Learning N-tuple Networks for Othello by Coevolutionary Gradient Search

Krzysztof Krawiec and Marcin Szubert
Institute of Computing Science
Poznan University of Technology
Piotrowo 2, 60965 Poznań, Poland
kkrawiec,mszubert@cs.put.poznan.pl

ABSTRACT

We propose Coevolutionary Gradient Search, a blueprint for a family of iterative learning algorithms that combine elements of local search and population-based search. The approach is applied to learning Othello strategies represented as n-tuple networks using different search operators and modes of learning. We focus on the interplay between the continuous, directed, gradient-based search in the space of weights, and fitness-driven, combinatorial, coevolutionary search in the space of entire n-tuple networks. In an extensive experiment, we assess both the objective and relative performance of algorithms, concluding that the hybridization of search techniques improves the convergence. The best algorithms not only learn faster than constituent methods alone but also produce top ranked strategies in the online Othello League.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Connectionism and neural nets, Parameter learning*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*; J.m [**Miscellaneous**]:

General Terms

Algorithms

Keywords

Othello, Coevolution, Temporal Difference Learning, N-tuple Networks

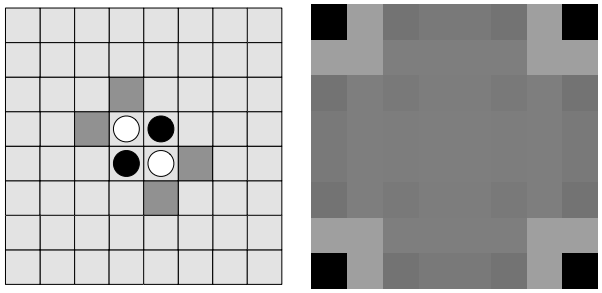
1. INTRODUCTION

For most of non-trivial learning and optimization problems finding a solution in one step is formally impossible or computationally infeasible. This often remains true not only when the goal of the search algorithm is to find a truly optimal solution but also when a well-performing approximate solution is sought for. As a result, most of the contemporary search algorithms are, in a wide sense, iterative (incremental), meaning that a working candidate solution (or a set of solutions) is maintained and improved towards some goal(s).

At least two questions have to be answered in order to design a specific search algorithm along this template. Firstly, how to define the update rule according to which the working solution(s) would be modified? And secondly, where should the information required for that update come from? In an attempt to answer the first question, we can draw a broad distinction between the *directed* and *undirected* search. Well-known representatives of the former category are gradient-based methods. Their common feature is the assumption that it is possible to estimate, analytically or numerically, the direction and the rate of change of the objective function in the close vicinity of the considered candidate solution, with respect to the variables that describe it. Thanks to that, they typically perform well in smooth, continuous search spaces, even in presence of many variables. In the undirected search, no special assumptions concerning the objective function are made and the update rule typically operates with a certain degree of randomness without an explicit analysis of the objective function (apart from sampling its values). These methods are applicable in a wider range of contexts, including combinatorial problems, but often converge slow to well-performing solutions. Many variants of evolutionary computation belong to this category.

The answer to the second question is apparently trivial: in a typical setting, it is the objective function that is the only and ultimate source of information that guides the search. Here, however, we consider the surprisingly common class of problems for which an objective function does not exist or it is computationally expensive – a class that embraces, among others, learning of game strategies. Within that class, the primary driver of search are *interactions* that serve as a surrogate for the objective function. In the simplest degenerate scenario, an interaction takes place between a solution and itself. Self-play, a technique used for game strategy learning, represents this case. In population-based methods, like coevolutionary algorithms, interaction involves two or more different solutions that can be considered as *peers*, as they typically come from the same population (or from the equivalent generation of another population). Such approach enables the algorithm to build a kind of 'internal gradient' between individuals that mutually act as teachers and learners. Finally, sometimes it may be useful to enforce interactions with some *reference solutions*, which have some special status in the algorithm. They typically embody the knowledge

GECCO'11, July 12–16, 2011, Dublin, Ireland.



(a) Othello initial board state (b) Heuristic WPC weights

Figure 1: Othello board and its colouring according to heuristic player weights (darker color – greater weight).

acquired by the algorithm in the previous iterations, as it is the case in archives used in evolutionary computation or belief space in cultural algorithms.

Different search algorithms and different interaction modes feature then mutually complementary features that deserve some form of fusion. The past literature on the topic (see Section 2.3) misses a systematic investigation into the interplay between the above components. In response to that, our contribution in this study is a family of algorithms that combine these design choices in different ways, termed jointly *Coevolutionary Gradient Search*, and their experimental application to the task of learning n-tuple networks strategies for the game of Othello, without explicit involvement of human expert knowledge.

2. THE GAME OF OTHELLO

Othello is played by two players on an 8×8 board. Typically, pieces are disks with a white and black face, each face representing one player. Figure 1a shows the initial state of the board; each player starts with two stones 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. Firstly, the position of the piece must be adjacent to an opponent's piece. Secondly, the new piece and some other piece of the current player must form a vertical, horizontal, or diagonal line 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. This makes the game particularly dramatic: a single move may gain the player a large number of pieces and swap players' chances of winning.

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 at the end has more disks wins; the game can also end with a draw.

2.1 The Othello League

A good overview of different Othello player architectures and their estimated performance is provided by the Othello Position Evaluation Function League [12]. 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* player (WPC). WPC is the simplest 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 is illustrated in the Fig. 1b. We use it also as an opponent in our experiments 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 that involve numerous parameters are: a *symmetric n-tuple network*, a *multi-layer perceptron* (MLP), and a *spatial MLP*. The best player found so far in the league was an n-tuple network with a winning percentage of just below 80%. Such result can be achieved by a simple self-play training with *temporal difference learning* (TDL, see Section 3.1) [11]. In our research we use the same architecture to compare performance acquired by different learning methods.

2.2 The N-tuple Network Architecture

The idea of n-tuple systems was originated by Bledsoe and Browning [2] for use in character recognition. Since then it has been successfully applied to both classification [17] and function approximation tasks [9]. Their main advantages include conceptual simplicity, speed of operation, and capability of realizing non-linear mappings to spaces of higher dimensionality. Recently, Lucas proposed employing the n-tuple architecture also for game-playing purposes [11].

By definition, an n-tuple system expects as input some compound entity (matrix, tensor, image) $\mathbf{x}$, which elements (usually scalar variables) can be indexed and retrieved in a systematic way, most frequently by using some form of coordinates. An n-tuple network operates by sampling that input object with m n-tuples. Each n-tuple t_i is a sequence of n variables $a_{ij}, j = 0..n-1$ corresponding to locations in the input. Assuming that each variable has one of v possible values, a single n-tuple can be viewed as a template for an n-digit number in base-v numeral system. For this reason, we can state that n-tuple represent one of v^n possible values. The number represented by the n-tuple t_i is used as an index in associated look-up 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:

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

where $\mathbf{x}(a_{ij})$ denotes selecting from $\mathbf{x}$ the element located at position indicated by a_{ij}. In general, two n-tuples are allowed to overlap, i.e., to select the same variable from $\mathbf{x}$.

In the context of Othello, an n-tuple network acts as a state evaluation function. It takes a board state as an input and returns its utility. Input variables are sampled board locations and their value is 0, 2, or 1 if, respectively, location is occupied by a white piece, black piece, or remains empty. Consequently, n-tuple represents a ternary number which is used as an index for the associated look-up table containing 3^n entries (see Fig. 2). Additionally, symmetric sampling (introduced in [11]) can be incorporated – a single

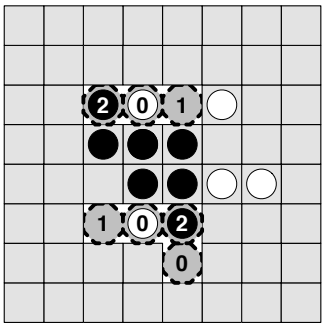


Figure 2: Two sample n-tuples viewed as templates for base-3 numbers. Each input location represents a ternary digit. Multiplying them by successive powers of 3 leads to decimal values of $2 \cdot 3^2 + 1 \cdot 3^0 = 19$ and $1 \cdot 3^3 + 2 \cdot 3^1 = 33$, which are used as indexes in the associated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

n-tuple is employed eight times, once for each of possible board reflection and rotation. LUT values indexed by all such equivalents are summed together to form the output of the particular n-tuple. Final value of a board is simply the sum of all n-tuple outputs.

The number of possible n-tuple instances is exponential in function of the size of $\mathbf{x}$ and n, so assigning the 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 neighbourhood of chosen locations is more essential. For this reason, and particularly for Othello, it sounds natural to choose connected set of locations, like a straight line or a rectangle area, from the board and assign it as successive n-tuple inputs. In our implementation we allowed for more flexible assignments in the form of *snake* shapes, proposed by Lucas in [11]. 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 4.1.

2.3 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.

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. Today, one of the strongest Othello programs is Logistello [3], which makes use of advanced search techniques and applies several methods to learn from previous games.

Recently, the mainstream research on Othello has moved towards better understanding of what types of learning algorithms and player architectures work best. The CEC Othello Competitions [12] 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 and opponent strategies given a priori. This task formulation is addressed by, among others, Temporal Difference Learning (TDL) and Coevolutionary Learning (CEL), which were applied to Othello by Lucas and Runarsson [13]. Other examples of using self-learning approaches for Othello include coevolution of spatially aware MLPs [4], coevolutionary temporal difference learning [23], and Nash Memory applied for coevolved n-tuple networks [15].

3. METHODS

3.1 Temporal Difference Learning

Since the influential work of Tesauro [24] and the success of his *TD-Gammon* player trained through self-play, *Temporal Difference Learning* (TDL) [21] 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 intra-game 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 to match more closely 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 (Eq. 1) whose outputs are squeezed to the interval $(-1, 1)$ by hyperbolic tangent. Using such prediction function within $TD(0)$ update rule (Eq. 2) results in changing LUT weights according to:

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

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 white, and 0 when the game ends in a draw.

The process of learning consists in applying the above formula to the look-up table entries after each move. The training data for that process, i.e., a collection of games, each of them being a sequence of states $\mathbf{x}_1, \mathbf{x}_2, \ldots$, is 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 can be encountered and explored by a learner, what makes learning ineffective. To remedy this situation, at each turn, a random move is forced with certain probability. After such a random move, weight update does not occur. Thanks to random moves, players are confronted with a wide spectrum of possible behaviours of their opponents, including the quite unexpected ones, which makes them more robust and versatile.

3.2 Coevolutionary Learning

Temporal difference learning approach is a gradient-based local search method that maintains a single solution and as such has no built-in mechanisms for escaping from local minima. Evolutionary computation, 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 minima by its implicit parallelism and nondeterministic update of candidate solutions. Therefore, the local characteristics of the fitness landscape is no longer the only force that moves individuals in the search space – the second one is randomness. Consequently, evolutionary computation seems to be an attractive complementary alternative for TD for learning game strategies.

However, one faces substantial difficulty when designing fitness function, an indispensable component of an evolutionary algorithm that drives the search process, for the task of learning game strategies. To properly guide the search, fitness function should *objectively* assess the utility of the evaluated individual, which in case of games can be done only by playing against *all* possible opponent strategies. For the majority of games this is computationally intractable. Considering instead only a limited sample of opponents lessens the computational burden but biases the search. For this reason, a much more appealing alternative, from the viewpoint of game strategy learning, is *coevolution* where individual's fitness depends on the results of interactions with other individuals from the population. 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. This evaluation scheme is termed as competitive coevolution [1].

Coevolutionary Learning (CEL) of game strategies follows this idea and typically starts with generating a random initial population of player individuals. Individuals play games with each other and the outcomes of these interactions determine their fitness values. 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 CEL to embrace a broad class of algorithms that have been successfully applied to many two-person games, including Backgammon [16], Checkers [7],and a small version of Go [19]. 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 [13].

3.3 Coevolutionary Learning with Archives

As the set of opponent strategies faced by an individual is limited by the population size, the evaluation scheme used in pure CEL is still only a substitute for the objective fitness function. The advantage of this approach, when compared to evolution with fitness function based on a fixed sample of strategies, is that the set of opponents changes with time (from one generation to another) so that the individuals belonging to a particular lineage are exposed to more opponents.. In this way, the risk of biasing the search towards an arbitrary direction is expected to be reduced. However, without some extra mechanisms, there is no guarantee that the population will change in the desired direction(s) or change at all. The latter scenario, lack of progress, can occur when, for instance, player's opponents are not challenging enough or much too difficult to beat. These and other undesirable phenomena, jointly termed *coevolutionary pathologies*, have been identified and analyzed [25, 6].

In order to deal with coevolutionary pathologies, *coevolutionary archives* were introduced that try to sustain progress. A typical archive is a (usually limited in size, yet diversified) sample of well-performing strategies found so far. In this study we use the Hall of Fame (HoF, [18]), which simply stores all the best-of-generation individuals encountered so far. The individuals in population, apart from playing against their peers, are also forced to play against randomly selected players from the archive. The fitness is thus partially determined by confrontation with past 'champions'.

3.4 Coevolutionary Temporal Difference Learning

The past results of learning WPC strategies for Othello [13] and small-board Go [19] 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 [11]. Still, it sounds reasonable to combine these approaches into a hybrid algorithm exploiting different characteristics of the search process performed by each method. In [23] and [10] a method termed *Coevolutionary Temporal Difference Learning* (CTDL) was proposed and applied to learn WPC strategies. CTDL maintains a population of players and alternately performs TD learning and coevolutionary learning. In the TD 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.* The overall conclusion was positive for CDTL, 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 architecture.

Other hybrids of TDL and CEL have been occasionally considered in the past. Kim *et al.* [8] 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 [15] a bounded-size Nash Memory archive for coevolution is combined with the TDL used as a weight mutation operator.

4. EXPERIMENTAL SETUP

To verify the hybridization of coevolution with temporal difference learning, several experiments were conducted. All algorithms were implemented using our coevolutionary algorithms library called cECJ [22] built upon Evolutionary Computation in Java (ECJ) framework [14]. Our unit of computational effort is a single game and the time of other operations is neglected. To provide fair comparison, all runs were stopped when the number of games played reached $2,000,000$. Each experiment was repeated 24 times.

4.1 Player Architecture

We rely on n-tuple network because of its appealing potential demonstrated in recent studies [15, 11] and promising results in the Othello League [12]. We start from small networks formed by 7 instances of 4-tuples (7×4) which include 567 weights. Later, we move to 9×5 networks (2197 weights on aggregate) to end up with the largest 12×6 architecture (8748 weights) that has been recently successfully applied to Othello by Manning [15]. 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 2.2). Regarding the look-up table weights, their initial values depend on particular learning algorithm. As previous research shows, TDL learns faster when 0-initialized. Evolutionary methods, on the other hand, assume that the population is randomly dispersed in the search space. For this reason, in pure coevolutionary algorithms we start from weights initialized randomly from the $[-1, 1]$ range.

4.2 Search Operators

The considered search heuristics operate in two spaces – discrete network topology space and 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 tuples 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 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 too [20].

We employ two types of operators: genetic and gradient-based. Let us note that the former ones rely on direct encoding of strategies, i.e., individual's genome is a concatenation of lookup table weights associated with its n-tuples. Overall, we use three operators:

- *weight mutation* (m_w) – each weight (LUT entry) with probability $p_{mw} = 0.05$ undergoes Gaussian mutation ($\sigma = 0.25$)
- *topology mutation* (m_t) – each input (board location) is replaced, with probability $p_{mt} = 0.01$, by another input from its neighbourhood.
- *topology crossover* (x) – with probability $p_x = 1.00$, individual reproduction is sexual – two individuals mate and exchange genes, i.e., entire tuples with look-up tables. An offspring inherits $m/2$ randomly selected tuples from each. Without crossover reproduction is agamic – a new individual is a (mutated) copy of its single parent.

Our gradient-based search operators work in the weight space and consists in a single training game incorporating $TD(0)$ algorithm (see Section 3.1) We use learning rate $\alpha = 0.001$ and force random moves with probability $\varepsilon = 0.1$. The crucial question is how to get an opponent for this game. We examine three possibilities: *self-play game* (s), *population opponent* (p) and *archival opponent* (a). Interestingly, learning by interactions with other players brings to mind the cultural aspect of learning and cultural transmission of behaviours which were investigated in the context of neural networks by Denaro *et al.* [5].

4.3 Learning Algorithms

Several methods were prepared, each being a combination of CEL, TDL and HoF with specific search operators.

Temporal Difference Learning (TDL) searches only the weight space using a single search point and a self-play TDL game as the only search operator.

Coevolutionary Learning (CEL) uses a generational coevolutionary algorithm with population of 50 individuals. In the evaluation phase, a round-robin tournament is played between all individuals, with wins, draws, and losses rewarded by 2, 1, and 0 points, respectively, and the total number of points becomes individual's *competitive fitness*. For each pair of individuals, two games are played, with players swapping the roles of the black and the white player. The evaluated individuals are subject to tournament selection with tournament size 5, and then, undergo genetic operators (weight mutation and topology crossover).

The Hall of Fame Archive (HoF) extends the coevolutionary methods (CEL, CTDL). Each individual plays games with all 50 individuals from the population and, additionally,with 50 randomly selected individuals from the archive, so that its fitness is determined by the outcomes of 100 games scored as in CEL. After each generation, the individual with the highest fitness joins the archive.

Parallel TDL (PTDL) is a parallel version of TDL. Instead of a single player, this method maintains a population of 50 players, each learned through self-play. This setup allows us to check if it is beneficial is to distribute the available training games among several players rather than allocating them to a single one.

Coevolutionary TDL (CTDL) combines CEL and TDL as described in Section 3.4. The basic version (denoted as $CTDL$-s for self-play) is simply PTDL with coevolutionary selection. More complex setups are constructed by adding topology and weight search operators. For example, $CTDL$-sxm_w, employs self-play TDL training, topology crossover and weight mutation operators. The most sophisticated hybridization, called $CTDL$-$asxm_t$ involves TDL training (with itself and archival opponent) and topology crossover and mutation operators. By default, in each TDL phase 5000 training games are distributed among players. In case of uniform distribution, each player is learned in 100 games.

Evolutionary TDL (ETDL) operates as CTDL but uses an external objective to evaluate individuals. Instead of the round-robin tournament, each individual plays 50 randomized games again the WPC-heuristics player. All search operators used in CTDL could be applied here as well.

4.4 Performance Measures

Monitoring the progress of learning in interactive domains is hard since, generally, there is no precise and easily computable objective performance measure. A fully objective assessment requires playing against all possible opponents, but the sheer number of them makes this option impossible. To monitor the progress, 50 times per run (approximately every 40,000 games), we appoint the individual with the highest competitive fitness (i.e., the subjectively best strategy) as the best-of-generation individual and assess its performance (for TDL, the single strategy maintained by the method is the best-of-generation by definition). It plays then 1,000 games (500 as black and 500 as white) against a predefined, human-designed WPC strategy. The resulting percentage score becomes our estimate of its *absolute* performance. The second performance measure introduced below gauges the *relative* progress of particular methods via a round-robin tournament of representative individuals.

It should be emphasized that, except for the ETDL method, the interactions taking place in all assessment methods do not influence the learning process.

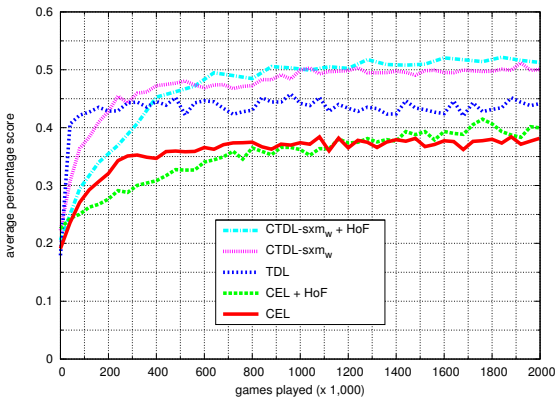
5. RESULTS

5.1 Performance Against the Heuristic Player

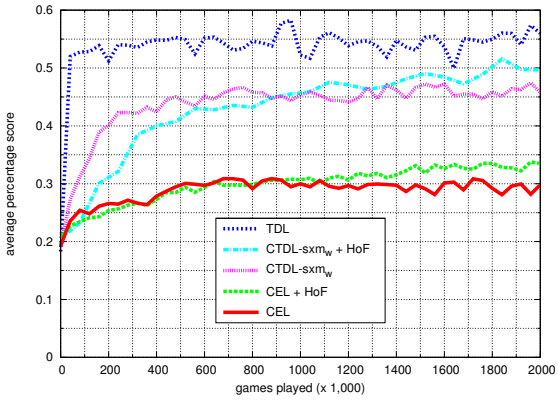
This performance measure, coming from the Othello League [12], is the percentage of points (one point awarded for a win, 0.5 for a draw) scored against the WPC-heuristics, a fixed player encoded as WPC architecture with weights graphically illustrated in Figure 1b. All players in our experiments are deterministic, as well as the game of Othello itself. Thus, in order to estimate the score of a given trained player against the WPC-heuristic, we forced both players to make random moves with probability $\varepsilon = 0.1$. This provides richer repertoire of players' behaviors and makes the resulting estimates more continuous and robust.

In the first experiment, we focused on the scalability of methods with respect to representation size. Figure 3 shows how their performance varies with growing network size. The results obtained for the smallest network confirm our expectations – hybrid methods are in the long run significantly better than non-hybrid ones. However, increasing the network size does not bring any benefits to the coevolution-based approaches. Indeed, only TDL is able to utilize the possibilities offered by a larger network and improve its performance, while other methods perform even worse than with smaller networks. Therefore, we can conclude that random mutation operator fails to elaborate progress in the exponentially growing weight space. To maintain its chance for successful modifications, the population size should also increase by the same degree. Conversely, directed changes realized by TDL result in rapid learning regardless of the weight space dimensionality (followed by stagnation, though).

For the above reasons, when passing to the largest 12×6 networks, we decided also to replace the genetic weight mutation by topology mutation. This leads to searching the weight space exclusively by gradient-based operators. Additionally, ETDL with the same set of operators was applied. As Fig. 4 demonstrates, once again the basic coevolutionary methods do not scale well. The modified CTDL with topology mutation is significantly better but stagnates just at the same level as the pure TDL. The most striking observation is the excellent score achieved by ETDL. The best player



(a) 7×4 n-tuple network



(b) 9×5 n-tuple network

Figure 3: Comparison of learning methods for two network sizes. Average performance of the best-of-generation individuals measured as a score against WPC-heuristic.

found by this method reached 90%, and it easily took the lead when submitted to the Othello League [12].

The average score of between 65% and 70% obtained by CTDL is the same as reported in [11] for simple self-play TDL and in [15] for sophisticated Nash Memory approach. Thus, it suggests that there is some kind of *ceiling effect* in evaluation of self-learning methods with this measure. The WPC-heuristic player does not offer sufficiently diversified challenge to differentiate players trained by these algorithms. To verify this hypothesis we conducted a series of relative performance assessments which are discussed below.

5.2 Tournament Between Teams

A handcrafted heuristic strategy, even when randomized, cannot be expected to represent in full the richness of possible behaviors of Othello strategies. In order to get a more realistic performance estimate, we recruit sets of diverse opponents composed of best-of-generation individuals representing particular methods. Technically, each team embraces all the best-of-generation strategies found by 24 runs of a particular method. 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 final score of a team is the overall sum of points obtained by its players.

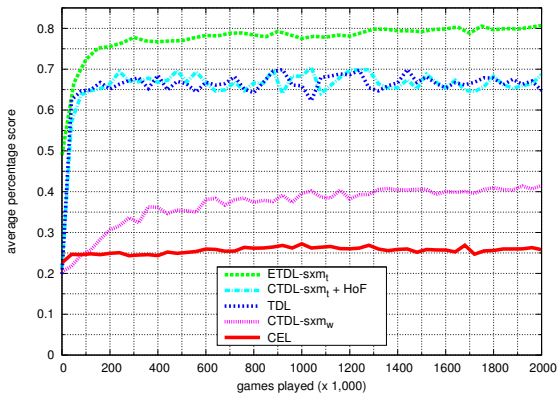


Figure 4: Average performance of 12×6 n-tuple network players measured as a score against WPC-heuristic.

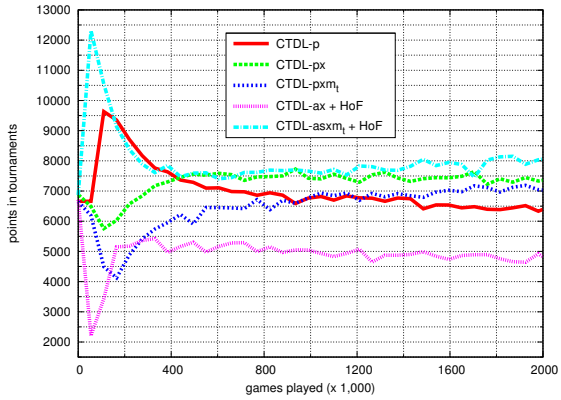


Figure 6: Relative comparison of algorithms which employ **mutual-play training** (population or archival opponent).

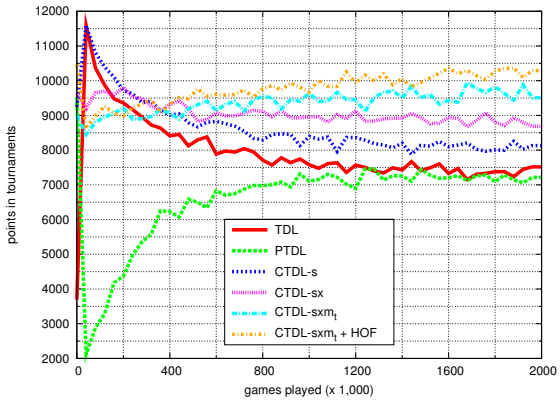


Figure 5: Relative comparison of algorithms which employ **self-play training**.

Let us notice that the round robin tournament offers yet another advantage: there is no need to randomize moves (as it was the case when playing against a single external player), since the presence of multiple strategies in the opponent team provides enough behavioral variability.

Figure 5 plots the relative performance of algorithms that use self-play TDL games as a weight modification operator. The methods that played at the same level against the WPC-heuristic, here obtain diametrically different results. Adding topology search operators to the basic $CTDL$-s leads to successively improved performance. Comparing TDL and PTDL lets us state that it is better to devote all available resources (training games) to the single TDL player than distribute them evenly among many players.

In the next experiment we used the same methodology to compare algorithms employing the mutual-play TDL games with population and/or archival opponents. Since the learning environment is more challenging in this setup, the number of training games played in the TDL phase was increased to 50000. In most methods, a trained player was paired with 10 randomly selected opponents (from the archive or population) and played 100 training games with each of them. The only exception is $CTDL$-$asxm_t$ where it played 400 games against itself and 200 games with each of the three newest archival individuals (his direct ancestors).

Figure 6 shows that, in contrast to the previously examined methods, the topology mutation operator appears to be much more disruptive now. This can be explained by the fact that random topology modifications are harder to compensate by TDL because training opponents are strongly diversified and can represent behaviours that were never encountered before. TDL by mutual-play with randomly selected opponents explores the game state space too broadly, lacking time for its thorough exploitation. The right balance in the exploration-exploitation tradeoff is kept in $CTDL$-$asxm_t$, where more games with fewer opponents were played. Finally, the weak performance of $CTDL$-ax is caused by the detachment of the subset of players used for learning from the subset used for evaluation. Consequently, coevolutionary gradient is lost and genetic drift is all that is left.

The last experiment involved the best methods from the previous assessments, with emphasis on relative performance of CTDL methods vs. ETDL. The latter one gives an impression of being the best with regard to external performance measure provided by the WPC-heuristic. Figure 7 demonstrates that this evolutionary method trained to win against a specific opponent is relatively the worst. We can conclude that it lacks game experience with players representing diametrically different strategies than the WPC-heuristic. Coevolutionary methods, on the other hand, are trained to play well against a wide variety of opponents and thus possess more universal Othello-playing skills.

6. CONCLUSIONS

This study is an attempt to bridge the gap between the combinatorial and the gradient-based search. The proposed approach of Coevolutionary Temporal Difference Learning is an interesting mixture that can be analyzed from many perspectives. From the evolutionary point of view it can be considered as a realization of Lamarckian coevolution, since organisms (Othello players) learn throughout their life to pass acquired traits on to the offspring. Regarding interactions between learning and evolution, it is important to mention the noticeable analogy to cultural coevolution and non-genetic transmission of behaviours accomplished with learning from the others. Finally, the two constituents of the proposed hybridization have completely different nature of the search process. Combining their operators for neuroevolution seems to be especially appealing.

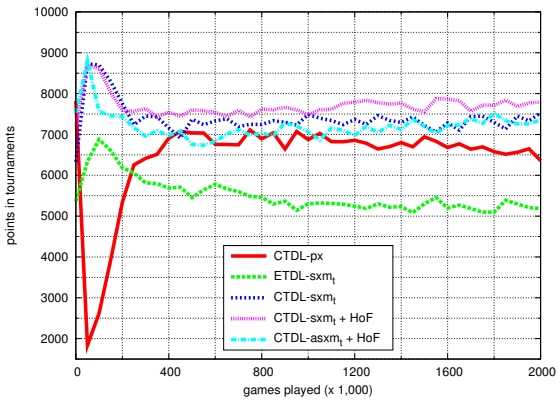


Figure 7: Relative comparison of the best coevolutionary methods with the evolutionary approach.

Regardless of the viewpoint, it has been shown that a properly balanced hybridization of directed and undirected search can result in sustaining the progress of learning and obtaining the best performance in the long-term perspective. Moreover, the efficiency of n-tuple network has been confirmed and verified in the Othello League. Although the player produced by evolutionary approach gains the advantage in the league, our experiments demonstrate that it overfits due to being taught only by a heuristic opponent. When confronted with other entries, it would be probably easily beaten by the coevolution-based solution.

Even though we find the results encouraging, some elements of the approach need more elaboration. For instance, the topology crossover that exchanges entire n-tuples is rather simplistic, as it implicitly assumes that the associated LUTs have weights within similar ranges. Generally this is not true, so we expect that the crossover is quite often disruptive. Designing genetic operators dedicated for n-tuple networks is an interesting direction for future work.

7. REFERENCES

[1] P. J. Angeline and J. B. Pollack. Competitive environments evolve better solutions for complex tasks. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 264–270, 1993.

[2] W. W. Bledsoe and I. Browning. Pattern recognition and reading by machine. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, IRE-AIEE-ACM '59 (Eastern), pages 225–232, New York, NY, USA, 1959. ACM.

[3] M. Buro. Logistello: A strong learning othello program. In *19th Annual Conference Gesellschaft für Klassifikation e.V.*, 1995.

[4] 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. Evolutionary Computation*, 9(3):240–251, 2005.

[5] D. Denaro and D. Parisi. Cultural evolution in a population of neural networks. In *Proceedings of the 8th italian workshop on neural nets*, 1997.

[6] S. G. Ficici. *Solution Concepts in Coevolutionary Algorithms*. PhD thesis, Waltham, MA, USA, 2004. Adviser-Jordan B. Pollack.

[7] D. B. Fogel. *Blondie24: playing at the edge of AI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[8] K.-J. Kim, H. Choi, and S.-B. Cho. Hybrid of evolution and reinforcement learning for othello players. *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 203–209, 2007.

[9] A. Kolcz and N. M. Allinson. N-tuple regression network. *Neural Netw.*, 9:855–869, July 1996.

[10] K. Krawiec and M. Szubert. Coevolutionary temporal difference learning for small-board go. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 1513–1520, 2010.

[11] S. 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.

[12] S. Lucas and T. P. Runarsson. Othello Competition; http://algoval.essex.ac.uk:8080/othello/League.jsp.

[13] S. M. Lucas and T. P. Runarsson. Temporal difference learning versus co-evolution for acquiring othello position evaluation. In *CIG*, pages 52–59, 2006.

[14] S. Luke. ECJ 20 — A Java-based Evolutionary Computation Research System. `http://cs.gmu.edu/~eclab/projects/ecj/`, 2010.

[15] E. 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.

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

[17] R. Rohwer and M. Morciniec. A theoretical and experimental account of n-tuple classifier performance. *Neural Comput.*, 8:629–642, April 1996.

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

[19] T. P. Runarsson and S. Lucas. Co-evolution versus self-play temporal difference learning for acquiring position evaluation in small-board Go. *IEEE Transactions on Evolutionary Computation*, 9, 2005.

[20] K. O. Stanley. *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 2004.

[21] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[22] M. Szubert. cECJ — Coevolutionary Computation in Java. `http://www.cs.put.poznan.pl/mszubert/projects/cecj.html`, 2010.

[23] M. Szubert, W. Jaśkowski, and K. Krawiec. Coevolutionary temporal difference learning for othello. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[24] G. Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, 1995.

[25] R. A. Watson and J. B. Pollack. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 702–709, 2001.