Int. J. Appl. Math. Comput. Sci., 2016, Vol. 26, No. 1, 215–229
DOI: 10.1515/amcs-2016-0015

amcs

THE PERFORMANCE PROFILE: A MULTI–CRITERIA PERFORMANCE EVALUATION METHOD FOR TEST–BASED PROBLEMS

WOJCIECH JAŚKOWSKI [a,*], PAWEŁ LISKOWSKI [a], MARCIN SZUBERT [a], KRZYSZTOF KRAWIEC [a]

[a] Institute of Computing Science
Poznań University of Technology, ul.Piotrowo 2, 60-965 Poznań, Poland
e-mail: {wjaskowski,pliskowski,mszubert,kkrawiec}@cs.put.poznan.pl

In test-based problems, solutions produced by search algorithms are typically assessed using average outcomes of interactions with multiple tests. This aggregation leads to information loss, which can render different solutions apparently indifferent and hinder comparison of search algorithms. In this paper we introduce the *performance profile*, a generic, domain-independent, multi-criteria performance evaluation method that mitigates this problem by characterizing the performance of a solution by a *vector* of outcomes of interactions with tests of various difficulty. To demonstrate the usefulness of this gauge, we employ it to analyze the behavior of Othello and Iterated Prisoner's Dilemma players produced by five (co)evolutionary algorithms as well as players known from previous publications. Performance profiles reveal interesting differences between the players, which escape the attention of the scalar performance measure of the expected utility. In particular, they allow us to observe that evolution with random sampling produces players coping well against the mediocre opponents, while the coevolutionary and temporal difference learning strategies play better against the high-grade opponents. We postulate that performance profiles improve our understanding of characteristics of search algorithms applied to arbitrary test-based problems, and can prospectively help design better methods for interactive domains.

Keywords: coevolutionary algorithms, evolution strategies, Othello, Reversi, games, multi-objective analysis.

1. Introduction

Test-based problems (Bucci *et al*., 2004; de Jong, 2004) are learning and co-optimization problems in which *candidate solutions* and *tests* interact with each other. Every interaction has an *outcome*, typically expressed as a scalar, and the goal of the search is to find a solution with a desired characteristic of interaction outcomes. Common scenarios of solving test-based problems are learning game strategies, evolving designs for different environments, or algorithms tested on instances. One approach to this class of problems are evolutionary and coevolutionary algorithms (Hillis, 1990).

The challenge in designing effective algorithms for test-based problems lies, among others, in obtaining accurate evaluation of solutions. An objective assessment of a solution's performance is often computationally too expensive to be useful in practice. For example, the *expected utility*, one of the popular performance measures in the game domain, is the expected score of a game playing strategy against a random opponent strategy. Even for the apparently trivial problem of learning strategies for the game of tic-tac-toe, Jaśkowski and Krawiec (2011) showed that calculating the exact expected utility requires playing games against roughly 3.47×10^{162} unique opponent strategies.

Therefore, effective algorithms employ heuristic methods of evaluating solution performance, which rely on a limited number of interactions between solutions and tests. The most commonly used evaluation measures are (i) an average score against a pool of fixed, manually-designed tests (Lucas and Runarsson, 2006; Fogel, 2001), (ii) a round-robin tournament between the co-evolving entities (Jaśkowski *et al*., 2008; Samothrakis *et al*., 2012; Szubert *et al*., 2013a), or (iii) an estimated expected utility (the average score obtained against a random sample of tests) (Chong *et al*., 2012; Jaśkowski *et al*., 2013).

*Corresponding author

A common feature of all conventional performance measures (both heuristic and the exact ones) is that they aggregate the results of multiple interactions into a single scalar value. Though convenient and compact, the performance value obtained in this manner tells very little about the differences between the compared solutions. In particular, Jaśkowski (2011) showed that aggregation leads to compensation: the rewards received in interactions with a group of tests can cancel out the penalties incurred in interactions with another group of tests. As a result, solutions can receive the same value of the performance measure, even if the results of their interactions with the same tests are completely different.

The compensation of interaction outcomes affects the internal dynamics of algorithms that use them to drive the search, as well as the *post-hoc* comparison of solutions they produce. In this paper, we address this problem in the latter context, proposing a means for a many-aspect assessment of solutions produced by algorithms applied to any search-based problems. Jaśkowski *et al*. (2013) introduced the *performance profile*, a multi-criteria performance evaluation method that characterizes performance using, rather than a scalar, a *vector* of results against tests of various difficulty. Here, we extend that work in four directions. First, we formalize the performance profiles. Second, we introduce an evolutionary method of sampling tests for performance profiles, which allows us to obtain robust performance estimates on tests varying from the most difficult to the easiest ones. Third, we demonstrate the versatility of performance profiles by applying them not only to Othello (as in our previous work (Jaśkowski *et al*., 2013)), but also to a variant of Iterated Prisoner's Dilemma. Last, we carry out a comparative analysis of performance profiles of a well-performing evolved Othello player and a set of players known from the previous work, including a hand-designed standard heuristic player, a temporal difference learning player, and a strategy learned by preference learning. The observed differences, which would pass unnoticed or remain unexplained when using scalar performance measures, provide new insights into the characteristics of the algorithms considered.

2. Performance profiles

2.1. Test-based problems. Test-based problems (Bucci *et al*., 2004; de Jong, 2004) belong to a broad class of co-optimization problems (Popovici and De Jong, 2009). While being aware of various framings of this concept (Popovici *et al*., 2011), we rely here on the following, most useful in practice, definition.

A *test-based problem* is the quadruple $(\mathcal{S}, \mathcal{T}, \mathcal{G}_S, Q_\mathcal{T})$, where

- $\mathcal{S}$ is a space of *candidate solutions*,
- $\mathcal{T}$ is a set of *tests*, on which candidate solutions are evaluated,
- $\mathcal{G}_s : \mathcal{T} \to \mathbb{R}$ is a *payoff function* for a candidate solution $s \in \mathcal{S}$ so that $\mathcal{G}_s(t)$ returns the result of an interaction between s and a test t, and
- $Q_\mathcal{T} : \mathcal{S} \to \mathbb{R}$ is a candidate solution *quality function*.

The goal in a test based-problem is to find a candidate solution $s^* \in \mathcal{S}$ that maximizes $Q_\mathcal{T}$.

The quality function of interest in this paper is the *expected utility*, which for an $s \in \mathcal{S}$ is defined as

$$Q_\mathcal{T}(s) = \mathbf{E}_{t \in \mathcal{T}} \left[\mathcal{G}_s(t) \right], \tag{1}$$

where $\mathbf{E}$ is the expectation operator.

The expected utility performance measure corresponds to the maximization of the *expected utility solution concept* (Ficici, 2004) in co-optimization and is also known as *generalization performance* (Chong *et al.*, 2008; 2009). Examples of other solutions concepts considered in the literature include the Pareto optimal set, the Nash equlibrium (Ficici, 2004) and the correlated equlibrium (Hart and Mas-Colell, 2000).

Trivial problems aside, it is computationally infeasible to calculate (1), because the set of tests $\mathcal{T}$ is too large. Instead, an estimator is used, leading to the *approximate quality function*:

$$\hat{Q}_T(s) = \frac{1}{|T|} \sum_{t \in T} \mathcal{G}_s(t), \tag{2}$$

where $T \subset \mathcal{T}$ is a (computationally manageable) subset of tests. When $t \in T$ are uniformly drawn from $\mathcal{T}$, $\hat{Q}_T$ is an unbiased estimator of $Q_\mathcal{T}$.

2.2. Test difficulty. A quality function, whether exact or approximate, aggregates multiple interaction outcomes. This makes it possible for them to cancel each other, which in turn blurs the differences between candidate solutions, and in an extreme case renders them apparently indistinguishable. To alleviate this problem, we propose to 'multi-objectivize' (term borrowed from Knowles *et al*. (2001)) the assessment of candidate solutions and present the underlying information in a structural way.

Let us notice that while candidate solutions differ in quality, tests can be likewise said to vary in difficulty. A test can be said to be *difficult* if a candidate solution is expected to get a low payoff from an interaction with it; and *vice versa*: it is *easy* if a candidate solution is expected to get a high payoff on it.

In order to formalize test difficulty, we assume that (i) an interaction of a candidate solution s and a test t, besides producing a payoff for s, generates also a payoff for t, denoted by $\mathcal{G}_t(s)$, and (ii) that $\mathcal{G}_t(s)$ fulfills $\mathcal{G}_t(s) + \mathcal{G}_s(t) = C$ for all $s \in S$ and $t \in T$, where C is

a problem-specific constant. Without loss of generality, we assume that $C = 1$ and $0 \leq \mathcal{G}_t, \mathcal{G}_s$, thus $\mathcal{G}_t, \mathcal{G}_s \leq 1$. For example, if an Othello player s wins against a player t, $\mathcal{G}_s(t) = 1$ and $\mathcal{G}_t(s) = 0$; when s loses against t, $\mathcal{G}_s(t) = 0$ and $\mathcal{G}_t(s) = 1$, and $\mathcal{G}_s(t) = \mathcal{G}_t(s) = 0.5$ in the case of draw.

We define the difficulty of a test as the following function $D_{\mathcal{S}} : \mathcal{T} \to \mathbb{R}$:

$$D_{\mathcal{S}}(t) = \mathbf{E}_{s \in \mathcal{S}} \left[\mathcal{G}_t(s) \right] = \mathbf{E}_{s \in \mathcal{S}} \left[1 - \mathcal{G}_s(t) \right].$$

Note that both the quality of a candidate solution and the difficulty of a test range in $[0, 1]$.

By analogy to solution quality, computing $D_{\mathcal{S}}(t)$ is infeasible in practice, so we approximate it using a finite sample:

$$\hat{D}_S(t) = \frac{1}{|S|} \sum_{s \in S} \mathcal{G}_t(s),$$

where $S \subset \mathcal{S}$ is a (computationally manageable) subset of tests. When $s \in S$ are uniformly drawn at random from $\mathcal{S}$, $\hat{D}_S$ is an unbiased estimator of $D_{\mathcal{S}}$.

Notice that for symmetric problems, where $\mathcal{S} = \mathcal{T}$, every candidate solution is a test, and *vice versa*. In such domains, the higher the quality of a solution, the more difficult it is as a test, i.e., $D_{\mathcal{S}}(t) = Q_{\mathcal{T}}(s)$ for $s = t \in \mathcal{S} = \mathcal{T}$. However, performance profiles, introduced in the following section, handle asymmetric problems as well.

2.3. Performance profile. The key idea of the method presented in this paper is *to characterize the performance of a candidate solution as a function of test difficulty*. We will call such a function a *performance profile* of a candidate solution. We define the performance profile p_s of a candidate solution $s \in \mathcal{S}$ as

$$p_s(d) = Q_{\mathcal{T}_d}(s) \text{ for } d \in [0, 1] \text{ such that } \mathcal{T}_d \neq \emptyset, \quad (3)$$

where $\mathcal{T}_d \subset \mathcal{T}$ is the set of all tests of difficulty d, i.e., $\mathcal{T}_d = \{t \in \mathcal{T} | D_{\mathcal{S}}(t) = d\}$. Let us note that $p_s(d)$ is undefined when there are no tests of difficulty d.

In general, p_s may be incomputable, because there may be infinitely many difficulty values d for which $p_s(d)$ is defined, and for each such d the set of tests $\mathcal{T}_d$ can be infinite or large. Thus, to estimate p_s, we discretize difficulty by splitting it into disjoint intervals of equal width. For example, in this paper, we use 100 bins of width 0.01. We define the discretized profile P_s as

$$P_s(B) = Q_{\mathcal{T}_B}(s) \text{ for } B \in \mathcal{B} \text{ such that } \mathcal{T}_B \neq \emptyset, \quad (4)$$

where B is a bin, $\mathcal{B}$ is the set of bins, and $\mathcal{T}_B \subset \mathcal{T}$ is a set of tests t of difficulty $D(t) \in B$. Notice that P_s is undefined for empty bins.

As in practice $\mathcal{T}_B$ can be too large to compute $Q_{\mathcal{T}_B}$, we fall back to its approximation $\hat{Q}_{T_B}$ (Eqn. (2)), where T_B is a (computationally manageable) sample of $\mathcal{T}_B$.

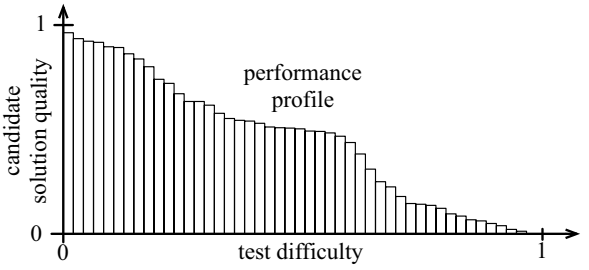

Fig. 1. Sample performance profile. The height of each bar (y-axis) represents the quality of the candidate solution when interacting with tests of difficulty represented by the range occupied by the bar on the x-axis.

Figure 1 presents the performance profile of an imaginary candidate solution. Each bar represents the performance on a separate bin. We expect typical performance profiles to be weakly decreasing functions of test difficulty, since it is usually easier to get higher payoff from interactions with easier tests than from the more difficult ones. However, there are no fundamental reasons that would prevent a performance profile from taking on an arbitrary shape.

Let us notice that each bin B gives rise to a separate performance criterion, and $P_s(B)$ is the quality of candidate solution s on that criterion. In other words, a bin determines one dimension of candidate solution characteristics. In this sense, the performance profile can be considered a multi-criteria performance measure.

In a related work, Ashlock and Lee (2013) presented agent-case embedding, which could be also used for characterizing and comparing the performance of solutions of a test-based problem. In contrast to performance profiles, which provide a multi-objective solution evaluation on tests of increasing difficulty, agent-case embeddings measure the diversity of evolved phenotypes and visualize them in the Euclidean space.

2.4. Test sampling methods. The fidelity of a discretized performance profile (Eqn. (4)) with respect to its exact counterpart (Eqn. (3)) depends on the characteristics of the test samples supporting particular bins. Ideally, bins should be backed by statistical evidence of the same strength, which in turn means that every one of them should contain the same number of tests, and the tests in each bin should be generated independently.

We propose two methods for generating test samples for bins: *random sampling* and *evolutionary sampling*. Both methods make an attempt to fill every bin up to *bin capacity* N ($|T_B| = N$ for $B \in \mathcal{B}$). This can be computationally demanding, especially when both N and the number of bins are high. Nevertheless, this is a one-off process: once the bins have been filled up, they can be

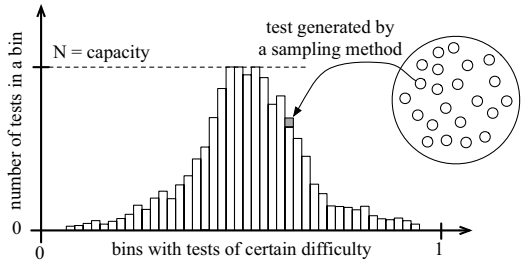

Fig. 2. Visual illustration of a random sampling method for generating tests for the bins.

used *ad infinitum* to assess the performance profiles of arbitrary many candidate solutions.

2.4.1. Random sampling. In random sampling (Jaśkowski *et al.*, 2013), we fill the bins with tests via repetitive independent sampling. In each iteration, we draw at random a test t from the set of all tests $\mathcal{T}$, estimate its difficulty, and place it in the appropriate bin if that bin's capacity has not yet been reached; otherwise, the test is discarded. The difficulty of a test is approximated using M candidate solutions drawn at random from $\mathcal{S}$, independently for every evaluated test (see Fig. 2).

The advantage of this method is that it guarantees the independence of tests, within every bin as well as across bins. We thus call the bins generated in this way *unbiased.*

Unfortunately, random sampling does not scale well with bin capacity N and the number of bins. Typically, some bins can be easily filled up, but filling up others is computationally infeasible. For example, in Othello it is difficult to draw very weak or very strong players (which are, at the same time, very easy or very difficult tests, respectively) at random. We can expect to run into a similar problem for most nontrivial symmetric test-based problems: a problem for which a very good candidate solution can be easily generated *at random* is simple. This makes us abandon this technique in this study, and employ a more sophisticated evolutionary sampling that provides more balanced bin occupancy.

2.4.2. Evolutionary sampling. In the face of challenges troubling the random sampling algorithm, we propose *evolutionary sampling.* In this method, we run an evolutionary process that evolves a population of tests, where a test's fitness is defined as its difficulty, approximated using a small number of candidate solutions ($n\_samples_{fit} = 200$). At every generation, we pick the fittest test and calculate a more accurate estimate of its difficulty using a greater number of candidate solutions (e.g., $n\_samples_{diff} = 1\,000$). If this estimate matches a bin that has not yet reached its capacity, the test is placed in that bin and the evolutionary process is stopped. The pseudocode of this procedure is shown in Algorithm 1.

Algorithm 1. Evolutionary sampling.

```
1: function EVOL-SAMPLING(n_samples_fit, n_samples_diff,
   N, B, pop_size, max_gen)
2:   for B ∈ B do
3:     T_B ← ∅
4:   end for
5:   S ← SAMPLE-SOLUTIONS(n_samples_diff)
                                  ▷ for difficulty estimation
6:   while not stopped do
7:     F ← SAMPLE-SOLUTIONS(n_samples_fit)
                                  ▷ for fitness function
8:     P ← SAMPLETESTS(pop_size)   ▷ initial population
9:     for gen ← 1, max_gen do
10:      P ← EVOLVE-NEXT-GENERATION(P, D̂_F)
11:      t ← arg max_{x∈P} D̂_F(x)
12:      d ← D̂_S(t)
13:      B ← B' ∈ B : d ∈ B'
14:      if |T_B| < N then
15:        T_B ← T_B ∪ {t}
16:        break
17:      end if
18:    end for
19:  end while
20:  return {T_B}_{B∈B}
21: end function
```

The main advantage of evolutionary sampling is its capability to generate tests of extreme values of difficulty, i.e., the very difficult and the very easy ones. To provide for both, we run two types of evolutionary processes. In the first one, fitness is defined as test difficulty, so evolution is driven to produce tests of increasing difficulty in consecutive generations. In the second type, fitness is the *negated* difficulty of a test. Thus, evolution tends to produce the easy tests (note that the extremely easy tests may be as rare as the extremely difficult ones). Compared with random sampling, evolutionary sampling is more likely to fill the extreme bins (the far-left and the far-right ones) up to the desired capacity N. From a practical perspective, having well populated difficult bins is usually more important, as this part of the performance profile provides information on how a given candidate solution copes with the most challenging tests.

On the downside, evolutionary sampling is biased. Although each test is a result of an independent evolutionary run, the underlying evolutionary processes may favor certain parts of the test space. However, in Section 6 we show that, at least for Othello, the bias is in practice negligible.

3. Coevolutionary algorithms

In Sections 4 and 5, we will use performance profiles to characterize and compare the candidate solutions produced by different flavors of *coevolutionary algorithms*, stochastic metaheuristics which, in their *compet-*

itive flavor (Reynolds, 1994), are popular tools for solving test-based problems (Popovici *et al*., 2011). Similarly to conventional evolutionary algorithms, a coevolutionary one maintains a population of candidate solutions, and uses the fitness function to select some of them and thus form the next generation of solutions. The fitness function is typically some estimate of the expected utility (Eqn. (1)), and as such requires a sample of tests (T in Eqn. (2)). In the simplest *one-population coevolution* (Luke and Wiegand, 2002), the other candidate solutions in the population serve that purpose. This limits the applicability of this approach to *symmetric* problems (where $\mathcal{S} = \mathcal{T}$), and, more importantly, is arguably controversial, as candidate solutions are to maximize performance, while the tests in T should differentiate them and thus provide a learning gradient (Juillé and Pollack, 1998). This led to the concept of *two-population competitive coevolution* (Nolfi and Floreano, 1998), where tests evolve in a second separate population.

Note that, by being based on interactions between the constantly changing entities (whether in one or in two populations), the fitness function in coevolutionary algorithms is *internal* and *subjective*, contrary to conventional evolutionary algorithms, where fitness evaluation is independent for each candidate solution and thus *external* and *objective*. As a result, a highly fit candidate solution is not guaranteed to be objectively good. Therefore, the coevolutionary dynamics are usually more complex than in conventional evolutionary search (Watson and Pollack, 2001). Nonetheless, coevolutionary algorithms belong nowadays to the most successful methods for learning game strategies (Pollack and Blair, 1998).

In the following subsections we describe three one-population and two two-population (co)evolutionary algorithms considered in this study. All of them employ the $(\mu+\lambda)$ generational evolution strategy (Beyer and Schwefel, 2002) independently for each maintained population. A population is initialized with μ randomly generated individuals (candidate solutions or tests). In every generation, each of the μ fittest individuals produces λ/μ offspring via mutation (thus, all populations consist of μ parents and λ offspring of those parents). Following Chong *et al*. (2012), we use $\mu = 25$ and $\lambda = 25$.

The algorithms vary only in the way they assign fitness to individuals, which is illustrated in Fig. 3. More specifically, the fitness of the candidate solutions is defined as the approximate quality function $\hat{Q}_T$ (Eqn. (2)), but the algorithms differ in the way they choose the sample of tests T.

3.1. Evolutionary learning with random sampling (EVOL-RS). EVOL-RS (Fig. 3(a)) is a degenerated one-population coevolutionary algorithm in which candidate solutions in the population do not interact with each other. Instead, each candidate solution is evaluated against an external set of random opponents. The sample of tests T is drawn at random from $\mathcal{T}$ once per generation. In order to maintain the same number of interactions per generation as in the coevolutionary methods, we set $|T| = \mu + \lambda = 50$.

EVOL-RS was shown to surpass one-population coevolution on generalization performance for 1-ply Othello and Iterated Prisoner's Dilemma (Chong *et al*., 2012).

3.2. One-population coevolution (1-COEV). 1-COEV (Fig. 3(a)) is a one-population coevolutionary algorithm. All candidate solutions in population interact with each other (in a round-robin tournament). Formally then, the sample T is simply the current population.

3.3. One-population coevolution with random sampling (1-COEV-RS). 1-COEV-RS (Fig. 3(a)) is a hybrid of 1-COEV and EVOL-RS that combines the competitive fitness with random sampling. Technically, the sample T is filled in half by the other candidate solutions drawn uniformly from the current population, and in half by the tests drawn at random from $\mathcal{T}$.

3.4. Two-population coevolution (2-COEV). 2-COEV (Fig. 3(b)) is a two-population competitive coevolutionary algorithm, where individuals are bred in two separate populations, one for the candidate solutions and one for the tests. The population of tests employs a $(\mu + \lambda)$ evolutionary strategy, where $\mu = 25$ and $\lambda = 25$. The fitness of a candidate solution s is $\hat{Q}_T(s)$ with the sample T being the current population of tests. Conversely, the fitness of a test t is $\hat{D}_S(t)$ with S being the current population of candidate solutions.

3.5. Two-population coevolution with random sampling (2-COEV-RS). 2-COEV-RS (Fig. 3(b)) is 2-COEV hybridized with random sampling. The fitness of a candidate solution s is $\hat{Q}_T(s)$, with T filled in half by the tests from the current population of tests, and in half by the tests generated at random. Compared to 2-COEV, the population of tests in this method is half the size of the population of candidate solutions. Tests' fitness is assessed as in 2-COEV.

4. Experimental analysis of Iterated Prisoner's Dilemma

In this section, we apply the algorithms presented in Section 3 to the Iterated Prisoner's Dilemma game and compare the resulting strategies using the single-objective expected utility and the performance profiles.

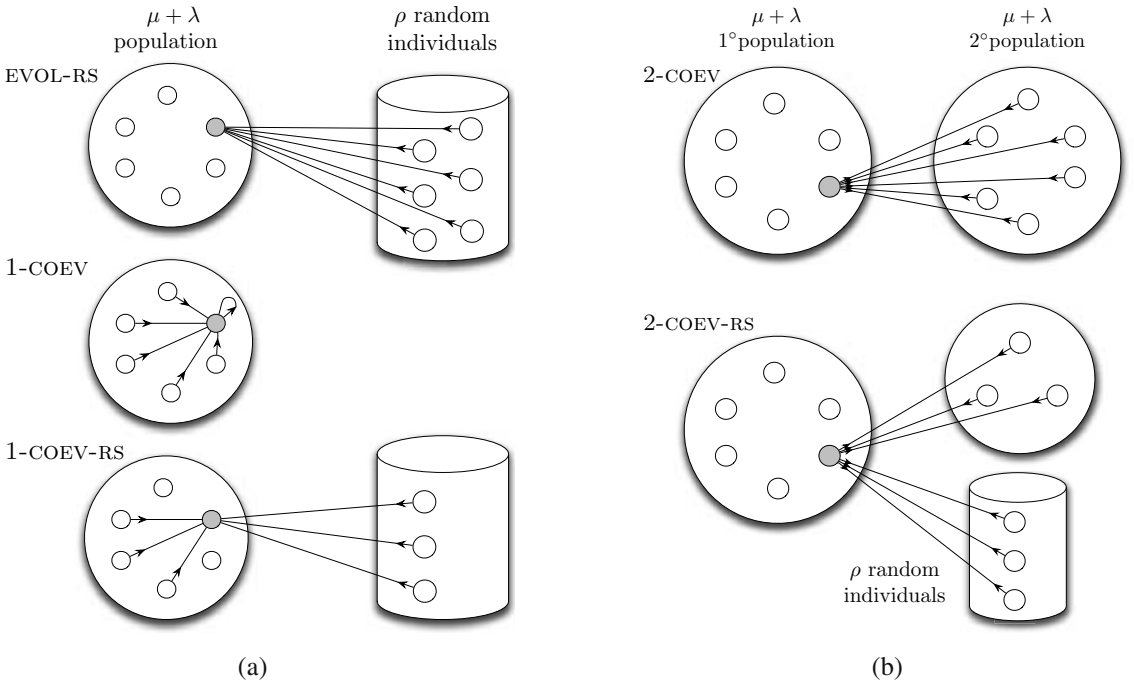

Fig. 3. Visualization emphasizes the differences in fitness assignment among methods considered in the paper. An arrow means that a game is played between two players: EVOL-RS and 1-COEV-* (a), 2-COEV-* (b).

Table 1. Payoff matrix in the classic Prisoner's Dilemma. A tuple (p_A, p_B) denotes the payoffs for players A and B, respectively. R is the payoff granted to each player for mutual cooperation, S for cooperating when the other player defects, T for defecting when the other player cooperates, and P for mutual defection.

		Player B	
	choice	Cooperate	Defect
Player A	Cooperate	(R, R)	(S, T)
	Defect	(T, S)	(P, P)

4.1. Classical Prisoner's Dilemma. The classical Iterated Prisoner's Dilemma (IPD) is a two-player game involving a series of interactions, each of which is a Prisoner's Dilemma (PD) game. In PD, a player can make one of two choices: cooperate or defect. The PD payoff matrix, shown in Table 1, must satisfy two conditions: (i) $T > R > P > S$ and (ii) $2R > S + T$. The first inequality ensures that defection is the most profitable action, and that the payoffs resulting from mutual cooperation are greater than those arising from mutual defection. The second inequality is IPD-specific and ensures that a series of mutual cooperations pays off more than a series of alternating defections and cooperations (Poundstone, 1992). IPD is a sequence of PD interactions where each player remembers its and its opponent's move from one or more previous interactions.

4.2. n-Choice Prisoner's Dilemma. In this study, we consider IPD extended to multiple choices (or *levels of cooperation*) (Darwen and Yao, 2001; Chong and Yao, 2005). In an *n-choice Prisoner's Dilemma* game, the choices are encoded as values from the set $C = \{2i/n - 1 | i \in \{0, \ldots, n-1\}\}$, where the extreme values -1 and 1 mean full defection and full cooperation, respectively. The payoff values must satisfy the following conditions: (i) $p(c_A, c_B) > p(c'_A, c_B)$, (ii) $p(c_A, c_B) < p(c_A, c'_B)$, and (iii) $p(c'_A, c'_B) > \frac{1}{2}(p(c_A, c'_B) + p(c'_A, c_B)$ for any choices $c_A, c_B, c'_A, c'_B \in C$ such that $c_A < c'_A$ and $c_B < c'_B$ (Frean, 1996). We will use the following payoff function that meets the above constraints:

$$p(c_A, c_B) = \begin{cases} 2.5 - 0.5c_A + 2c_B & \text{for player } A, \\ 2.5 - 0.5c_B + 2c_A & \text{for player } B. \end{cases}$$

4.3. Strategy representation. Different strategy representations for coevolutionary learning of IPD such as finite state machines (Fogel, 1991) and neural networks (Darwen and Yao, 2000) have been studied in the past. Here, we adopt the arguably simplest one, the direct look-up table (Axelrod, 1984), and make the players remember the moves from the previous iteration only (*memory-one* IPD). In that case, the n-choice IPD strategy is an $n \times n$ matrix M, where m_{ij} for $i, j = 1, 2, \ldots, n$ specifies the choice to be made given the player's own previous move i and the opponent's previous move j.

The other element of the strategy is the initial move m_{00}, which does not depend on the opponent's strategy.

4.4. Experimental setup. In the experiments, we focus on IPD with $n = 9$ choices (levels of cooperation), which we found to be much more demanding than 3-choice IPD used in earlier coevolutionary investigations by Chong *et al*. (2012). Thus, each strategy is a look-up table containing $9 \times 9 + 1 = 82$ choices and the size of search space is $9^{82} \approx 1.77 \times 10^{78}$.

Although IPD is primarily used to study cooperation (Axelrod, 1984), we consider it here, following recent works by Chong *et al.* (2009; 2012), a competitive domain.

Each IPD game consists of 150 PD episodes. To assess the result of an interaction of two IPD strategies we compute their cumulative payoff over the episodes. The highest cumulative score indicates the winner, which is assigned the interaction payoff 1, while the loser gets 0. In the case of draw, the interaction results in 0.5 points for both strategies.

As discussed in Section 3, all algorithms considered here maintain a population of 50 candidate solutions which interact with the same number of tests. As a result, in each generation, $50 \times 50 = 2{,}500$ IPD games are played. Since each evolutionary run consists of 200 generations, it requires the total effort of 500,000 games.

All methods start with an initial population filled with candidate solutions (strategies) randomly drawn from the space of direct look-up tables. The only search operator used by the algorithms is a simple mutation which iterates over all elements of the look-up table and with probability $p_{mut} = 0.2$ replaces the original choice with one of the remaining $n - 1$ choices, selected at random. Chong and Yao (2005) found the adopted mutation operator to provide sufficient variation of strategy behaviors for an IPD game with multiple choices.

Some of our coevolutionary algorithms and performance assessment methods employ *random players*. Every random player is obtained independently by filling the look-up table with random choices. In the following, by a 'random player/opponent' we mean a player obtained in this way. Note that this definition of a random player differs from the one that assumes selecting each action by uniformly drawing it from a set of all available actions in a given position. It is, however, coherent with the expected utility measure defined on the set of all tests (see Section 2.3). A random player is a test drawn at random from the set $\mathcal{T}$. Having said that, the performances of the random players obtained in both ways are similar.

We performed 120 runs for each method presented in Section 3. In the following, the best-of-generation candidate solution is the individual with the highest fitness in the population of candidate solutions (where fitness is subjective and specific for a given method; see Section 3). By the best-of-run solution we mean the best-of-generation player of the last generation. In the following, we analyze those players using expected utility (Section 4.5) and performance profiles (Section 4.6).

Table 2. Expected utilities and 95% confidence intervals of best-of-run individuals obtained by five algorithms for Iterated Prisoner's Dilemma.

Algorithm	Expected utility
1-COEV-RS	0.9832 ± 0.0035
EVOL-RS	0.9676 ± 0.0042
2-COEV-RS	0.9561 ± 0.0085
1-COEV	0.9263 ± 0.0126
2-COEV	0.9091 ± 0.0131

4.5. Results for the expected utility. To estimate the expected utility of an individual, we let it play 50,000 games against random players. With 1 point for winning the game, 0 for losing, and 0.5 for a draw, the expected utility of a player is in the range of $[0, 1]$. In this section, the term 'performance' refers to this measure.

Table 2 presents the average performance of the best-of-run individuals for each algorithm accompanied by 95% confidence intervals.

To compare the algorithms, we performed statistical analysis with a significance level $\alpha = 0.01$ using a nonparametric Kruskal–Wallis rank sum test, which revealed a statistically significant ($\chi^2 = 116.7$, p-value $< 2.2 \times 10^{-16}$) difference between the results obtained by particular algorithms. A post-hoc analysis using the pairwise Wilcoxon rank sum test with the Holm correction indicated the following differences:

$$\text{1-COEV-RS} > \text{EVOL-RS} = \text{2-COEV-RS} > \text{1-COEV} = \text{2-COEV},$$

where '$>$' denotes significant difference and '$=$' means no statistical difference.

Let us first discuss the results of 'pure' methods that use homogeneous sets of opponents, i.e., EVOL-RS, 1-COEV, and 2-COEV. The observed relationship between these methods confirms the previous findings by Chong *et al*. (2012) that evolutionary learning guided by fitness estimates based on random sampling (EVOL-RS) achieves a higher expected utility when compared to the simple coevolutionary learning approach (1-COEV). We also observe that coevolution of two autonomous populations of candidate solutions and tests (2-COEV) is not beneficial in terms of the expected utility.

The methods that use a mixture of competitive fitness and random sampling (1-COEV-RS, 2-COEV-RS) turn out to be able to evolve strategies with the highest expected

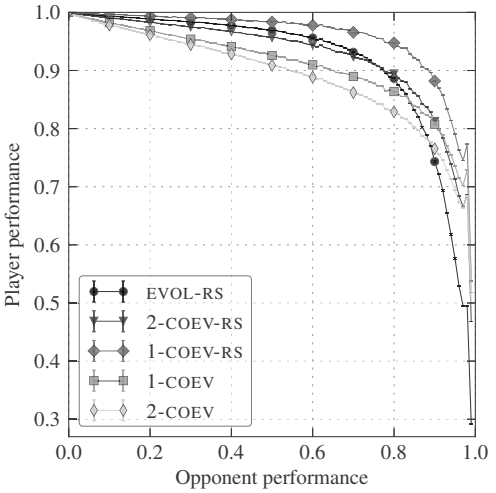

Fig. 4. Performance profiles for Iterated Prisoner's Dilemma, for the opponent's performance ranging from 0 to 1. Each point (x, y) in a plot indicates the average performance y when playing against the opponents of performance x. This and all subsequent graphs feature 95% confidence intervals, though for many bins they are very narrow.

utility. There is no statistical difference between 2-COEV-RS and EVOL-RS, but 1-COEV-RS is clearly the best algorithm for this problem, resulting in the highest median and the lowest variance.

Though the results demonstrate the positive effect of hybridizing different fitness functions, the measure of the expected utility does not reveal any details about the strengths or weaknesses of the evolved individuals. For instance, EVOL-RS and 2-COEV-RS produce players of roughly the same expected performance, but do they differ in the capability of winning with the opponents of particular strengths? In the following we demonstrate how this question can be conveniently answered using performance profiles.

4.6. Analysis with performance profiles. In order to scrutinize the best-of-run individuals produced by each algorithm, we apply performance profiles (cf. Section 2.3). We used evolutionary sampling (cf. Section 2.4) with $(25 + 25)$-ES ($n\_samples_{fit} = 200$) to generate 100 tests samples, each corresponding to a bin of width 0.01. Each bin's sample was filled up with $N = 5\,000$ tests, where the difficulty of every test was evaluated on the basis of games with $n\_samples_{diff} = 10\,000$ random players.

Figure 4 shows the performance profiles averaged over the best-of-run individuals produced by 120 runs of every algorithm. A point at coordinates (x, y) indicates the average performance y when playing against the opponents of performance x. For instance, the performance of 2-COEV is about 0.9 for the opponents with performance of 0.5 (by which we mean the opponents with performance in the range of $[0.5, 0.51)$, since bin width is 0.01). Recall that IPD is a symmetric problem, thus a player's difficulty (when it acts as a test) is equal to its quality (when it acts as a candidate solution).

The decreasing trend in each profile confirms the supposition that the stronger opponents are harder to defeat than the weaker ones. The only exception of the decreasing trend is the bin $[0.98, 0.99)$, which is 'easier' than the bin $[0.97, 0.98)$ for all algorithms. Despite some effort, we are unable to explain this artifact.

Some methods clearly dominate others. The profile of 1-COEV-RS dominates all other profiles, which explains its best result in terms of the expected utility (cf. Table 2). Also, 1-COEV dominates 2-COEV. The statistical analysis conducted in Section 4.5 did not reveal them as significantly different because difficult tests are few and far between in the random sample used to estimate the expected utility. On the contrary, the rightmost bins of performance profiles host many such tests. We take this as evidence that 1-COEV should be preferred to 2-COEV for this problem, *although they perform the same on average*.

Other profiles are mutually non-dominated—their plots cross each other. In this respect, the most interesting is the EVOL-RS profile. Although Table 2 suggests no significant differences between 2-COEV-RS and EVOL-RS, their profiles reveal that 2-COEV-RS copes with the strong opponents much better, while EVOL-RS is more effective against the weaker ones. Such a profile shape reflects the method's trade-off in the ability to cope with opponents of various strength. The single-criteria performance measures like, for instance, expected utility, are not able to pinpoint such differences and therefore are much less descriptive.

Moreover, the analysis with performance profiles shows that for the strongest opponents (performance $>$ 0.9) EVOL-RS is worse not only than 1-COEV, but even than 2-COEV, which ranks last on expected utility. For the opponents of the last bin (difficulty 0.97), the expected interaction outcome of EVOL-RS is worse by 0.17–0.26 than for the other algorithms.

5. Experimental analysis of 1-ply Othello

In this section, we apply the five algorithms considered to the game of Othello and compare the resulting strategies using the single-objective expected utility, performance profiles and a round robin tournament.

5.1. 1-ply Othello.

5.1.1. Problem definition. The game of Othello is a deterministic, perfect information, zero-sum board game

Table 3. Expected utilities and 95% confidence intervals of best-of-run individuals obtained by five algorithms for Othello.

Algorithm	Expected utility
2-COEV-RS	0.866 ± 0.0024
EVOL-RS	0.8624 ± 0.0023
1-COEV-RS	0.8371 ± 0.0036
1-COEV	0.7997 ± 0.0052
2-COEV	0.7963 ± 0.0064

played by two players on an 8×8 board. There are 64 identical pieces which are white on one side and black on the other, with the colors representing the players. The game starts with the four central squares of the board occupied with two black and two white pieces. Players make moves alternately by placing their pieces on the board until it is completely filled or until neither of them is able to make a legal move. The location to place a piece on has to fulfill two conditions: (i) be adjacent to the opponent's piece, and (ii) form a vertical, horizontal, or diagonal line with another player's piece, with a continuous sequence of the opponent's pieces in between. After placing a piece, all such opponent pieces are flipped. A legal move requires flipping at least one of the opponent's pieces. The objective of the game is to have the majority of pieces on the board at the end of the game. If both players have the same number of pieces on the board, the game ends in a draw.

5.1.2. Strategy representation. We represent Othello strategies using position-weighted piece counter (WPC). The WPC is a linear weighted board evaluation function which implements the *state evaluator* concept; i.e., it is explicitly used to evaluate how desirable is a given board state. It assigns a weight w_i to a board location i and uses scalar product to calculate the utility f of a board state $\mathbf{b}$:

$$f(\mathbf{b}) = \sum_{i=1}^{8\times 8} w_i b_i,$$

where b_i is 0 in the case of an empty location, $+1$ if a black piece is present or -1 in the case of a white piece. The players interpret $f(\mathbf{b})$ conversely: the black player prefers the moves leading to the states with a higher value, whereas the lower values are favored by the white player.

We employ the WPC as a state evaluator in a 1-ply setup: given the current state of the board, the player generates all legal moves and applies f to the resulting states. The state gauged as the most desirable determines the move to be made. Ties are resolved at random.

5.2. Experimental setup. To maintain a learning environment similar to that used for IPD, we retain most of the evolutionary parameters such as the number of generations, population sizes and the total effort per generation (cf. Section 4). In order to learn strategies that are able to play both sides, we assume that a single interaction is a double game, where each of interacting individuals plays one game as a black player and one game as a white player. With a population size of 50, this leads to $2\,500$ interactions (double games) and $5\,000$ single games per generation. In each single game, half a point is divided between the players: the winner gets 0.5 point and the loser 0 points, or they get 0.25 points each in case of a draw. Thus, the result of an interaction is in the $[0, 1]$ range, as it was for IPD.

The population is initialized with random players whose weights are uniformly drawn from the range $[-0.2, 0.2]$. The only search operator used by all algorithms is a mutation that perturbs all the weights w_i with additive noise:

$$w_i^{'} = w_i + 0.1 \cdot U[-1, 1],$$

where $U[-1, 1]$ is a real number drawn uniformly from $[-1, 1]$. Weights resulting from mutation are clamped to the interval $[-10, 10]$. Consequently, the space of strategies is a $[-10, 10]^{64}$ hypercube. As in the case of IPD, we performed 120 runs for each algorithm.

5.3. Results for the expected utility. We start the performance analysis of the best-of-run solutions by using the scalar measure of the expected utility, which we estimate via $25\,000$ double games ($50\,000$ games in total) against the random WPC players. Table 3 reports the results of this experiment.

We performed the same statistical analysis as for IPD in Section 4.5 and obtained the following partial ordering of algorithms:

$$\begin{aligned} &\text{2-COEV-RS} = \text{EVOL-RS} \\ &\qquad > \text{1-COEV-RS} > \text{1-COEV} = \text{2-COEV}, \end{aligned}$$

where '$>$' denotes significant difference and '$=$' means no statistical difference at a significance level $\alpha = 0.01$.

What this result has in common with the IPD ranking is the superiority of the methods involving random sampling. However, the relationship between them is not the same. In particular, 2-COEV-RS is now better than 1-COEV-RS, which is, in turn, worse than EVOL-RS.

5.4. Analysis with performance profiles. To generate the samples of tests for bins (each defined as an $[x, x + 0.01)$ range of performance), we used evolutionary sampling engaging a $(25 + 25)$ evolutionary strategy with $max_{gen} = 10\,000$. The fitness and difficulty of each individual were approximated with double games against $n\_samples_{fit} = 200$ and $n\_samples_{diff} = 1000$ random players, respectively. In contrast to IPD, despite

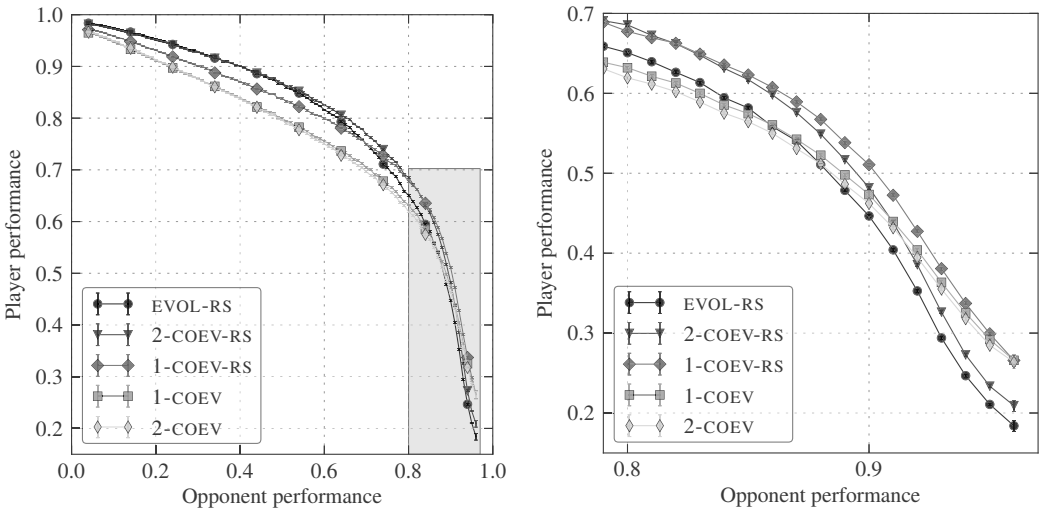

Fig. 5. Performance profiles for Othello in the 0.0–1.0 range of opponent performance—each point (x, y) in a plot indicates the average performance y when playing against the opponents of performance x: performance profiles in the 0.0–1.0 range of opponent performance (grayed region is zoomed on the right) (a), performance profiles zoomed to the 0.8–0.96 range of opponent performance (b).

computing on 60 cores of modern CPUs for a few days, we were not able to fill up all the buckets to the assumed capacity of $N = 1000$ opponents. The first three and the last three bins (performance ranges of $[0, 0.03)$ and $[0.97, 1]$) remained empty, and the bins $[0.03, 0.04)$ and $[0.96, 0.97)$ were filled only partially. In total, 100 samples of tests contain 91727 opponents of performances ranging in $[0.03, 0.97]$.

Figure 5 shows the average performance profiles for the best-of-run Othello players evolved by particular algorithms. In contrast to IPD, it is hard to observe any dominance between the profiles, except for 1-COEV-RS, which dominates both 1-COEV and 2-COEV.

Noteworthy, in the large part of the opponent difficulty range (performance of 0.0–0.6), the order of profiles is consistent with the ranking obtained with the single-criteria measure of the expected utility. However, the order changes dramatically for the strongest opponents. Strikingly, EVOL-RS and 2-COEV-RS, the two best algorithms according to the statistical analysis based on the expected utility, become the two worst ones when confronted against the strongest opponents (see Fig. 5(b)). In contrast, 1-COEV and 2-COEV, the two worst algorithms in terms of the expected utility, are significantly better than both EVOL-RS and 2-COEV-RS on the rightmost bins, showing performance similar to 1-COEV-RS.

Clearly, the performance profiles reveal the strong points of 1-COEV, 2-COEV and 1-COEV-RS, which could not be noticed using the expected utility. However, attaining higher performance against the stronger opponents is not sufficient to compensate the inferior position when it comes to mediocre opponents, because the latter ones occur much more frequently in an unbiased sample used to estimate the expected utility.

5.5. Round-robin tournament. The round-robin tournament is a popular method that determines a valued ranking of methods by playing matches between teams of players they produced (Jaśkowski *et al.*, 2008; Samothrakis *et al.*, 2012). The important conceptual difference with respect to other performance indicators considered above is the *direct* confrontation between the solutions produced by particular algorithms (rather than referring to an external sample of opponents). In this way, the round-robin tournament provides a different means for performance assessment that can be used as an alternative to the expected utility.

In our tournament, every team consists of 120 best-of-run players produced by a certain algorithm. Thus, a single match involves $120 \times 120 = 14,400$ double games. By a 'match score' and the 'tournament score' we mean, respectively, a team's average score obtained in a single match or in the whole round-robin tournament.

Table 4 presents the results of the tournament for the Othello players produced by particular algorithms. By bootstrapping the outcomes of double games, we calculated also 95% confidence intervals of the scores. We present them for the total score in square brackets. For particular matches between methods the confidence intervals were repeatedly very close to $[x - 0.006; x + 0.006]$, so they were omitted.

Evolutionary learning with random sampling loses to all other algorithms in head-to-head matches and its aggregated overall score is significantly lower (confidence intervals do not overlap) than 1-COEV-RS, 2-COEV-RS, and 1-COEV and 2-COEV. Adding the random sampling component improves both one- and two-population coevolution (1-COEV-RS vs. 1-COEV and 2-COEV-RS vs. 2-COEV). Also, the one-population variants are consistently better than two-population ones (2-COEV vs. 1-COEV and 2-COEV-RS vs. 1-COEV-RS). 1-COEV-RS wins against all the other algorithms and is clearly the best in terms of the round-robin score.

The ranking according to the expected utility presented in Table 3 is different than the one resulting from the round-robin tournament. In general, one cannot expect them to match up due to the absolute nature of the former and the relative nature of the latter. However, how can we explain the differences and reconcile these results? This question can be addressed by further examination of performance profiles.

5.6. Performance profiles explain the round-robin tournament and the expected utility. Performance profiles allow us to explain the discrepancy between the rankings based on the expected utility and the round-robin tournament. Let us first notice that, contrary to the expected utility assessments which involved random opponents, in the round-robin tournament each individual in a team plays only with the players from the opponent teams, and those players (co)evolved to be strong. According to Table 3, the performance of team members is in the range of 0.79 to 0.87, i.e., well above 0.5, the expected performance of a random player.

Let us analyze the profiles in Fig. 5(b) in this range. The two best algorithms, with performance between 0.79 to 0.87, are 1-COEV-RS (unquestionably) and 2-COEV-RS (better than the remaining algorithms on most bins). The other three algorithms are significantly worse but certain differences between them are still observable. In particular, 1-COEV and 2-COEV start to surpass EVOL-RS when test difficulty reaches ~ 0.87. 1-COEV is subtly but consistently better than 2-COEV in this test difficulty interval. This order is consistent with the ranking obtained in the round robin tournament.

Now let us use the performance profiles to explain the order of algorithms induced by the expected utility measure presented in Section 5.3. In that case, games are played against random players, whose performance is 0.5 on average (a random player is equally likely to win and lose a game against another a random player). Moreover, the random player's performance is most often very close to 0.5 (the standard deviation of a random player's performance is 0.28, because well- and bad-performing random players are few and far in between). Thus, it is not surprising that in Fig. 5, for opponents of difficulty 0.5, the order of algorithms is consistent with the ranking obtained for the expected utility in Section 5.3.

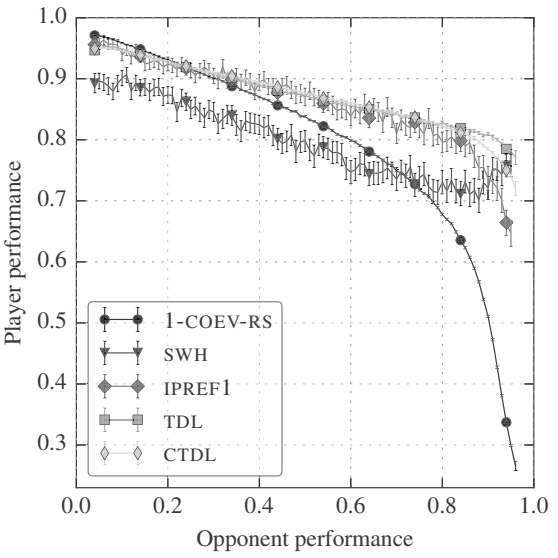

Fig. 6. Performance profiles of the five players—the hand-crafted SWH and four obtained by the following learning methods: coevolution with random sampling (1-COEV-RS), coevolutionary temporal difference learning (CTDL), temporal difference learning (TDL) or preference learning (IPREF1). Whiskers mark the 95% confidence intervals.

5.7. Performance profiles of selected published players. Up to this point, our analysis has concerned the players produced by (co)evolutionary algorithms. Despite the differences we pointed out, all of the profiles have a similar course, with high performance for weak opponents that monotonically decreases with the opponents getting stronger. An interesting question is: Does a performance profile of every Othello player look like that?

To answer this question we analyzed the profiles of four players obtained by different methods:

1. Standard WPC Heuristic (SWH) player, hand-crafted by Yoshioka *et al.* (1998), and often used as an opponent in Othello research (Lucas and Runarsson, 2006; Szubert *et al.*, 2009; Manning, 2010). Its expected utility performance is 0.787 ± 0.002.

2. TDL player (87.26 ± 0.32), obtained by Jaśkowski *et al.* (2014) using temporal difference learning.

3. CTDL player (0.906 ± 0.001), trained by Szubert *et al.* (2011) using a hybridization of coevolution and temporal difference learning (Szubert *et al.*, 2009; 2011; Krawiec *et al.*, 2011).

Table 4. Round-robin tournament scores for the five coevolutionary algorithms in Othello. The total scores are followed by 95% confidence intervals.

	Match scores					**Tourn. score [%]**
Algorithm	1-COEV-RS	2-COEV-RS	1-COEV	2-COEV	EVOL-RS	
1-COEV-RS	–	.522	.521	.526	.556	**.531** [.528, .534]
2-COEV-RS	.478	–	.509	.518	.536	**.510** [.508, .513]
1-COEV	.479	.491	–	.507	.513	**.497** [.494, .500]
2-COEV	.474	.482	.493	–	.5	**.487** [.484, .490]
EVOL-RS	.444	.464	.487	.5	–	**.474** [.471, .477]

4. IPREF1 player (0.869 ± 0.001), obtained by Runarsson and Lucas (2014), the only player in this list that represents its strategy using (a simple variant of) n-tuple networks instead of WPC.

Figure 6 presents how the profiles of the above players compare with the average profile of 1-COEV-RS (performance 0.837 ± 0.004). Note that the confidence intervals for SWH and IPREF1 are higher because these profiles are based on a single player, while the profiles of 1-COEV-RS, CTDL and TDL are averaged over, respectively, 120 and 30 runs.

Interestingly, no profile strictly dominates all others. CTDL has the highest performance, but, except SWH, the other players are more effective in the left-hand side of the spectrum. Also, the plots show that the flattest profiles characterize the players obtained by the methods that employ temporal difference learning (TDL or CTDL).

The profile of the handcrafted SWH player is significantly different from those obtained by learning algorithms. While the performance of the latter generally decreases with the opponents getting stronger, the SWH player does not strictly subscribe to that trend. Its profile crosses the 1-COEV-RS and IPREF1 curves at about 0.7 and 0.9, respectively, and surpasses them afterwards significantly, even though its overall performance is lower. More surprisingly, the SWH profile seems then to reverse its trend, and copes better with the strongest opponents (performance of 0.9–0.96) than with the slightly weaker ones (performance of 0.8–0.9). Strategies with such characteristics could have been considered as candidate solutions during the runs of the other methods that also use WPC as the underlying representation. However, their poor overall performance destined them to be dropped in favor of strategies that perform better against the mediocre and weak players.

6. Bias of evolutionary sampling

In our previous work involving performance profiles (Jaśkowski *et al.*, 2013; Szubert *et al.*, 2013b), we used the unbiased random sampling method (see Section 2.4) to generate bin samples. Here, we rely on evolutionary sampling, which trades bias for better occupancy of extreme bins. It is then desirable to quantitatively assess the extent of that bias.

To this aim, we compare the performance profiles obtained using random sampling with the ones generated by means of evolutionary sampling. Random sampling is much worse at generating the strongest and the weakest players than evolutionary sampling. As a result, it fills up only the bins in the interval $[0.17, 0.83]$ for IPD, and $[0.17, 0.78]$ for Othello. For these intervals, in Fig. 7, we plot the *differences* between the corresponding profiles, which show the bias of evolutionary sampling.

Let us notice first that the bias is predominantly positive for IPD, while for Othello it can be either positive or negative. The bias characteristics is thus problem-dependent.

Secondly, the figure shows that the bias of evolutionary sampling is generally low. The maximum absolute differences between profiles are around 0.04 for IPD and (only) 0.015 for Othello. The bias is generally growing with increasing the opponent's performance, and can be expected to be higher for the bins right of 0.8. Nevertheless, the bias is similar for all methods (except for 1-COEV for IPD), thus its influence on the ranking of algorithms is limited. We can, therefore, assume that the bias of evolutionary sampling does not preclude the resulting samples of tests from being reliable performance indicators.

7. Conclusions

In this study, we presented and formalized the technique of performance profiles and demonstrated its usefulness for comparing solutions for test-based problems, and, implicitly, learning algorithms. Performance profiles proved to be capable of revealing the differences in characteristics between the algorithms that have a similar expected utility and explaining the discrepancy between the outcomes of the round-robin tournament and the expected utility. Because they abstract from the internals of learning algorithms, nothing precludes them to be applied to other than best-of-run solutions, and we anticipate that they can be useful for, e.g., explaining the dynamics of a search process and characterizing the behavior (Szubert *et al.*, 2015) of other (i.e.,

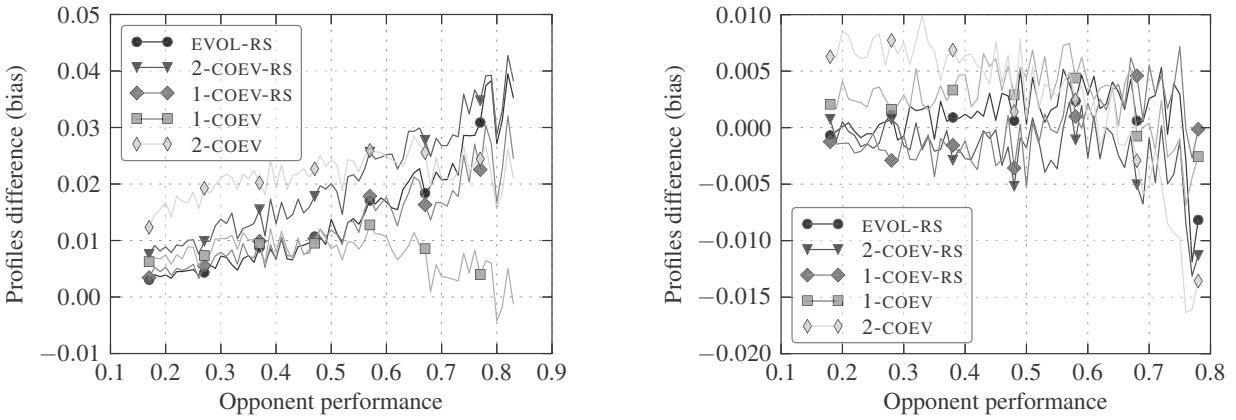

Fig. 7. Differences between profiles obtained using random sampling and evolutionary sampling: IPD (a), Othello (b).

non-evolutionary) algorithms (Jaśkowski *et al.*, 2014). In prospect, we envision them as a valuable tool providing researchers with a feedback on algorithm's characteristics, thus helping them design new approaches.

Although here we applied the performance profiles to two *symmetric* test-based problems, we defined them in a general way, and thus they are applicable to the asymmetric case as well (when $\mathcal{T} \neq \mathcal{S}$). The only formal requirement is that the sum of payoffs obtained in a single interaction by a solution and a test needs to be constant.

For a relatively simple problem such as IPD, simply drawing samples at random may be sufficient to populate a sufficiently wide range of bins and reveal the differences between the algorithms in question. For harder problems like Othello, a more sophisticated evolutionary sampling of tests may be necessary to populate the extreme bins and obtain an assessment on the very strong opponents, which may be important in practice. Such instrumentation of profiles proved helpful in this study. However, the more powerful strategy representations, like n-tuples (Lucas, 2007; Jaśkowski, 2014), may provide players with performance near 1.0 against all random opponents. In such cases, the specific samples used in this paper may turn out to be insufficient to reveal any differences, and even more sophisticated sampling methods (or more computational effort) may be required. In any case, we would recommend assessing bias by confronting (where possible) the bin sample with the random one as we did in this study.

When it comes to contributions to research in coevolutionary algorithms, this study presented evidence that coevolution can offer an advantage over evolution with random sampling for learning game strategies. This advantage was observable for both IPD and Othello in the right side of the performance profiles and in head-to-head comparison for Othello. We conclude thus that an algorithm that autonomously and dynamically redefines its own fitness function (1- and 2-COEV) can produce strategies which are better in such confrontation than those produced by an algorithm that relies on an (approximately) stationary fitness function (EVOL-RS). However, pure coevolution is not sufficient to attain that, and additional means like involvement of random opponents are necessary.

Acknowledgment

This work has been supported by the Polish National Science Centre grant no. DEC-2013/09/D/ST6/03932. M. Szubert acknowledges the support through the grant no. DEC-2012/05/N/ST6/03152 and K. Krawiec through the grant no. 2014/15/B/ST6/05205.

References

Ashlock, D. and Lee, C. (2013). Agent-case embeddings for the analysis of evolved systems, *IEEE Transactions on Evolutionary Computation* **17**(2): 227–240.

Axelrod, R. (1984). *The Evolution of Cooperation*, Basic Books, New York, NY.

Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies—a comprehensive introduction, *Natural Computing* **1**(1): 3–52.

Bucci, A., Pollack, J.B. and de Jong, E. (2004). Automated extraction of problem structure, *in* K. Deb *et al.* (Eds.), *Genetic and Evolutionary Computation—GECCO-2004, Part I*, Lecture Notes in Computer Science, Vol. 3102, Springer-Verlag, Berlin/Heidelberg, pp. 501–512.

Chong, S.Y., Tiňo, P., Ku, D.C. and Xin, Y. (2012). Improving generalization performance in co-evolutionary learning, *IEEE Transactions on Evolutionary Computation* **16**(1): 70–85.

Chong, S.Y., Tiňo, P. and Yao, X. (2008). Measuring generalization performance in coevolutionary learning,

IEEE Transactions on Evolutionary Computation **12**(4): 479–505.

Chong, S.Y., Tiňo, P. and Yao, X. (2009). Relationship between generalization and diversity in coevolutionary learning, *IEEE Transactions on Computational Intelligence and AI in Games* **1**(3): 214–232.

Chong, S.Y. and Yao, X. (2005). Behavioral diversity, choices and noise in the iterated prisoner's dilemma, *IEEE Transactions on Evolutionary Computation* **9**(6): 540–551.

Darwen, P.J. and Yao, X. (2001). Why more choices cause less cooperation in iterated prisoner's dilemma, *Proceedings of the 2001 Congress on Evolutionary Computation, Seoul, South Korea*, Vol. 2, pp. 987–994.

Darwen, P. and Yao, X. (2000). Does extra genetic diversity maintain escalation in a co-evolutionary arms race, *International Journal of Knowledge-Based Intelligent Engineering Systems* **4**(3): 191–200.

de Jong, E.D. (2004). The incremental Pareto-coevolution archive, *in* K. Deb *et al.* (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference*, Lecture Notes in Computer Science, Vol. 3102, Springer-Verlag, Berlin/Heidelberg, pp. 525–536.

Ficici, S.G. (2004). *Solution Concepts in Coevolutionary Algorithms*, Ph.D. thesis, Brandeis University, Waltham, MA.

Fogel, D.B. (1991). The evolution of intelligent decision making in gaming, *Cybernetics and Systems* **22**(2): 223–236.

Fogel, D.B. (2001). *Blondie24: Playing at the Edge of AI*, Morgan Kaufmann Publishers, San Francisco, CA.

Frean, M. (1996). The evolution of degrees of cooperation, *Journal of Theoretical Biology* **182**(4): 549–59.

Hart, S. and Mas-Colell, A. (2000). A simple adaptive procedure leading to correlated equilibrium, *Econometrica* **68**(5): 1127–1150.

Hillis, W.D. (1990). Co-evolving parasites improve simulated evolution as an optimization procedure, *Physica D* **42**(1–3): 228–234.

Jaśkowski, W. (2011). *Algorithms for Test-Based Problems*, Ph.D. thesis, Poznań University of Technology, Poznań.

Jaśkowski, W. (2014). Systematic n-tuple networks for Othello position evaluation, *ICGA Journal* **37**(2): 85–96.

Jaśkowski, W. and Krawiec, K. (2011). How many dimensions in cooptimization?, *in* N. Krasnogor (Ed.), *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, NY, pp. 829–830.

Jaśkowski, W., Krawiec, K. and Wieloch, B. (2008). Evolving strategy for a probabilistic game of imperfect information using genetic programming, *Genetic Programming and Evolvable Machines* **9**(4): 281–294.

Jaśkowski, W., Liskowski, P., Szubert, M. and Krawiec, K. (2013). Improving coevolution by random sampling, *in* C. Blum (Ed.), *GECCO'13: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ACM, Amsterdam, pp. 1141–1148.

Jaśkowski, W., Szubert, M. and Liskowski, P. (2014). Multi-criteria comparison of coevolution and temporal difference learning on Othello, *in* A.I. Esparcia-Alcazar and A.M. Mora (Eds.), *EvoApplications 2014*, Lecture Notes in Computer Science, Vol. 8602, Springer, Berlin/Heidelberg, pp. 301–312.

Juillé, H. and Pollack, J.B. (1998). Coevolving the ideal trainer: Application to the discovery of cellular automata rules, *Proceedings of the 3rd Annual Conference on Genetic Programming, Madison, WI, USA*, pp. 519–527.

Knowles, J.D., Watson, R.A. and Corne, D. (2001). Reducing local optima in single-objective problems by multi-objectivization, *EMO'01: Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization, Zurich, Switzerland*, pp. 269–283.

Krawiec, K., Jaśkowski, W. and Szubert, M. (2011). Evolving small-board go players using coevolutionary temporal difference learning with archive, *International Journal of Applied Mathematics and Computer Science* **21**(4): 717–731, DOI: 10.2478/v10006-011-0057-3.

Lucas, S.M. (2007). Learning to play Othello with n-tuple systems, *Australian Journal of Intelligent Information Processing Systems* **9**(4): 1–20.

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

Luke, S. and Wiegand, R.P. (2002). When coevolutionary algorithms exhibit evolutionary dynamics, *in* A.M. Barry (Ed.), *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, AAAI, New York, NY, pp. 236–241.

Manning, E.P. (2010). Using resource-limited Nash memory to improve an Othello evaluation function, *IEEE Transactions on Computational Intelligence and AI in Games* **2**(1): 40–53.

Nolfi, S. and Floreano, D. (1998). Coevolving predator and prey robots: Do "Arms races" arise in artificial evolution?, *Artificial Life* **4**(4): 311–335.

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

Popovici, E., Bucci, A., Wiegand, R.P. and de Jong, E.D. (2011). Coevolutionary principles, *in* G. Rozenberg *et al.* (Eds.), *Handbook of Natural Computing*, Springer-Verlag, Berlin/Heidelberg, pp. 987–1033.

Popovici, E. and De Jong, K. (2009). Monotonicity versus performance in co-optimization, *FOGA'09: Proceedings of the 10th ACM SIGEVO Workshop on Foundations of Genetic Algorithms, Orlando, FL, USA*, pp. 151–170.

Poundstone, W. (1992). *Prisoner's Dilemma: John von Neuman, Game Theory, and the Puzzle of the Bomb*, Doubleday, NY.

Reynolds, C. (1994). Competition, coevolution and the game of tag, *in* R.A. Brooks and P. Maes (Eds.), *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, MIT Press, Cambridge, MA, pp. 59–69.

Runarsson, T. and Lucas, S. (2014). Preference learning for move prediction and evaluation function approximation in Othello, *IEEE Transactions on Computational Intelligence and AI in Games* **6**(3): 300–313.

Samothrakis, S., Lucas, S., Runarsson, T. and Robles, D. (2012). Coevolving game-playing agents: Measuring performance and intransitivities, *IEEE Transactions on Evolutionary Computation* **17**(2): 1–15.

Szubert, M., Jaśkowski, W. and Krawiec, K. (2009). Coevolutionary temporal difference learning for Othello, *IEEE Symposium on Computational Intelligence and Games, Milan, Italy*, pp. 104–111.

Szubert, M., Jaśkowski, W. and Krawiec, K. (2011). Learning board evaluation function for Othello by hybridizing coevolution with temporal difference learning, *Control and Cybernetics* **40**(3): 805–831.

Szubert, M., Jaśkowski, W. and Krawiec, K. (2013a). On scalability, generalization, and hybridization of coevolutionary learning: A case study for Othello, *IEEE Transactions on Computational Intelligence and AI in Games* **5**(3): 214–226.

Szubert, M., Liskowski, P., Jaśkowski, W. and Krawiec, K. (2013b). Shaping fitness function for evolutionary learning of game strategies, *in* C. Blum (Ed.), *GECCO'13: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ACM, Amsterdam, pp. 1149–1156.

Szubert, M., Jaśkowski, W., Liskowski, P. and Krawiec, K. (2015). The role of behavioral diversity and difficulty of opponents in coevolving game-playing agents, *in* M.A. Mora and G. Squilero (Eds.), *EvoApplications 2015*, Lecture Notes in Computer Science, Vol. 9028, Springer, pp. 394–405.

Watson, R.A. and Pollack, J.B. (2001). Coevolutionary dynamics in a minimal substrate, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), San Francisco, CA, USA*, pp. 702–709.

Yoshioka, T., Ishii, S. and Ito, M. (1998). Strategy acquisition for the game "Othello" based on reinforcement learning, *in* S. Usui and T. Omori (Eds.), *Proceedings of the Fifth International Conference on Neural Information Processing, ICONIP98*, IOA Press, Kitakyushu, pp. 841–844.

Wojciech Jaśkowski received B.Eng., M.Sc. and Ph.D. degrees in computing science from the Institute of Computing Science of the Poznań University of Technology, Poland, in 2004, 2006 and 2011, respectively. Currently he is an assistant professor at the Laboratory of Intelligent Decision Support Systems there. He is an author of more than 30 publications on computational intelligence. His research interests concern evolutionary computations, reinforcement learning, competitive co-evolution, test-based problems, genetic programming, visual learning, and learning strategies for games.

Paweł Liskowski received the B.Eng. and M.Sc. degrees in computing science from the Poznań University of Technology, Poland, in 2011 and 2012. Currently he is a research assistant and a Ph.D. student at the Laboratory of Intelligent Decision Support Systems, Institute of Computing Science, Poznań University of Technology. Primary areas of his research interest include various subfields of artificial intelligence. In particular, his work includes co-evolution for test-based problems, genetic programming and machine learning.

Marcin Szubert received his B.Eng., M.Sc. and Ph.D. degrees in computing science from the Poznań University of Technology, Poland, in 2008, 2009 and 2014, respectively. Currently he is an assistant professor at the Laboratory of Intelligent Decision Support Systems, Institute of Computing Science, Poznań University of Technology. His research interests primarily cover the area of artificial intelligence with a focus on evolutionary computation and machine learning with applications to game playing. Particularly, he works on hybridizing reinforcement learning methods and coevolutionary algorithms for developing artificial neural networks.

Krzysztof Krawiec received his Ph.D. and habilitation degrees in 2000 and 2004 from the Poznań University of Technology, Poland, where he is currently an associate professor. His work includes program synthesis, in particular semantics and program behavior in genetic programming; coevolutionary algorithms and test-based problems; evolutionary computation for machine learning, primarily for learning game strategies and for synthesis of pattern recognition systems; and applications in medicine, biology, and climate science. Dr. Krawiec is the author of over 100 publications on the above and related topics and an associate editor of the *Genetic Programming and Evolvable Machines* journal.

Received: 13 February 2015
Revised: 20 May 2015
Re-revised: 7 July 2015