Coordinate System Archive for Coevolution

Wojciech Jaśkowski and Krzysztof Krawiec, *Member, IEEE*

***Abstract*—Problems in which some entities interact with each other are common in computational intelligence. This scenario, typical for co-evolving artificial-life agents, learning strategies for games, and machine learning from examples, can be formalized as *test-based problem*. In test-based problems, candidate solutions are evaluated on a number of test cases (agents, opponents, examples). It has been recently shown that at least some of such problems posses *underlying problem structure,* which can be formalized in a notion of coordinate system, which spatially arranges candidate solutions and tests in a multidimensional space. Such a coordinate system can be extracted to reveal *underlying objectives* of the problem, which can be then further exploited to help coevolutionary algorithm make progress. In this study, we propose a novel coevolutionary archive method, called Coordinate System Archive (COSA) that is based on these concepts. In the experimental part, we compare COSA to two state-of-the-art archive methods, IPCA and LAPCA. Using two different objective performance measures, we find out that COSA is superior to these methods on a class of artificial problems (COMPARE-ON-ONE).**

I. Introduction

A canonical coevolutionary algorithm evolves a population of (candidate) *solutions* and a population of *tests* based on the results of elementary interactions between them. This simple scheme is applicable to a surprisingly wide scope of *test-based problems* [1], including learning game strategies, machine learning from examples, artificial life, etc. Because an outcome of a single interaction is usually not informative enough, multiple outcomes are typically aggregated to drive the evaluation and selection during evolution. Unfortunately, the aggregation of outcomes is one of the reasons for which coevolutionary algorithms often suffer from so-called *pathologies*: over-specialization, loss of gradient, cycling [2] or disengagement [3]. These phenomena are not problematic for natural evolution that has no externally imposed goals, but make it difficult to force a coevolutionary algorithm to make a steady progress towards a specific *solution concept* [4] posed by the researcher.

In *Pareto-coevolution* [5], [6] proposed to overcome some of these these drawbacks, each test gives rise to a separate objective that orders the solutions with respect to how well they fare against it, and thus the aggregation is no longer required. This transforms the test-based problem into a multiobjective optimization problem and allows relying on the well-defined concept of dominance – solution s_1 is not worse than solution s_2 if and only if s_1 performs at least as good as s_2 on all tests. Unfortunately, in real test-based problems the number of tests is usually prohibitively large; take for example the number of strategies in chess. Therefore, also the dimensionality of search space in Pareto-coevolution could be enormous.

Fortunately, it was shown [7] that at least some test-based problems possess an *underlying problem structure*, which manifests itself by the existence of groups of tests that examine the same skill or aspect of solution performance, but with different intensity. Such tests can be ordered with respect to difficulty and placed on a common axis to form a new objective that replaces the constituent (old) objectives, leading to reduction of the dimensionality of the search space. These so-called *underlying objectives* [8] are typically not know *a priori* and have to be revealed during exploration of the problem. For instance, underlying objectives in chess could measure skills of controlling the center of the board, using knights, playing endgames, etc.

The above intuition about underlying objectives and internal structure of a problem was first formalized in the notion of *coordinate system* in [1] and then further investigated in [9], the studies which our work is based on. An important feature of coordinate system is that while *compressing* the initial set of objectives, it preserves the relations between solutions and tests. Each solutions is embedded in the system and the outcome of its interaction with any test can be determined given its position on all axes.

The idea of extracting and using the underlying problem structure to support the progress in coevolution was first applied in Dimension Extraction Coevolutionary Algorithm (DECA) [10]. Here we propose another method, called Coordinate System Archive (COSA), that is based on similar principles. Our method differs substantially from the earlier attempt, since DECA uses a different definition of coordinate system (see Section IV) and exploits it in a different way. COSA is a proof-of-concept that the coordinate system defined in [1] can be also successfully used in a coevolutionary archive algorithm.

The paper is organized as follows. We start with formally introducing all the necessary concepts and elaborating on the coordinate system defined by Bucci [1] in Sections II, III, and IV. After describing in details our method in Section V, we present our co-evolutionary framework in Section VI. This section describes also the LAPCA and IPCA algorithms, the problem COMPARE-ON-ONE and the experimental setup. Finally, we discuss the results in Section VII.

II. Mathematical Preliminaries

Definition 1: Partially ordered set (*poset,* for short) is a pair (X, P), where X is a set and P is a reflexive, antisymmetric, and transitive binary relation on X. We call X the *ground set* while P is a *partial order* on X.

W. Jaśkowski and K. Krawiec are with the Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60965 Poznań, Poland; email: {wjaskowski,kkrawiec}@cs.put.poznan.pl.

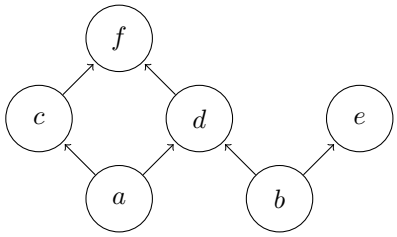
We write $x \leq y$ in P when $(x, y) \in P$ and $x \geq y$ in P when $(y, x) \in P$. The notations $x < y$ in P and $y > x$ in P mean $x \leq y$ in P and $x \neq y$. When the context is obvious, we will abbreviate $x < y$ in P by just writing $x < y$.

Definition 2: For a poset (X, P), $x, y \in X$ are *comparable* ($x \perp y$) when either $x \leq y$ or $x \geq y$; otherwise, x and y are *incomparable* ($x \parallel y$).

Definition 3: A poset (X, P) is called a *chain* if every pair of elements from X is comparable. When (X, P) is a chain, we call P a *linear order* on X. Similarly, we call a poset an *antichain* if every pair of distinct elements from X is incomparable. A chain (respectively, antichain) (X', P') is a *maximum chain* (respectively, *maximum antichain*) in (X, P), $X' \subseteq X, P' \subseteq P$ if no other chain (respectively, *antichain*) in (X, P) has more elements than it.

Definition 4: The *width* of a poset (X, P), denoted as $\mathrm{width}(X, P)$, is the number of elements in its maximum antichain.

Theorem 1: (Dilworth [11]*) If (X, P) is a poset and $\mathrm{width}(X, P) = n$, then there exists a partition of $X = C_1 \cup C_2 \cup \cdots \cup C_n$, where C_i is a chain for $i = 1, 2, \ldots, n$.* We call it *minimum chain partition*, as it comprises the smallest possible number of chains. Note that an important consequence of Dilworth theorem is that each C_i contains exactly one element of the maximum antichain.



In (X, P) some of elements are comparable, e.g. $a \leq f$, $b \leq e$; others are incomparable, e.g., $c \parallel d$, $a \parallel e$, thus it is not a linear order. Examples of chains in (X, P) are $a < c$ and $b < d < f$. The latter is a maximum chain in (X, P). $\{a, e\}$ is an example of antichain, and $\{c, d, e\}$ is a maximum antichain of this poset, thus $\mathrm{width}(X, P) = 3$. $\{a, c, f\} \cup \{b, d\} \cup \{e\}$ is an example of a minimum chain partition. $\{c, d, e\}$ is the only antichain in P of size 3.

III. Test-Based Problem

In this paper, by *test-based problem* we will mean a formal object $\mathcal{G} = (S, T, G)$ that consists of a set S of *solutions* (a.k.a. candidate solutions [12], candidates [1], learners [13] or hosts [14]), a set T of *tests* (a.k.a. teachers, parasites), and an *interaction function* $G : S \times T \to \mathbb{R}$. We restrict our attention to the case where the codomain of G is a binary set $\{0, 1\}$. If $G(s, t) = 1$, we say that solution s *solves* test t; if $G(s, t) = 0$, we say that s *fails* test t. Where convenient, we will treat G as a relation and denote the fact that solutions s solves test t as $G(s, t)$ and the fact that it fails test t as $\neg G(s, t)$. Notice that in the perspective of game theory, G can be interpreted as a payoff matrix, S, T as sets of strategies and $\mathcal{G}$ as a game. Thus, in this paper we will use the terms 'test-based problem' and 'game' interchangeably.

S*olutions failed set* $SF(t) \subseteq S$ is comprised of all solutions that fail the test t. Dually, *tests solved set* $TS(s) \subseteq T$ is comprised of all tests that are solved by solution s. Notice also that $t \in TS(s) \iff s \notin SF(t)$ for all $s \in S$, $t \in T$, since both sides hold if and only if s solves t.

Test t_1 is *weakly dominated* by test t_2, written $t_1 \leq t_2$, when $SF(t_1) \subseteq SF(t_2)$ for $t_1, t_2 \in T$. Dually, solution s_1 is *weakly dominated* by solution s_2, written $s_1 \leq s_2$, when $TS(s_1) \subseteq TS(s_2)$ for $s_1, s_2 \in S$. For brevity we use the same symbol $\leq$ for both relations, as they are univocally determined by the context. Since $\leq$ inherits transitivity and reflexivity from $\subseteq$, it is a preorder in both S and T. To make $\leq$ a partial order we need to assume that no two elements of one set are indiscernible with respect to how they interact with the elements of the other set, precisely: $\nexists t_1, t_2 \in T, t_1 \neq t_2 : SF(t_1) = SF(t_2)$ and $\nexists s_1, s_2 \in S, s_1 \neq s_2 : TS(s_1) = TS(s_2)$. Under this assumption $s_1 = s_2 \iff TS(s_1) = TS(s_2)$ and, dually, $t_1 = t_2 \iff SF(t_1) = SF(t_2)$; thus $(S, \leq)$ and $(T, \leq)$ are posets, what eases our further arguments. In case when some indiscernible objects do exist (it can happen in practice), we can merge them into one object without losing any important features of $\mathcal{G}$.

IV. Coordinate System

In the context of test-based problems, a coordinate system is a formal concept revealing internal problem structure by enabling solutions and tests to be embedded into a multidimensional space. Of particular interests are such definitions of coordinate systems, in which the relations between solutions and tests ($\leq$) are reflected in spatial arrangement of their locations in the coordinate system. Previous work suggests that this formalism can help designing better coevolutionary algorithms [10] and examining properties of certain problems [7].

There is no unique definition of coordinate system for a test-based problem; currently we are aware of two formulations: by Bucci *et al.* [1] and by de Jong and Bucci [10], [7]. The difference between them lies in the way they define the axes. [1] defines an axis as a sequence of tests ordered by the domination relation, while [10] as a sequence of sets of solutions ordered by the inclusion relation. In the algorithm introduced in this paper we use the historically first definition of coordinate system introduced in [1], so in the following by coordinate system we mean the one defined there. There are slight differences in our formulation, which, however, do not affect any important properties of the coordinate system.

For convenience, we introduce a formal element t_0 such that $G(s, t_0)$ for all $s \in S$. Also, we define an operator 'overline' that augments a set of tests with t_0, i.e., $\bar{X} = X \cup \{t_0\}$.

Definition 5: A *coordinate system* $\mathcal{C}$ for a game $\mathcal{G}$ is a set of axes $(A_i)_{i \in I}$, where each axis $A_i \subseteq T$ is linearly ordered by $<$. I is an index set and the *size of the coordinate system*, denoted by $|\mathcal{C}|$, is the cardinality of I.

We interpret axis as an underlying objective of the problem. Tests on an axis are ordered with respect to increasing

difficulty ($<$ relation), so that every solution can be positioned on it according to the results of its interaction with these tests. The position of a solution is determined be the position function defined below.

Definition 6: Position function $p_i : S \to \bar{A}_i$ is a function that assigns a test from $\bar{A}_i$ to solution $s \in S$ in the following way:

$$p_i(s) = \max\{t \in \bar{A}_i | G(s,t)\}, \quad (1)$$

where the maximum is taken with respect to the relation $<$. The test $p_i(s)$ is the *position* of s on the axis $\bar{A}_i$.

To understand the above definition better, we show an important property of a coordinate system. Let $p_i(s) = t$. From the definition of position function p_i as the maximal test t for which $G(s,t)$, it follows immediately that $\neg G(s,t_1)$ for each $t_1 > t$. On the other hand, tests on axis A_i are linearly ordered by the relation $<$, which means that for each $t_1, t_2 \in A_i$, $t_1 < t_2$ when $SF(t_1) \subset SF(t_2)$. Thus, according to the definition of $SF(t)$, $G(s,t_2)$ for each $t_2 < t$. Consequently, if $A_i = \{t_1 < t_2 < \cdots < t_{k_i}\}$ is an axis and $p_i(s) = t_j$, we can picture s's placement on A_i in the following way [1]:

$$\begin{array}{ccccccccc} G(s,t) & 1 & 1 & \dots & 1 & 0 & \dots & 0 \\ \bar{A}_i & t_0 & t_1 & \dots & t_j & t_{j+1} & \dots & t_{k_i} \end{array}$$

As we can see, according to the position function, s is placed in such a way that for each axis it solves all tests on its left and fails all on its right.

Definition 7: Coordinate system $\mathcal{C}$ is *correct* for a game $\mathcal{G}$ iff for all $s_1, s_2 \in S$

$$s_1 \leq s_2 \iff \forall_{i \in I} p_i(s_1) \leq p_i(s_2)$$

Basically, this definition means that all relations between solutions in set S have to be preserved the coordinate system.

Definition 8: We say that test t *orders solution* s_1 *before solution* s_2, written $s_1 <_t s_2$, if $\neg G(s_1,t)$ and $G(s_2,t)$. Similarly, we use $s_1 =_t s_2$ when $G(s_1,t) = G(s_2,t)$, and $s_1 \leq_t s_2$ when $s_1 <_t s_2$ or $s_1 =_t s_2$. This is similar to the notion of *distinctions* [5]. Dually, we say that solution s *orders test* t_1 *before test* t_2, written $t_1 <_s t_2$, if $G(s,t_1)$ and $\neg G(s,t_2)$. Similarly, we use $t_1 =_s t_2$ when $G(s,t_1) = G(s,t_2)$, and $t_1 \leq_s t_2$ when $t_1 <_s t_2$ or $t_1 =_s t_2$

Definition 9: A correct coordinate system $\mathcal{C}$ is a *minimal coordinate system* for $\mathcal{G}$ if there does not exist any correct coordinate system for $\mathcal{G}$ with smaller size.

Definition 10: The *dimension* $\dim(\mathcal{G})$ of a game $\mathcal{G}$ is the size of the minimal coordinate system for $\mathcal{G}$.

A. Example

Let us consider an exemplary test-based problem from [7], i.e., the misère version of game of Nim-(1,3) with two piles of sticks: one containing a single stick and one containing three sticks. The exact rules of this game are not important here, but an interested reader is referenced to [7].

The payoff matrix of Nim-(1,3) is shown in Table I. There are a total of 144 strategies, but merging indiscernible strategies reduces the number of first player strategies to six (candidate solutions $s_1 - s_6$) and second player strategies to nine (tests $t_1 - t_9$).

Table I: The payoff matrix for Nim-(1,3) . An empty cell means 0.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
s_1	1		1	1		1	1		1
s_2									
s_3	1	1	1	1	1	1	1	1	1
s_4	1	1	1				1	1	1
s_5	1			1			1		
s_6							1	1	1

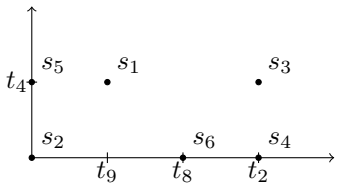


Figure 1: A minimal coordinate system for Nim-(1,3)

Figure 1 presents a minimal coordinate system for this game. We can see that the initial set of nine tests was "compressed" to only two underlying objectives represented by axes $A_1 = \{t_9 < t_8 < t_2\}$, $A_2 = \{t_4\}$. First, notice that the tests on both axes are placed according to the definition of coordinate system, that is in the order of increasing difficulty (in A_1, t_9 is less difficult than t_8, which is in turn less difficult than t_2). The correctness of this coordinate system can be verified by checking whether all relations between pairs of solutions are preserved (see conditions in Def. 7). For instance, consider a pair (s_1, s_3) for which $s_1 < s_3$: s_1 is also dominated by s_3 in the 2D space; on the other hand, $s_1 \| s_2$ (since $s_1 <_{t_8} s_2$ and $s_2 <_{t_1} s_1$), thus s_1 and s_2 do not dominate each other also in the figure. Second, solutions are placed with accordance to the position function. For example, s_6 is placed so that it solves t_9 and t_8, but fails t_4 and t_2, which is consistent with the relations in the original payoff matrix.

Finally, with a little of effort one could demonstrate that in this example $\mathrm{width}(T, \leq) = 3$ and the minimum chain partition of $(T, \leq)$ consists of the following chains: $(t_9, t_3, t_6), (t_8, t_2, t_5), (t_7, t_1, t_4)$.

V. Coordinate System Archive (COSA)

The goal of *coevolutionary archives* is to sustain progress during a coevolutionary run. A typical archive is a (usually limited in size, yet diversified) sample of well-performing individuals found so far. New individuals submitted to the archive are forced to interact with its members, who may be replaced when proved to be no more useful. Archives in coevolution may be seen as a counterpart of elitism in standard evolution. Historically, one of the oldest and simplest archives was Hall of Fame [15] that stores all the best-of-generation individuals encountered so far. Modern examples

Algorithm 1 COSA

1: **procedure** SUBMIT(S_{new}, T_{new})
2: $T \leftarrow T_{arch} \cup T_{new}$
3: $S \leftarrow S_{arch} \cup S_{new}$
4: $T \leftarrow$ GETUNIQUE(T, S)
5: $S \leftarrow$ GETUNIQUE(S, T)
6: $S_{Pareto} \leftarrow \{s \in S | \forall_{s' \in S} s' \leq_T s\}$
7: $T_{base} \leftarrow$ FINDTESTS(T, S_{Pareto})
8: $S_{req} \leftarrow$ PAIRSETCOVER$(S_{Pareto}, S, T_{base})$
9: $T_{req} \leftarrow$ PAIRSETCOVER$(T_{base}, T, S_{Pareto})$
10: $S_{arch} \leftarrow S_{req}$
11: $T_{arch} \leftarrow T_{req}$
12: **end procedure**
13:
14: **procedure** GETUNIQUE(A, B)
15: $U \leftarrow \emptyset$
16: **for** $a \in A$ **do**
17: $I \leftarrow \{e \in A | \forall_{b \in B} G(a, b) = G(e, b)\}$
18: $U \leftarrow U \cup$ oldest individual from I
19: $A \leftarrow A \setminus I$
20: **end for**
21: **return** U
22: **end procedure**
23:
24: **procedure** PAIRSETCOVER(A_{must}, A, B)
25: $A \leftarrow A \setminus A_{must}$
26: $\mathcal{N} \leftarrow \{(b_1, b_2) | b_1, b_2 \in B$ ▷ Pairs to be ordered
27: $\wedge \exists_{a \in A} b_1 <_a b_2) \wedge \nexists_{a \in A_{must}} b_1 <_a b_2)\}$
28: $\mathcal{V} \leftarrow A_{must}$
29: **while** $\mathcal{N} \neq \emptyset$ **do** ▷ Are all pairs ordered?
30: $u \leftarrow \text{argmax}_{a \in A \setminus \mathcal{V}} |\{(b_1, b_2) \in \mathcal{N} | b_1 <_a b_2\}|$
31: $\mathcal{N} \leftarrow \mathcal{N} \setminus \{(b_1, b_2) \in \mathcal{N} | b_1 <_u b_2\}$
32: $\mathcal{V} \leftarrow \mathcal{V} \cup \{u\}$
33: **end while**
34: **return** $\mathcal{V}$
35: **end procedure**

Algorithm 2 A procedure that finds the tests that should be kept in the archive

1: **procedure** FINDTESTS(T, S_{Pareto})
2: $\mathcal{C} \leftarrow$ CHAINPARTITION$(T, \leq)$
3: $n_{dims} = |\mathcal{C}|$
4: $S_{Sorted} \leftarrow S_{Pareto}$ sorted descendingly by $\min($GETPOS$(s, \mathcal{C}))$
5: **for** $s \in S_{Sorted}$ **do**
6: $T' \leftarrow \{t \in T | G(s, t)\}$
7: $(A, found) \leftarrow$ greatest antichain in poset $(T', \leq)$
8: **if** found **then**
9: **return** A
10: **end if**
11: **end for**
12: **return** T
13: **end procedure**
14:
15: **procedure** CHAINPARTITION(X, P)
16: **return** minimal chain partition of poset (X, P)
17: **end procedure**
18:
19: **procedure** GETPOS$(s, \mathcal{C})$
20: $n_{dims} \leftarrow |\mathcal{C}|$
21: $P \leftarrow array[1 \ldots n_{dims}]$
22: **for** $i = 1 \ldots n_{dims}$ **do**
23: $P[i] \leftarrow |\{c \in \mathcal{C}[i] | G(s, c)\}|$
24: **end for**
25: **return** P
26: **end procedure**

of archives include IPCA and LAPCA (see Sections VI-A and VI-B).

The idea of our novel archive algorithm is based on the concept of underlying problem structure and coordinate system, described in Section IV. The archive extracts a coordinate system of the game using the currently available solutions and tests and uses it to retain the *base set of tests*, which contains only one test from each axis. Apart from that, COSA maintains a set of Pareto non-dominated solutions. Details of the algorithm are described in the following.

The algorithm maintains two separate archives, one for solutions (S_{arch}) and one for tests (T_{arch}). Each time new solutions and tests (S_{new}, T_{new}) are submitted by evolution (line 10 of Alg. 3), COSA joins them with the archives into temporary sets of candidates, S and T (see Alg. 1, lines 2, 3). Ultimately, only some of these candidates will be retained.

The algorithm first gets rid of any duplicates in both archives (lines 4-5): from every equivalence class (group of individuals that are indiscernible in terms of the payoff matrix) only the oldest one is retained. Preferring the older individuals prevents replacing the objectively better individuals by the worse ones, which is likely due to predominantly destructive character of mutation and crossover typical for evolution.

In lines 6-7, COSA decides which individuals must be retained. For solutions, it is S_{Pareto}, the set of solutions that are mutually Pareto non-dominated with respect to T. For tests, it is the *base set of tests* T_{base} determined by the FINDTESTS procedure described in details later. However, S_{Pareto} and T_{base} are not sufficient to provide each other *stability*, i.e., prevent changing the mutual relationship between these individuals and, in consequence, being removed during subsequent submissions. Therefore COSA selects some additional individuals and forms supersets S_{req} and T_{req} of, respectively, S_{Pareto} and T_{base}, which become the new archives at the end of submission phase (lines 10 and 11). $S_{req} \subseteq S$ is a set of solutions that orders all pairs of tests from T_{base} that can be ordered. Formally, if for a pair $t_1, t_2 \in T_{base}$ there exists $s \in S$ such that $t_1 \leq_s t_2$, then S_{req} must contain s' such that $t_1 \leq_{s'} t_2$. Dually, T_{req} is a set of test that orders all pairs of tests from S_{Pareto} that can be ordered.

Both S_{req} and T_{req} are computed by PAIRSETCOVER,

which accepts generic arguments A and B. A_{must} is the set of individuals that have to be retained, thus pairs of elements of B ordered by individuals from A_{must} are not considered (line 27). $\mathcal{N}$ is the set of pairs that have to be ordered. Starting with set $\mathcal{V}$ containing only individuals from A_{must}, PAIRSETCOVER in each step extends it by such an element from A that orders the maximal number of not yet ordered pairs from $\mathcal{N}$. The procedure stops when all pairs are ordered. PAIRSETCOVER is a greedy heuristics and may not produce the minimal set of required elements, but has a very good logarithmic approximation ratio [16]; roughly speaking, it is the best possible polynomial-time approximation algorithm for this problem (c.f. [17]).

The most important part of the archive algorithm is the FINDTESTS procedure that determines the base set of tests (see Alg. 2). FINDTESTS builds up a coordinate system using CHAINPARTITION procedure, which finds a minimal chain partition $\mathcal{C}$ of poset $(T, \leq)$ (line 2). CHAINPARTITION implements the standard $O(|X|^3)$ algorithm (c.f. [18], [19]) that computes max-flow on a bipartite network with unit capacities[1]. Each chain of $\mathcal{C}$, being linearly ordered by $<$ (Def. 5), corresponds to one axis of the coordinate system, thus $\mathcal{C}$ is a coordinate system. This coordinate system is even correct (Def. 7), but the formal proof of this fact is beyond the scope of this paper.

The number of dimensions of the extracted coordinate system n_{dims} (line 3) is a temporal estimation of the true dimension of our game. It may be, but it does not have to be, the exact dimension of the test-based problem we are trying to solve, because (i) the extracted coordinate system is correct but not necessarily minimal, and (ii) we operate only on samples of all possible solutions and tests.

Having the coordinate system $\mathcal{C}$, we can easily determine the coordinates of each solution s in $\mathcal{C}$ by computing the number of tests that are solved by s on each axis (procedure GETPOS, lines 19-26). The position is represented by a vector of n_{dims} cardinal numbers (c.f. Def. 6).

In lines 4–11, FINDTESTS tries to find an antichain (a set of tests that are mutually incomparable) in T with three properties: (i) its size is n_{dims}, (ii) it is the *greatest antichain*, and (iii) there exists a solution that solves all its elements. Considering the first property, according to the Dilworth theorem (Theorem 1), such antichain always exists, since $\mathrm{width}(T, \leq) = n_{dims}$.

Let us explain the second property. Let $\mathcal{X}$ be the set of all maximum antichains of $(T, \leq)$ and $\mathcal{P}$ be a relation on $\mathcal{X}$ such that $X_1 \mathcal{P} X_2$ iff there exists a bijection $f : X_1 \to X_2$ such that $x_1 \leq f(x_1)$. Then, it can be proved (beyond the scope of this paper) that the poset $(\mathcal{X}, \mathcal{P})$ has the maximum element, called here the *greatest antichain* of T. The polynomial-time algorithm (line 7) finding the greatest antichain of T is easy but long, so we do not describe it here.

The third property cannot be always fulfilled, since the maximum antichain of size n_{dims}, guaranteed to exist in $(T, \leq)$, does not necessarily exist in $(T', \leq)$, where $T' \subseteq T$ is a set of tests solved by certain solution s.

FINDTESTS iterates over all solutions from S_{Pareto} and returns the first encountered antichain with the three properties described above (line 9); if such antichain does not exist, it returns all tests from T. The solutions from S_{Pareto} are considered in the order of decreasing minimal dimension, which identifies the weakest element of solution (strategy). Thanks to that, the algorithm prefers the search direction that equally treats all the underlying dimensions, which is intended to protect COSA from over-specialization.

VI. EXPERIMENT

We conducted an experiment in which we compared COSA with two state-of-the-art Pareto-coevolutionary archives that are shortly described in the following sections

A. Iterated Pareto-Coevolutionary Archive (IPCA)

IPCA is a coevolutionary archive proposed in [13] and further investigated in [21]. It guarantees monotonic progress for solution concept of Pareto-Optimal Set. This, however, comes at a cost: its test archive may grow infinitely.

IPCA, maintains a set of tests and a set of solutions. A newly generated solution is accepted by the archive only if it is non-dominated with respect to the tests maintained in the archive. When a solution is being accepted, a newly generated test that is required to keep it non-dominated is also accepted to the archive. The solutions in archive that become dominated by newly accepted solutions are removed. For details of this procedure see [21].

B. Layered Pareto-Coevolutionary Archive (LAPCA)

LAPCA [22] maintains a set of *Pareto layers*. For a set of solutions and tests ($S = S_{arch} \cup S_{new}$, $T = T_{arch} \cup T_{new}$), the first Pareto layer consists of all non-dominated solutions. Each subsequent layer is obtained in a similar way, after removing the solutions from all previous layers. The solution archive consists of solutions of first l layers, where l is a parameter of the method. In theory, l makes it easy to trade-off the archive size (thus computational power) and reliability of the archive. However, generally it is unknown which value of l is right for a particular problem.

LAPCA maintains also archive T_{arch} of tests that separate the solutions stored by S_{arch}. For any two solutions $s_i, s_j \in S_{arch}$ in layers i and j respectively, where $|i - j| \leq 1$, if there exists a test $t \in T$ that orders s_i before s_j, then such a test must also be retained in T_{arch}. Tests are processed in any order, and the first one that orders a not-yet-ordered pair of solutions is selected to T_{arch}.

C. Compare-on-one game

We compare the discussed methods on COMPARE-ON-ONE, a variant of the abstract Numbers Game [2], proposed in [8] and widely used as a coevolutionary benchmark [22], [23], [8], [13], [24], [10], [21], [1], [25]. In this game, strategies are represented as vectors of non-negative real numbers of length d, which we call here the *a priori dimension* of the

[1]This can be further improved to $O(n^{5/2}/\sqrt{\log n})$ by a method introduced in [20].

game. The outcome of an interaction between solution s and test t is determined in two steps. First, t appoints the index m of vector elements to be compared: $m = \arg\max_i t[i]$, where $t[i]$ denotes the i-th element of vector t. Then,

$$G(s,t) \iff s[m] \geq t[m]$$

Figure 2 illustrates the principle beyond COMPARE-ON-ONE for $d = 2$. Solution s_1 solves only the tests from the shaded polygon.

Both solutions and tests are represented as individuals with a genotype of d genes, each corresponding to one objective of the game. But, obviously, this fact, known to the experimenter, is not known to the evolutionary algorithm.

Despite a straightforward formulation, COMPARE-ON-ONE is challenging because it has been designed to induce over-specialization: a co-evolving system of solutions and tests can easily focus on some (or even a single) underlying objectives, while ignoring the remaining ones. As soon as all the tests start appointing the same comparison index m, other objectives cease to matter, and it becomes difficult to discover their importance. To make steady progress on this problem, an algorithm has to carefully maintain the tests that support all underlying objectives from the very beginning of the run.

COMPARE-ON-ONE is obviously an artificial problem. Thanks to that, we can objectively and precisely measure the progress of coevolution, what is impossible in case of practical test-based problems and real games[2].

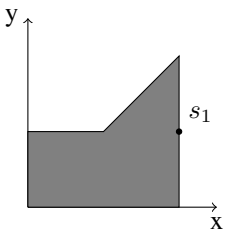


Figure 2: Visualization of COMPARE-ON-ONE for $d = 2$. s_1 solves all tests from the gray area.

D. Objective progress measures

To objectively monitor progress of algorithms on this problem, we employ two measures: lowest dimension and expected utility.

Lowest dimension was designed to detect over-specialization and is the standard progress measure for COMPARE-ON-ONE. For a given solution s, it is defined as $\max_i s[i]$, and determines the 'weakest point' of the solution.

Expected utility corresponds to the solution concept [4] of Maximization of Expected Utility (c.f. [24]). Expected utility of a solution s is formally the probability that s solves a test. However, in absence of upper limits on vector elements, strategies can contain arbitrarily large numbers, and such probability amounts to zero for all solutions. This is why we equal the expected utility with the hypervolume of the polyhedron that contains all the tests solved by s, visualized for COMPARE-ON-ONE for $d = 2$ in Fig. 2 and for $d = 3$ in Fig. 3. The hypervolume of such a polyhedron is given by:

$$U(s) = \frac{1}{d} \sum_{i=1\ldots d} s[i]^d$$

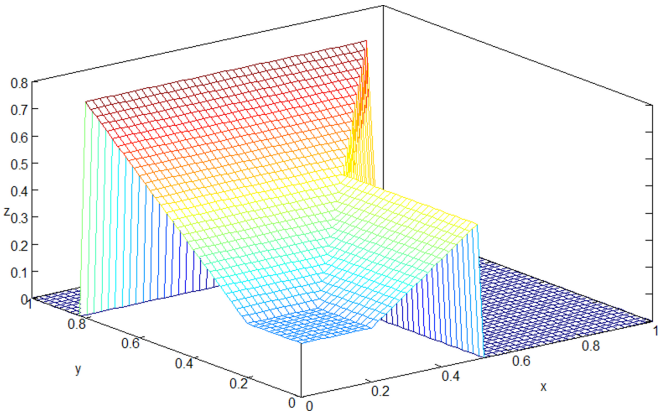


Figure 3: Visualization of COMPARE-ON-ONE for $d = 3$. A solution $s = [0.5, 0.8, 0.2]$ solves all tests bounded by the polyhedron.

Algorithm 3 Coevolutionary framework

```
1: procedure COEVOLUTION
2:     S, T ← Initialize populations
3:     S_arch ← ∅
4:     T_arch ← ∅
5:     while ¬stopped do
6:         S_new ← GenerateNewSolutions(S, S_arch)
7:         T_new ← GenerateNewTests(T, T_arch)
8:         S ← S ∪ S_new
9:         T ← T ∪ T_new
10:        Archive.Submit(S, T)        ▷ Updates S_arch and T_arch
11:        Evaluate(S, T)
12:        S, T ← Select(S, T)
13:    end while
14: end procedure
```

E. Experiment setup

Algorithm 3 details the overall coevolutionary framework that is common for all the archive methods discussed in this paper. The coevolutionary algorithm maintains n solutions and n tests (here, $n = 20$) and works as follows. At the beginning, both populations are initialized, with each gene drawn at random from the $[0, 1]$ range (line 2). Archives are initially empty (lines 3-4). Afterwards, COEVOLUTION iterates over consecutive generations in a way that resembles

[2]If it were possible, the objective measure could be used as fitness function for a standard evolution and no coevolution would be necessary.

an ordinary evolutionary algorithm. Each generation involves breeding of new solutions and tests using the generator (lines 6 and 7), evaluation (line 11), and selection (line 12). The following paragraphs detail these stages.

The exploration of the search space is driven by a *generator* which, in evolutionary terms, is responsible for providing genetic variation. Technically, the generator of solutions (GenerateNewSolutions in Alg. 3) and the generator of tests (GenerateNewTests) work similarly. Generator produces n individuals (offspring) in the following way. First, it randomly decides (with equal probability) whether the parent of the generated individual should come from the population or from the archive. Next, the generator mutates the parent and returns it as a new individual. The difference between generating solutions and generating tests lies in handling the archive. Generator of tests uses all tests from T_{arch} as potential parents, whereas the generator of solutions uses only the Pareto non-dominated solutions from S_{arch} as potential parents (this increases the pressure towards better solutions).

Mutation randomly perturbs two genes of an individual (solution or test). As in [21], the mutation is uniform in interval $[-0.2, 0.1]$, thus it is negatively biased: it is more likely to decrease a gene than to increase it. Such asymmetry causes the problem to be harder and bear more resemblance to real problems, where variation is more likely to cause regress than progress.

For a similar reason our generators refrain from crossing over the parent strategies: for this particular game of consideration and this particular strategy encoding, crossover could easily produce very good solutions and could alleviate to some extent the intentionally built-in tendency to focusing, rendering the game too easy[3].

The rationale behind mutating exactly *two* of vector elements is that such variation simulates epistatic interactions between the elements of the strategy or, in other words, a non-trivial genotype-phenotype mapping [22] – the offspring (generated strategy) differs from the parent on more than one dimension (skill). Mutating only one element would imply one-to-one correspondence between genes (strategy elements) and game skills, which would be simplistic and unrealistic from real-world perspective. Thus, modifying two elements is the minimal non-trivial perturbation. This and other aforementioned design choices make COMPARE-ON-ONE, despite its simplicity, a realistic and scalable approximation of practical test-based problems.

After updating S_{arch} and T_{arch} by an appropriate archive algorithm, COEVOLUTION proceeds to the evaluation phase, where each solution from S is given a fitness value computed as the number of tests it solves from T. Dually, the fitness of a test t from T equals to the number of solutions from S that does not solve t. This fitness determines the odds for an individual to pass the subsequent selection, which is implemented as tournament of size 2. Selection of solutions proceeds independently from the selection of tests.

COEVOLUTION quits when the total number of evaluations (games played) reaches $1,000,000$. Note that the actual number of elapsed generations can be, and usually is, different for particular algorithms and runs, because the number of evaluations depends directly on the size of the archive, which varies over time.

[3]Consider, for example, two solutions: [0,10] and [10,0] and a possible product of crossing-over them: [10,10]

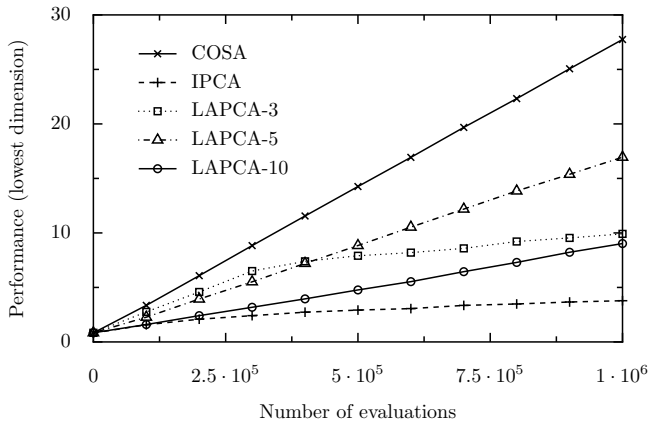
VII. Results and Discussion

Charts presented in Figs. 4 and 5 summarize the results of the conducted experiment for COMPARE-ON-ONE of dimensions $d = 2, 3$, and 5. LAPCA has been run thrice, for $l = 3, 5, 10$ layers, to give it a chance of attaining good performance. The left column of charts in Fig. 4 depicts the performance of the algorithm expressed by the lowest dimension of the best solution in the archive, and the right one – the expected utility of the best solution in the archive. In Fig. 5, all charts visualize the sizes of archives, left-hand charts for solutions, right-hand charts for tests. All charts present the averages of 10 runs.

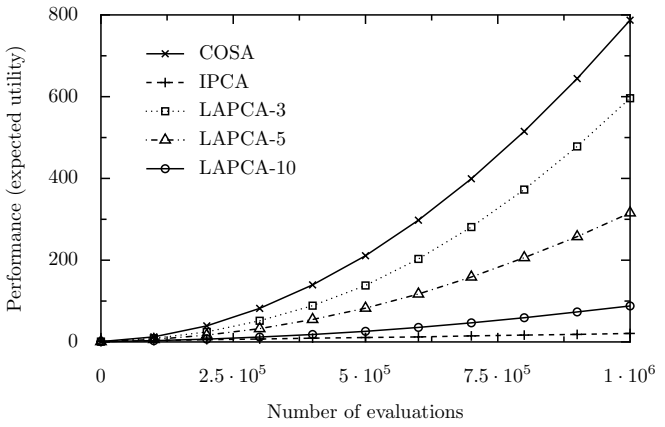
The superiority of COSA for this problem is evident and spans all considered problem instances and performance measures. In particular, COSA is able to make a steady progress in terms of the lowest dimension of the solutions. The other algorithms lag far behind, particularly for harder problems (larger d). LAPCA is quite good for $d = 2$. The best version of LAPCA seems to be the one with 5 layers: LAPCA-3 is initially better, but then slows down. For $d = 2$, LAPCA-10 is slow, but still faster than IPCA. The things change for $d = 3$, when IPCA gets better than all LAPCAs. For $d = 5$ no substantial progress in the lowest dimension is observed both for IPCA and LAPCA. All in all, COSA copes with over-specialization better than other considered methods.

The expected utility of the best evolved solution also votes in favor of COSA. Therefore, the probability of a test being solved by the best solution in the archive is highest for COSA. The progress on this criterion looks polynomially for all algorithms, because, roughly speaking, the hypervolume depends on the d^{th} power of the largest dimensions of a solution. For the same reason, the absolute values of this measure get much higher when increasing the problem dimension.

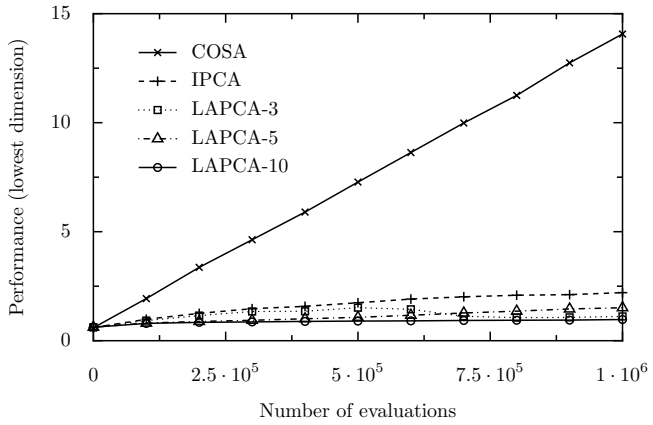
Also from the viewpoint of archive sizes, COSA is among the leaders. For solutions, it is neck and neck with IPCA, and together they beat LAPCA. For $d = 5$, COSA seems to noticeably subdue IPCA in the later phases of the search. Strikingly, for $d = 2$ and 3 the number of solutions stored in the archive is extremely low and does not increase over time. For test archives, the differences between methods are even more striking: this time, even IPCA is unable to keep pace with COSA, which is not surprising as IPCA never discards any older tests, while COSA stores only one test per dimension and a few additional tests to separate solutions in the Pareto layer. Our algorithm manages to maintain the lowest number of tests, which is usually only a small fraction of the archives of other methods. Most importantly, COSA's



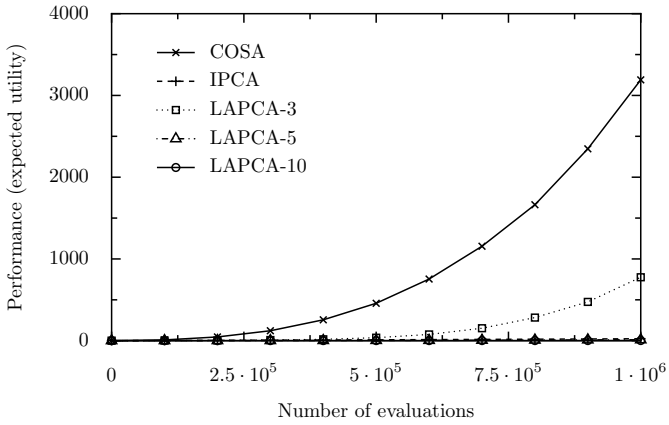
(a) $d = 2$, lowest dimension



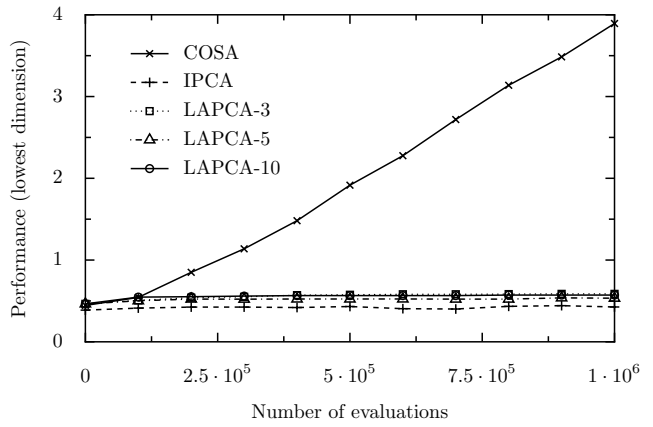
(b) $d = 2$, expected utility



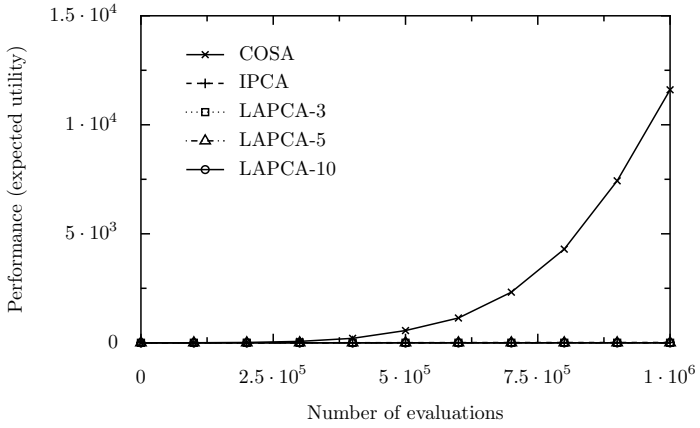
(c) $d = 3$, lowest dimension



(d) $d = 3$, expected utility



(e) $d = 5$, lowest dimension

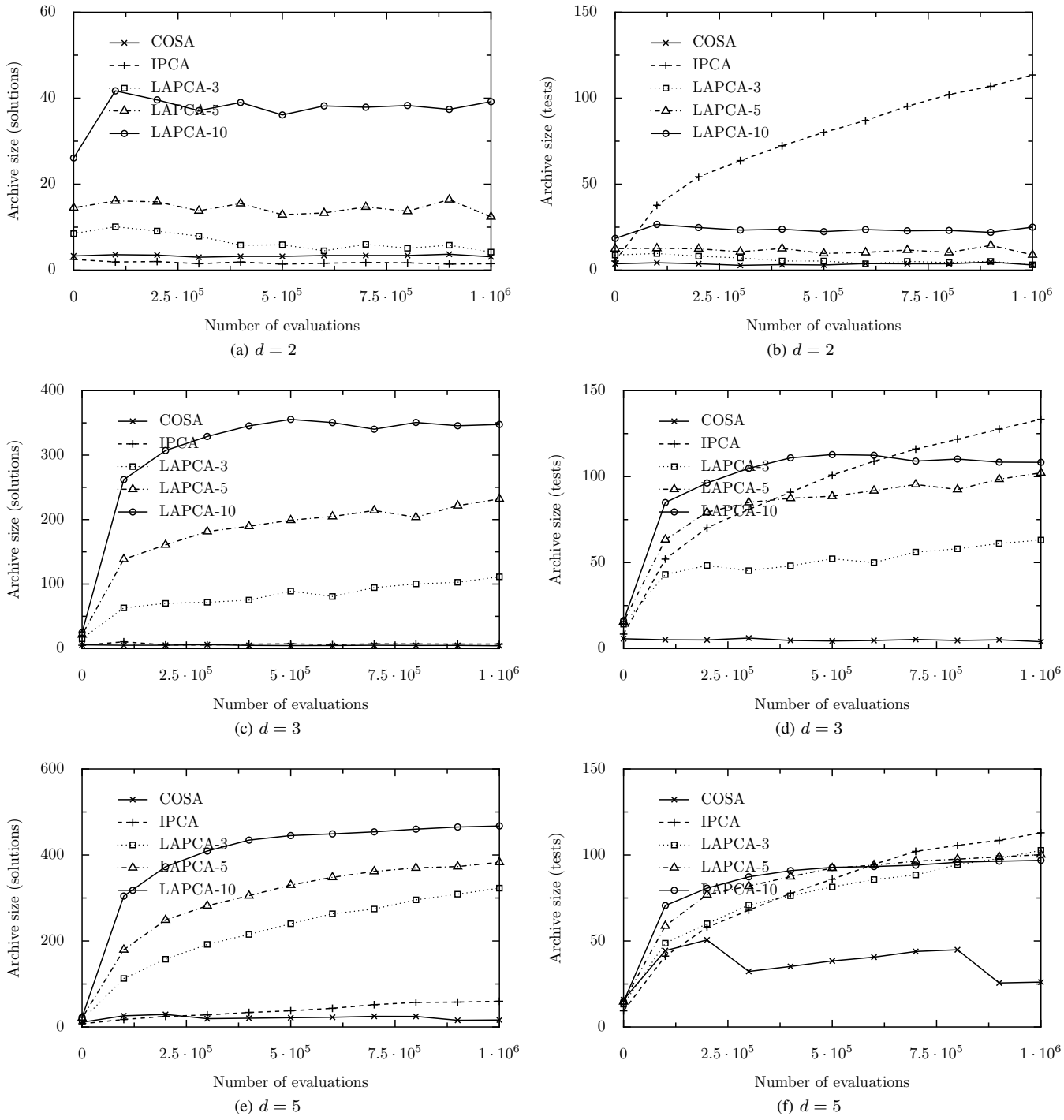


(f) $d = 5$, expected utility

Figure 4: Results for d-dimensional COMPARE-ON-ONE ($d = 2, 3, 5$) for two performance measures: expected utility and lowest dimension

dimension	COSA	IPCA	LAPCA-3	LAPCA-5	LAPCA-10
2	980.9±1.8	407.7±15.0	915.3±53.0	763.0±10.3	488.4±18.7
3	928.1±2.3	343.9±22.2	380.6±256.1	133.2±82.5	51.3±33.7
5	705.6±251.3	344.6±35.7	104.5±18.1	59.1±7.9	41.2±5.5

Table II: Average number of generations algorithms were able to perform during 1,000,000 evaluations



(a) $d = 2$ (b) $d = 2$

(c) $d = 3$ (d) $d = 3$

(e) $d = 5$ (f) $d = 5$

Figure 5: Archive sizes for d-dimensional COMPARE-ON-ONE ($d = 2, 3, 5$)

archive sizes remain virtually constant over time and there is no reason to doubt in further maintenance of this behavior, while other archives' sizes keep growing and do not seem to saturate, which at some stage can render them useless.

The fact that IPCA is better than LAPCA for $d = 3$ and that, in general, LAPCA performs so poorly is surprising, since it contradicts the findings of earlier research [21], where IPCA was found worse than LAPCA on a similar problem. The charts in Fig. 5 suggest that the presumed reason for that is an excessive growth of LAPCA's archive, because the Pareto layers contained many non-dominated solutions. This, in turn, could be caused by the absence of crossover in our setup ([21] used two-point crossover with 50% probability). However, this hypothesis deserves a separate investigation.

As mentioned earlier, the number of iterations executed by the coevolutionary loop (Alg. 3) depends on the sizes of archives. We illustrate this dependency in Table II, where we report the average number of generations that each algorithm went through. The presented numbers resonate with the charts: because COSA is able to maintain small and approximately constant-sized archives, its run length measured in generations is the highest and does not seem to be affected by the dimensionality of the problem d, which is not the case for LAPCA. This is critical, as the number of iterations is also the number of generator invocations, which are the only source of variability for the search.

VIII. Summary

In this paper we proposed Coordinate System Archive, a novel archive-based coevolutionary algorithm based on the concept of underlying problem structure and coordinate systems. We demonstrated that COSA performs better than two state-of-the-art algorithms on a class of problems, in terms of both avoiding over-specialization and expected utility of the produced solutions. Whether this class of COMPARE-ON-ONE-alike problems is representative to a large class of real-world problems or not is a question that we would like to answer in future.

Acknowledgments

This work was supported in part by Ministry of Science and Higher Education grant # N N519350533. W. Jaśkowski gratefully acknowledges financial support from grant # N N516188337.

References

[1] A. Bucci, J. B. Pollack, and E. de Jong, "Automated extraction of problem structure," in *Genetic and Evolutionary Computation – GECCO-2004, Part I*, ser. Lecture Notes in Computer Science, K. D. et al., Ed., vol. 3102. Seattle, WA, USA: Springer-Verlag, 26-30 Jun. 2004, pp. 501–512.

[2] R. A. Watson and J. B. Pollack, "Coevolutionary dynamics in a minimal substrate," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, L. S. et al., Ed. San Francisco, California, USA: Morgan Kaufmann, 7-11 Jul. 2001, pp. 702–709.

[3] J. P. Cartlidge, "Rules of Engagement: Competitive coevolutionary dynamics in computational systems," Ph.D. dissertation, University of Leeds, 2004.

[4] S. G. Ficici, "Solution concepts in coevolutionary algorithms," Ph.D. dissertation, Waltham, MA, USA, 2004, adviser-Pollack, Jordan B.

[5] S. G. Ficici and J. B. Pollack, "Pareto optimality in coevolutionary learning," in *Advances in Artificial Life, 6th European Conference, ECAL 2001*, ser. Lecture Notes in Computer Science, J. Kelemen and P. Sosík, Eds., vol. 2159. Prague, Czech Republic: Springer, 2001, pp. 316–325.

[6] J. Noble and R. A. Watson, "Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for pareto selection," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, L. S. et al., Ed. San Francisco, California, USA: Morgan Kaufmann, 7-11 Jul. 2001, pp. 493–500.

[7] E. de Jong and A. Bucci, "Objective Set Compression. Test-Based Problems and Multiobjective Optimization," in *Multiobjective Problem Solving from Nature: From Concepts to Applications*, J. K. et al., Ed. Berlin: Springer, 2008, pp. 357–376.

[8] E. D. de Jong and J. B. Pollack, "Ideal Evaluation from Coevolution," *Evolutionary Computation*, vol. 12, no. 2, pp. 159–192, Summer 2004.

[9] W. Jaśkowski and K. Krawiec, "Formal analysis and algorithms for extracting coordinate systems of games," in *IEEE Symposium on Computational Intelligence and Games*, Milano, Italy, 2009, pp. 201–208.

[10] E. D. de Jong and A. Bucci, "DECA: dimension extracting coevolutionary algorithm," in *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, M. C. et al., Ed. Seattle, Washington, USA: ACM Press, 2006, pp. 313–320.

[11] R. Dilworth, "A decomposition theorem for partially ordered sets," *Annals of Mathematics*, pp. 161–166, 1950.

[12] A. Bucci and J. B. Pollack, "A mathematical framework for the study of coevolution," in *Foundations of Genetic Algorithms 7*, K. A. De Jong, R. Poli, and J. E. Rowe, Eds. San Francisco: Morgan Kaufmann, 2003, pp. 221–236.

[13] E. D. de Jong, "The Incremental Pareto-Coevolution Archive," in *Genetic and Evolutionary Computation–GECCO 2004. Proceedings of the Genetic and Evolutionary Computation Conference. Part I*, K. D. et al., Ed. Seattle, Washington, USA: Springer-Verlag, Lecture Notes in Computer Science Vol. 3102, Jun. 2004, pp. 525–536.

[14] C. D. Rosin and R. K. Belew, "Methods for competitive co-evolution: Finding opponents worth beating," in *ICGA*, L. J. Eshelman, Ed. San Francisco, CA: Morgan Kaufmann, 1995, pp. 373–381.

[15] C. Rosin and R. Belew, "New methods for competitive coevolution," *Evolutionary Computation*, vol. 5, no. 1, pp. 1–29, 1997.

[16] D. S. Johnson, "Approximation algorithms for combinatorial problems," *Journal of Computer and System Sciences*, vol. 9, pp. 256–278, 1974.

[17] C. Lund and M. Yannakakis, "On the hardness of approximating minimization problems," *Journal of the ACM (JACM)*, vol. 41, no. 5, pp. 960–981, 1994.

[18] R. Möhring, "Algorithmic aspects of comparability graphs and interval graphs," *Graphs and Order: The Role of Graphs in the Theory of Ordered Sets and Its Applications*, pp. 41–102, 1984.

[19] S. Felsner, V. Raghavan, and J. Spinrad, "Recognition algorithms for orders of small width and graphs of small Dilworth number," *Order*, vol. 20, no. 4, pp. 351–364, 2003.

[20] H. Alt, N. Blum, K. Mehlhorn, and M. Paul, "Computing a maximum cardinality matching in a bipartite graph in time O n 1.5 m/log n," *Information Processing Letters*, vol. 37, no. 4, pp. 237–240, 1991.

[21] E. D. de Jong, "A Monotonic Archive for Pareto-Coevolution," *Evolutionary Computation*, vol. 15, no. 1, pp. 61–93, Spring 2007.

[22] E. De Jong, "Towards a bounded Pareto-Coevolution archive," in *Proceedings of the Congress on Evolutionary Computation CEC-04*, vol. 2. Portland, Oregon, USA: IEEE Service Center, Jun. 2004, pp. 2341–2348.

[23] E. De Jong and J. Pollack, "Learning the ideal evaluation function," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-03*. Berlin, 2003: Springer, 2003, pp. 274–285.

[24] E. de Jong, "The maxsolve algorithm for coevolution," in *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, H.-G. B. et al., Ed., vol. 1. Washington DC, USA: ACM Press, 25-29 Jun. 2005, pp. 483–489.

[25] T. C. Service and D. R. Tauritz, "Co-optimization algorithms," in *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2008, pp. 387–388.