing at the Edge of AI.* San Francisco, CA, USA: Morgan Kaufmann, 2002.

[12] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA, USA: Addison-Wesley, 1989.

[13] K.-J. Kim, H. Choi, and S.-B. Cho, "Hybrid of evolution and reinforcement learning for Othello players," in *Proc. IEEE Symp. Comput. Intell. Games*, 2007, pp. 203–209.

[14] A. Kolcz and N. M. Allinson, "N-tuple regression network," *Neural Netw.*, vol. 9, pp. 855–869, Jul. 1996.

[15] K. Krawiec, W. Jaśkowski, and M. Szubert, "Evolving small-board go players using coevolutionary temporal difference learning with archive," *Int. J. Appl. Math. Comput. Sci.*, vol. 21, no. 4, pp. 717–731, 2011.

[16] K. Krawiec and M. Szubert, "Coevolutionary temporal difference learning for small-board go," in *Proc. IEEE Congr. Evol. Comput.*, 2010, pp. 1513–1520.

[17] K. Krawiec and M. Szubert, "Learning N-tuple networks for Othello by coevolutionary gradient search," in *Proc. Genetic Evol. Comput. Conf.*, 2011, pp. 355–362.

[18] K.-F. Lee and S. Mahajan, "The development of a world class Othello program," *Artif. Intell.*, vol. 43, no. 1, pp. 21–36, 1990.

[19] J. Li, P. Hingston, and G. Kendall, "Engineering design of strategies for winning iterated prisoner's dilemma competitions," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 4, pp. 348–360, Dec. 2011.

[20] S. Lucas, "Learning to play Othello with n-tuple systems," *Australian J. Intell. Inf. Process. Syst.*, vol. 9, Special Issue on Game Technology, no. 4, pp. 1–20, 2007.

[21] S. Lucas and A. Amiri, "Recognition of chain-coded handwritten character images with scanning n-tuple method," *Electron. Lett.*, vol. 31, no. 24, pp. 2088–2089, 2002.

[22] S. Lucas and T. P. Runarsson, "Othello competition," [Online]. Available: http://algoval.essex.ac.uk:8080/othello/League.jsp

[23] S. M. Lucas, "Face recognition with the continuous n-tuple classifier," in *Proc. British Mach. Vis. Conf.*, 1997, pp. 222–231.

[24] S. M. Lucas and T. P. Runarsson, "Temporal difference learning versus co-evolution for acquiring Othello position evaluation," in *Proc. IEEE Symp. Comput. Intell. Games*, 2006, pp. 52–59.

[25] S. Luke, "ECJ 20—A Java-based evolutionary computation research system," 2010 [Online]. Available: http://cs.gmu.edu/eclab/projects/ecj/

[26] E. P. Manning, "Using resource-limited Nash memory to improve an Othello evaluation function," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 1, pp. 40–53, Mar. 2010.

[27] T. Miconi, "Why coevolution doesn't "work": Superiority and progress in coevolution," in *Proc. 12th Eur. Conf. EuroGP*, Tübingen, Germany, Apr. 2009, pp. 49–60, DOI: 10.1007/978-3-642-01181-8_5.

[28] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," Caltech Concurrent Computation Program C3P Rep., 1989, vol. 826.

[29] J. B. Pollack and A. D. Blair, "Co-evolution in the successful learning of backgammon strategy," *Mach. Learn.*, vol. 32, no. 3, pp. 225–240, 1998.

[30] R. Rohwer and M. Morciniec, "A theoretical and experimental account of n-tuple classifier performance," *Neural Comput.*, vol. 8, pp. 629–642, 1996.

[31] T. P. Runarsson and S. Lucas, "Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board Go," *IEEE Trans. Evol. Comput.*, vol. 9, no. 6, pp. 628–640, Dec. 2005.

[32] K. O. Stanley, "Efficient evolution of neural networks through complexification," Ph.D. dissertation, Dept. Comput. Sci., Univ. Texas at Austin, Austin, TX, USA, 2004.

[33] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, pp. 9–44, 1988.

[34] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[35] M. Szubert, "cECJ—Coevolutionary computation in Java," 2010 [Online]. Available: http://www.cs.put.poznan.pl/mszubert/ projects/cecj.html

[36] M. Szubert, W. Jaśkowski, and K. Krawiec, "Coevolutionary temporal difference learning for Othello," in *Proc. IEEE Symp. Comput. Intell. Games*, Milano, Italy, 2009, pp. 104–111.

[37] M. Szubert, W. Jaśkowski, and K. Krawiec, "Learning board evaluation function for Othello by hybridizing coevolution with temporal difference learning," *Control Cybern.*, vol. 40, pp. 805–831, 2011.

[38] G. Tesauro, "Practical issues in temporal difference learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 257–277, 1992.

[39] G. Tesauro, "Temporal difference learning and TD-Gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[40] J. Togelius, F. Gomez, and J. Schmidhuber, "Learning what to ignore: Memetic climbing in topology and weight space," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 3274–3281.

[41] S. van den Dries and M. A. Wiering, "Neural-fitted TD-leaf learning for playing Othello with structured neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 11, pp. 1701–1713, Nov. 2012.

[42] W. Vogt and R. Johnson, *Dictionary of Statistics & Methodology: A Nontechnical Guide for the Social Sciences*. New York, NY, USA: Sage, 2011.

[43] R. A. Watson and J. B. Pollack, "Coevolutionary dynamics in a minimal substrate," in *Proc. Genetic Evol. Comput. Conf.*, 2001, pp. 702–709.

[44] T. Yoshioka, S. Ishii, and M. Ito, "Strategy acquisition for the game "Othello" based on reinforcement learning," *IEICE Trans. Inf. Syst.*, vol. 82, no. 12, pp. 1618–1626, 1999.

Marcin Szubert received the M.Sc. degree in computer science from Poznań University of Technology, Poznań, Poland, in 2009, where he is currently working toward the Ph.D. degree at the Laboratory of Intelligent Decision Support Systems, Institute of Computing Science.

His research interests primarily cover the area of computational intelligence and machine learning with applications to game playing. Particularly, he works on improving the performance of reinforcement learning methods through the use of coevolution.

Wojciech Jaśkowski received the B.Eng., M.Sc., and Ph.D. degrees in computing science from Poznań University of Technology, Poznań, Poland, in 2004, 2006, and 2011, respectively.

Currently, he is an Assistant Professor at the Laboratory of Intelligent Decision Support Systems, Institute of Computing Science, Poznań University of Technology. He is an author of nearly 30 publications in computational intelligence, evolutionary computations in particular. His main research addresses coevolution, cooptimization, genetic programming, and learning strategies for interactive domains and games.

Krzysztof Krawiec (M'06) received the Ph.D. and Habilitation degrees from Poznań University of Technology, Poznań, Poland, in 2000 and 2005, respectively.

He is an Associate Professor at Poznań University of Technology, and works mainly on topics related to evolutionary computation, genetic programming, and pattern recognition. His recent work includes: evolutionary computation for machine learning, primarily for learning game strategies and for synthesis of pattern recognition systems; semantics in genetic programming, particularly in operator design and problem decomposition; coevolutionary algorithms, mainly the role of interactions and coordinate systems for test-based problems; and modeling of complex phenomena using genetic programming (e.g., climate modeling). He is the author of *Evolutionary Synthesis of Pattern Recognition Systems* (New York, NY, USA: Springer-Verlag, 2005).

Dr. Krawiec is the President of the Polish Chapter of the IEEE Computational Intelligence Society for the term 2013–2014.