Future and Emerging Technologies
FET-Open

Phoenix

665347

Co-evolutionary Schemes

Deliverable 4.3

Contents

This project is funded
by the European Union

1 Introduction

1.1 Phoenix concepts

Two main keywords for the Phoenix project's approach are co-evolution and reincarnation. The Phoenix process starts with a user's question about an unknown environment [1]. The answer to this question is then obtained via two loops as in Figure 1.

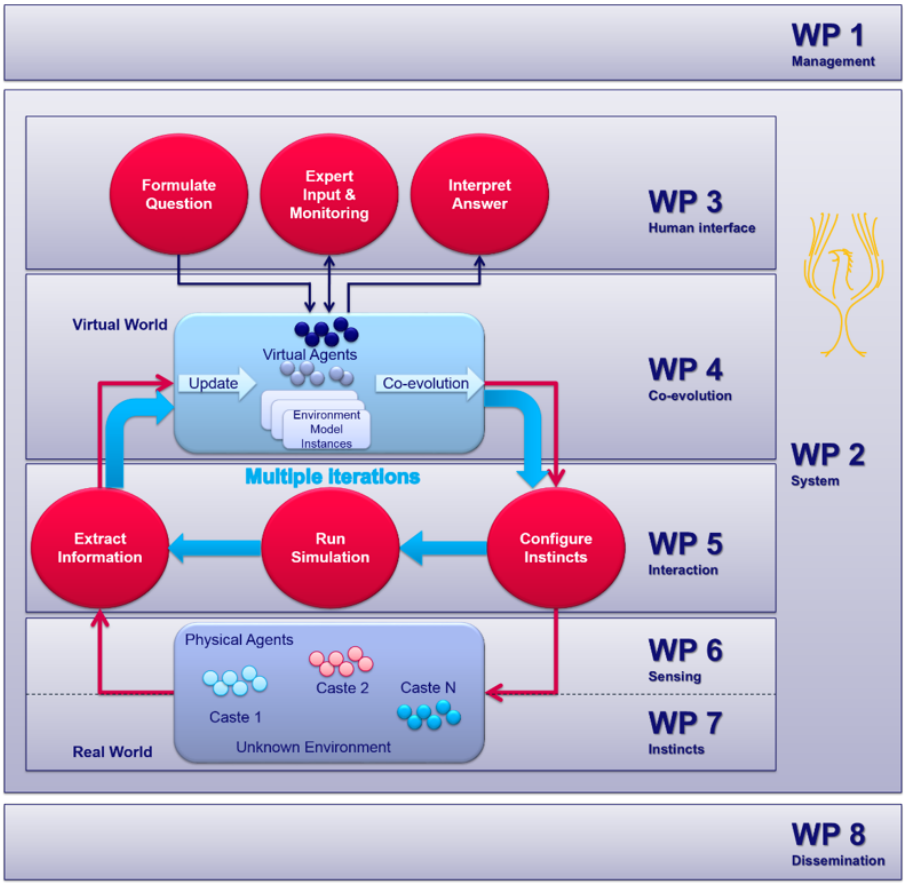

Figure 1: Phoenix Scheme

In the first *outer* loop (red cycle), extremely small autonomous physical agents explore the unknown environment. The first iteration of the loop is initiated based on a *best guess* model for the environment and the agent configurations derived from possibly little -and uncertain- knowledge elicited from experts, that is stored in a database [2, 3, 4].

Based on the data provided by the agents recovered from the environment, a virtual model of the environment is constructed, and further refined in the *inner* loop (blue cycle). During this nested loop, the virtual model (and possible multiple instances of it) is repeatedly explored by virtual representations of the physical agents using an Evolutionary Algorithm (EA), co-evolving both the virtual model and the virtual agents, helping to refine the next incarnation of physical agents.

These reincarnated agents will again explore the unknown environment to gather more data. In this scheme, the EA is used as the main machine learning algorithm to optimize the agents behavior and design, and to converge to the (closest-to) *real* model of the environment. Con-

sequently, its interaction with agents and environment models is a cornerstone in the project. In this context, the co-evolution of agents and environment is a crucial point. In the present deliverable, we focus on the EA designed to obtain such co-evolution, as briefly summarized in the next section.

1.2 About this deliverable

This purpose of the deliverable is twofold: on one hand, we describe the details of the proposed co-evolution scheme. Additionally, we present the preliminary results on a proof-of-concept case study used to validate the proposed co-evolutionary approach.

1.3 Deliverable structure

In Section 2, we provide the general description of the co-evolutionary scheme and its components. In Section 3, we give more insights regarding the solutions evolved by the EA, and we explain how the introduced solution concepts can affect the EA's stopping criteria. In addition, we introduce a case study to test the co-evolutionary scheme, where the proposed co-evolutionary scheme is applied on a simplified case study. Finally, we provide the conclusion in Section 5.

2 Co-evolution Scheme

One of the main reasons for using Evolutionary Algorithms (EAs) in this project is their general-purposeness, as well as their ability to handle optimization problems that are non-differentiable or even lack a mathematical formulation at all (for instance, Monte Carlo stochastic simulations, or the output of a pipeline of computing processes) [5]. Furthermore, the optimization problems underlying the Phoenix project (i.e., the optimization of environment models and agents) are characterized by a large solution space. Environment models include indeed a wide range of physical properties that are inter-related in a complex manner, e.g. temperature, pressure, density, etc., besides the geometrical parameters. On the other hand, the agents' parameters include their physical design, instincts (controllers) and extreme-energy constrained hardware.

Given all this, analyzing the solution space of the environment and agents to highlight the most interesting elements of the search space is a sound idea. However, this process has to be done in a co-evolutionary scheme, i.e. both environments and agents must evolve at the same time, within the same framework; otherwise, we would fail at identifying the relevant aspects of the solution space.

To analyze the solution space, we create an archive of *"interesting"* solutions (environments *and* agents). This archive includes agent solutions that can for example handle extreme environment cases, such that they can be reused during the evolutionary process to generate novel and possibly more efficient solutions for newly generated environment models. In addition, the archive contains a set of extreme environments as well, which can then be used in the environment comparisons while attempting to converge to the closest-to-reality environment.

In this part of the document, we introduce the co-evolutionary scheme, which is inspired from [6] as follows: firstly, we summarize the environment and agent representation in Section 2.1 and 2.2, respectively. Then, in order to have an adequate environment-agent co-evolution scheme, we need to define a common evolution ground, where they both interact (what we call a "mission"). This is detailed in Section 2.3. Section 2.4 and 2.5 introduce the environment and agent fitness. Finally, in Section 2.6 we provide the complete overview of the co-evolution scheme.

2.1 Environment Representation

In Deliverable 4.1 [7], we have already introduced the environment and agent representation used in the Phoenix project. However, here and in the next section we summarize (and update) the details of the two representations.

Before going into the details, it is important to highlight two crucial aspects of the environment representation:

1. Inaccurate environment representations may lead to fallacious agent evolution. Thus, in a co-evolutionary scheme, if the environment representation is not accurate, this would lead to an inefficient agent evolution as well, which, consequently, can lead to the generation of a useless archive of agents.
2. Inaccurate environment representations cause incorrect comparisons between newly estimated environments and old ones. This causes in turn the system to be either prematurely ending or indefinitely running.

Keeping these two observations in mind, we have devised an environment model representation (mapping) as follows. The input of the method is a collection of agent position estimates $\hat{\boldsymbol{p}}_{i,t} \in$

$\mathbb{R}^d$, for each agent i and each time step t of the exploration task. Such collection is henceforth given by:

$$\mathcal{V} = \{\hat{\boldsymbol{p}}_{i,t} \mid \forall i, \forall t\}. \tag{1}$$

Intuitively, the uncertainty of the environment features estimation in the denser regions (i.e., the regions where many points of $\mathcal{V}$ are clustered near each other) is lower than in less dense regions. In other words, we are more certain of the map features in the regions which were visited more frequently by the agents (thus, providing more information). This plays an important role in our presented mapping scheme.

Given these inputs, the environment mapping scheme must fulfill four main design objectives:

1. It should facilitate the ability to accumulate localization knowledge with each run.
2. It should be suitable for agents with limited resources, i.e. it should work with a limited number of sensors.
3. It should include a probabilistic measure of (un)certainty into the output map features.
4. It should be computationally inexpensive.

Vietoris-Rips (VR) complex is a technique widely used mapping for similar problems. Despite the fact that VR based mapping does not fulfill all previously identified design objectives, highlighting its definition and how it is used in mapping will play an important role in showing the points of strength of our presented mapping scheme.

One relevant definition to VR complex from graph theory is the one of a *clique*. A clique, $\mathscr{c}$, is defined in an undirected graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, as the subset of the vertices $\mathcal{V}$, such that every two distinct vertices are adjacent. For example, an edge in $\mathcal{G}$ is a clique with two vertices and a triangle in $\mathcal{G}$ is a clique with three vertices. This is useful in the forthcoming definitions. Given $\mathcal{V}$ (Eq. 1), the following clique complex set $\mathcal{C}$ can be obtained:

$$\mathcal{C}^{d,\epsilon} = \left\{ \mathscr{c} \subseteq \mathcal{V} \;\middle|\; \forall \boldsymbol{u}, \boldsymbol{v} \in \mathscr{c}, \boldsymbol{u} \neq \boldsymbol{v}, \|\boldsymbol{u} - \boldsymbol{v}\| \leq \epsilon \right\}, \tag{2}$$

where all points have a pairwise distance of at most ϵ. A simplex σ corresponding to each clique $\mathscr{c}$ in $\mathcal{C}^{d,\epsilon}$ is defined as the convex hull of the points constituting the clique. A simplex is called k-simplex if $|\sigma| = \kappa(\sigma) = k + 1$ where k is the cardinality of the simplex σ, which is equivalent to the cardinality of its corresponding $\mathscr{c}$, i.e. the number of vertices constituting the clique.

A VR complex with maximal dimension k is then defined as the collection of all simplices with cardinality at most k. Note that this includes full sub-simplices $\sigma' \subset \sigma$ as well as partial sub-simplices like $\sigma' = \{\sigma_1, \sigma_2, \ldots\}, \sigma_1 \subset \sigma, \sigma_2 \nsubseteq \sigma$. In Fig. 2, an illustrative example is shown: given four points, with Euclidean distance less that ϵ between them, simplices of 4-vertex (the entire graph) and 3-vertex cliques are highlighted.

From these definitions, it is now more clear why would VR complex be used in mapping: in Phoenix it can be used to show the positions (zones) of the map, where there are more dense data collected from the agents over repeated experiments, i.e. zones frequently visited by more agents, and where agents were very close to one another. This concept can then be used to present our (un)certainty for different zones in an environment [8]. In fact, the cardinality of the simplex can express such certainty, e.g. our certainty about the knowledge of the zones located in a simplex with 3 edges is less than the one with 4 edges. An algorithm describing the mapping process via VR complex is illustrated in Algorithm 1.

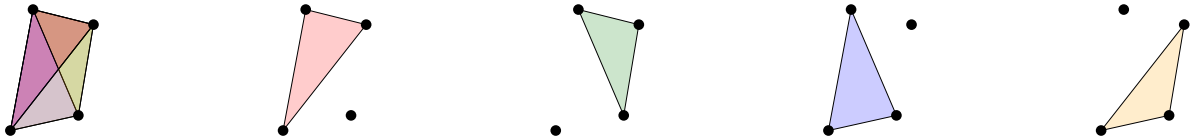

Figure 2: VR complex of four points (a), and its four sub-simplices of cardinality two (b)-(e).

Algorithm 1: VR Complex based Environment Mapping

Input: Position estimates $\hat{\boldsymbol{p}}_{i,t}$
Input: Maximum neighbor distance ϵ
Input: Maximal simplex order k
Output: Map estimate $\mathscr{M} = (\mathcal{R}, w), \mathcal{R} \subseteq \mathbb{R}^d, w{:}\mathbb{R}^d \to \mathbb{R}$
1 Determine set of all cliques $\mathcal{C}^{d,\epsilon}$ according to Eq. (2); Determine set of simplices and their cardinality $(\mathcal{S}, \mathcal{K}) \leftarrow \{(\mathcal{H}(\mathscr{c}), \kappa(\sigma)) \mid \forall \mathscr{c} \in \mathcal{C}^{d,\epsilon}, \kappa(\mathscr{c}) \leq k+1\}$;
2 Obtain map by plotting all simplices, starting in ascending order of the cardinality and assign the area of each simplex the weight $w(\mathcal{H}(\mathscr{c})) = \kappa(\sigma)^{\dagger}$;

†: In the map, (partial) sub-simplices may be covered by other simplices with higher cardinality (weight). Due to the processing in ascending order, only the highest weight of a region (hypervolume) contributes to the weight function w.

From what has been presented so far, it is clear that the VR complex is just set of simplices put together. Moreover, the cardinality of each simplex reflects the density of points in $\mathcal{V}$, thus it can be used as a proxy for (un)certainty as we elaborated earlier. However, this has a drawback: the weights of each of these cliques is expressed only by the number of its edges, and nothing more. For example, two cliques are considered equivalent, if the same number of edges of both cliques are smaller than ϵ. This might not reflect our needs, because one clique might have way much smaller edges than another, but because both have the same number of edges, that is less than ϵ, they are considered equivalent.

To overcome this drawback, we have proposed a new weighted VR-mapping scheme, as already described in Deliverable 5.1 [9]. The weights-driven VR Rips complex approach facilitates the creation of certainty distributions over the map, by introducing a weight function w as follows:

$$w_{\mathrm{MC}}(\mathscr{c}) = \frac{1}{2} \sum_{\substack{(\boldsymbol{u},\boldsymbol{v}) \in \mathscr{c} \times \mathscr{c}, \\ \boldsymbol{u} \neq \boldsymbol{v}}} \|\boldsymbol{u} - \boldsymbol{v}\|^{-1}, \tag{3}$$

This aims to give higher weights to sets of agent coordinates which are denser: for example, in the Phoenix context, the zones where there is a higher probability of existence of fluid in an environment, or where there is certainty that there is no fluid at all. In other words, the map consists of a collection of weighted hulls which represent the mapping certainty. Consequently, the weighted VR-complex mapping technique is adopted as the main method used for environment representation in Phoenix. In the next section, we shed the light of agent representation.

2.2 Agent Representation

Generally an agent is characterized by its morphology, instincts and hardware. The morphology has an impact on its kinetics, thus its trajectory in the environment, which in turn has an effect on the exploration capabilities, among other important factors such as energy consumption. The

instincts, on the other hand, are the evolved controller, which involves also learning. Finally, hardware includes the electronics and sensors available on the agents. In Phoenix, we attempt to unite all these factors under a single evolutionary scheme. In this section we mostly focus on the instinct part, which is the only one for which as at this stage of the project preliminary evolutionary experiments have been carried out. However, the agent representation and the co-evolutionary framework can be easily extended to include also the other elements characterizing the agents.

We represent the instincts in terms of behavioral trees (BTs), a technique that was originally introduced in the gaming industry mainly to offer a modular, flexible and readable representation scheme of agents. More recently, BTs have become widely popular also in robotics and unmanned air vehicles.

Formally, a BT is depth-first acyclic directed graph. Any node in this tree is either a *parent*, i.e. connected to lower level nodes dubbed as *child* nodes, or a *leaf* node, which has no child nodes. Moreover, each BT has one unique parent node named *root*, which can not have a parent and has only a single child.

Each node can have one of three possible states: Success, failure or running. Furthermore, all nodes, except the root node, fall under two main categories: control flow nodes, or execution (leaf) nodes. Control flow nodes are further sub-categorized into composites nodes and decorative nodes, each one including a wide range of possibilities, such as selector and parallel nodes. On the other hand, the execution (leaf) nodes have only two possibilities: action or condition nodes. The most commonly used nodes in each category and their functionality can be described as follows:

- **Leaf Nodes**
  - **Action node**: Returns *running* when the given action is still running, *success* when it is done, and *failure* otherwise.
  - **Condition node**: Returns *success* when the condition is fulfilled and *failure* otherwise.
- **Composite Nodes**
  - **Selector node**: Sequentially ticks its children nodes, starting from the out-most left node, and returns the state of the first non-failing child, i.e. either *success* or *running*. Otherwise, it returns *failure*.
  - **Sequence node**: Sequentially ticks its children nodes, starting from the out-most left node, and returns the state of the first non-succeeding child, i.e. either *failure* or *running*. Otherwise, it returns *success*.
  - **Parallel node**: Sequentially ticks its children nodes, starting from the out-most left node; if the number of succeeding children exceeds a defined node parameter limit S, it returns *success*, while if the number of failing children exceeds a defined node parameter limit F, it returns *failure*. Otherwise it returns *running*.

As for the execution process of BTs, it starts from the root *ticking*, which consequently starts ticking its children signaling the permission to start their functional properties, and, consequently, return their respective state to the parent that ticked them, and so on recursively on the tree. Figure 3 shows a BT example including various kinds of nodes.

Given this description, it important to highlight why BT is an appropriate scheme for representing instincts. Firstly, adding and removing nodes can be easily done without any changes

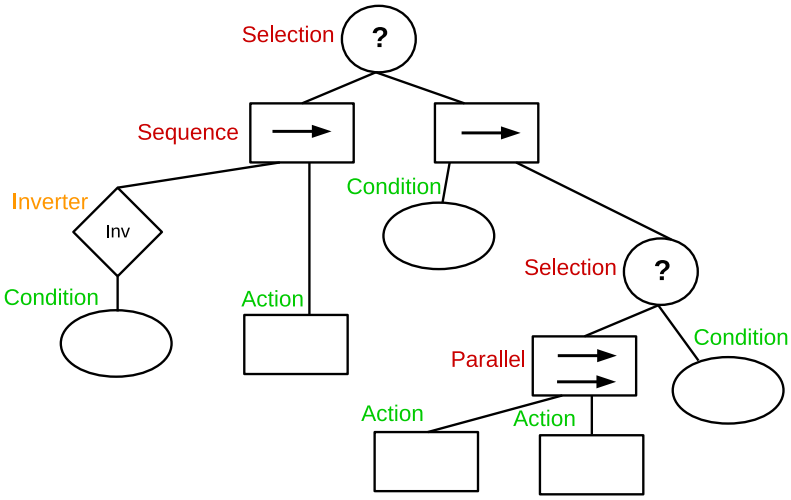

Figure 3: BT: an example

in either their parent nodes or child nodes. Furthermore, since a BT is an acyclic graph with a root node, it facilitates hierarchical expansion. Moreover, the availability of dedicated nodes for actions and conditions gives the user a certain degree of design flexibly. And since BTs are a tree by structure, their complexity in representation can be a quantitatively measured. Finally, another important aspect is handling parallelism: with the ticking method, all deadlock avoidance algorithms are not needed, thus minimizing the complexity, which is reflected positively on the need of hardware resources.

2.3 Mission Definition

Classic EAs are based on the concept of one or more *fitness functions* to be optimized, i.e. functions in the form of $f : G \to R$ that assign a real value R to each possible genotype in G [10]. Given such a function, the fitness relationship between any two genotypes $g_1, g_2 \in G$ is clear: compare $f(g_1)$ with $f(g_2)$ to see which is more fit (in the case of single-objective optimization; in multiple-objective optimization, the comparison is performed in terms of Pareto-optimality). By contrast, co-evolutionary schemes do not use such a direct metric of the fitness of individuals. Instead, two individuals are compared on the basis of their outcomes from interactions with other individuals. In our co-evolutionary scheme, we refer to this interaction as a *mission*. In other words, a mission defines the interaction between the agent population and environment population.

The objective here is to define a mission that will help us explore the solution space effectively, producing a set of environments and agents that if added to the population of the virtual loop would lead to fast and efficient convergence towards optimality.

It is important to highlight that a mission can consist of multiple *tasks*. For instance, one mission example could be requiring that an agent passes through different environment zones, where each zone has a certain 'challenge', i.e. the agent should perform an appropriate action to collect the needed sensor data or adapt some behavior parameters. In this case, the mission is a success when the agent passes though all zones conducting the proper action relative to the challenge introduced by each zone. In this regard, the link between the mission state (success or failure)

and state of its constituting tasks can be generalized as follows:

$$\text{Mission State} = f(T_1, T_2, \ldots, T_N) \tag{4}$$

here T_i is the state of task i in the mission. Depending on how one chooses the composition function $f(\cdot)$ and how T_i are defined, the overall fitness of the agents undergoing a given mission, and the score they have for each task, can be formulated with different ways. These possibilities are dubbed as a *solution concept*, as detailed in Section 3.

2.4 Environment Fitness

Given a mission, we can now define the environment fitness function as follows:

$$F_E = \frac{1-\text{Mission State}}{\text{Environment Coefficient}} \tag{5}$$

where "Mission State" has a value of 1 (success) or 0 (failure), and describes if the agent succeeded in the mission. It is represented in the form of function as in Eq. 4. Furthermore, given an environment representation, the value of the "Environment Coefficient" is the certainty of having such a configuration of that environment. In other words, this coefficient is supposed to ensure the likelihood of the environment representation.

2.5 Agent Fitness

A typical agent evaluation in our scheme can be described as follows:

$$F_A = \frac{\text{Mission State}}{\text{Action Cost}} \tag{6}$$

where "Mission State" is again a value of 1 or 0, describing if the agent succeeded in the mission (1) or failed (0) as specified in Eq. 4. Furthermore, given an instinct description, this can be defined as the energy consumed by it, i.e. the "Action Cost" required by that instinct.

2.6 Framework Overview

In Algorithm 2, we provide a pseudocode of the complete framework. The process starts with setting the mission, initializing an archive for the environments, an archive for the agents and the two coevolving populations. Afterwards, the algorithm starts a while-loop with a stopping criterion that can be (for instance) based on a maximum number of generations G_{max}. In this while-loop both the environments and agents populations are evaluated using the defined mission. If the fitness of any individual in the environment population is better that any individual in the environment archive, it replaces it. This same procedure is repeated for the agents as well. Afterwards, both environment and agent populations undergoes selection and reproduction procedures. The algorithm then returns the archives A_A and A_E for agents and environment, respectively.

Algorithm 2: High-level description of the Co-evolution framework

initialize max no. of generations G_{max};
initialize generation counter $g = 0$;
set initial mission M;
set initial archive for agents A_A of size N;
set initial archive for environments A_E of size N;
initialize environment population P_E;
initialize environment population P_A;
while $g < G_{max}$ **do**
    $F_A \leftarrow$ evaluate (P_E, P_A, M);
    $F_E \leftarrow$ evaluate (P_E, P_A, M);
    while $i < N$ **do**
        if $(F_{E_i} > A_{E_N})$ **then**
            $A_{E_N} \leftarrow$ Add (P_{E_i}, F_{E_i});
            $A_{E_N} \leftarrow$ Arrange A_E;
        end
        if $(F_{A_i} > A_{A_N})$ **then**
            $A_{A_N} \leftarrow$ Add (P_{A_i}, F_{A_i});
            $A_{A_N} \leftarrow$ Arrange A_A;
        end
        $i = i + 1$;
    end
    $P_A \leftarrow$ select (P_A, F_A);
    $P_A \leftarrow$ reproduce (P_A, F_A);
    $P_E \leftarrow$ select (P_E, F_E);
    $P_E \leftarrow$ reproduce (P_E, F_E);
    $g = g + 1$;
end

3 Solution Concepts

In this section we will shed the light on several solution concepts, which have been proposed in [6] and [10]. It is important to note that these possible solution concepts are connected to potential stopping criteria, because a solution concept specifies a subset of the potential solutions that is worth being saved in the archive, and one may specify a stopping criteria based on the size of the archive, or on some required properties (i.e. minimum fitness, average fitness, genotypic or phenotypic diversity, etc.) of the elements in the archive.

3.1 Simultaneous Maximization of All Outcomes

Simultaneous Maximization of All Outcomes (SMAO) requires a solution to maximizes its outcome over all possible tasks within a mission simultaneously. Due to its straightforwardness, this is considered the default solution concept in our co-evolutionary scheme.

3.2 Best Worst Case

Best Worst Case (BWC) is used to specify solutions that maximize the minimum possible outcome over interactions with all tasks in a mission. Thus, a solution's fitness is its worst performance measure against the fittest task in the mission. Generally, BWC is used in real-world problems [10], to protect against the worst scenario possible.

3.3 Maximization of Expected Utility

The Maximization of Expected Utility (MAU) concept is applied by looking for solutions which maximizes the expected score relative to a random case, i.e. a solution's fitness reflects its performance against all tasks (on average). Consequently, MAU looks at tasks as if they are all of equal importance.

3.4 Nash Equilibrium

Inspired from game theory, the Nash Equilibrium (NE) concept has a very interesting feature: at NE equilibrium, a "certainty" guarantee is associated with it. I.e., the expected payoff in a Nash equilibrium can not be lower than a certain value, regardless of the strategy with which the solutions interact with each other [10].

3.5 Pareto Optimal Set

Multi-Objective Optimization extends aims at solving optimization problems with multiple - typically contrasting- objectives. This may be viewed as the use of a fitness function that is vector-valued, rather than scalar. In the Pareto Optimal Set solution concept, every possible task is viewed as a separate objective to be optimized. This allows for an understanding of the different trade offs between possible Agent/Environment solutions and tasks within a mission.

4 Case Study

So far, we have introduced the main components constituting the proposed co-evolutionary scheme. In this section we introduce a case study as proof-of-concept of this scheme. We will project in this case study all the previously presented components: environment representation and fitness, agent representation and fitness, and finally a mission. The purpose is to give the reader an example of how such scheme would work.

4.1 Environment Representation

For the sake of simplifications, the environment in the case study is represented as a graph with n nodes, instead of a weighted VR complex. Each node can be regarded as a specific zone introducing a different "challenge" for an agent. We assume that every edge is unit-length, i.e. the environment edge length is not evolvable.

Each node can take one of H possible challenges, which mimics in real problems a hard communication zone, for example. For the environment topology, in the case-study we start with a simple topology to exhaustively search through all possible scenarios for analysis purposes. In the future steps of the project we plan to scale up to larger and more realistic typologies.

4.2 Environment Fitness

As discussed in Section 2.4), the general format of the environment fitness is defined by Eq. 5. However, in this specific case study we slightly modify the general definition by introducing the following formulation:

$$F_E = \frac{1-\text{Mission State}}{\text{Likelihood of environment}} \tag{7}$$

The main difference between Eqs. 5 and 7 is that we use as "Environment Coefficient" the likelihood of two adjacent challenges. For example, a very high pressure zone followed by a very low pressure zone is less likely than a high pressure zone followed by a low temperature zone. To achieve this, each element in the set of challenges H is mapped to a likelihood score for each other challenge.

4.3 Mission

In the case study, we define a mission as a sequence of tasks where each task specifies a start node (zone) and an end node (zone). A mission is successful if every task is completed, and fails if any of the tasks of the mission fails. An agent completes a task if it is running the adequate action at each zone, i.e. for each challenge (on each node) presented by the environment, the agent runs the correct action.

4.4 Agent Representation

For simplicity, an agent is represented as a set of actions running in series. Each action is suitable for a particular challenge. The agent goal is to conduct the optimal task for each environment node (zone) in the mission. In other words, each agent is basically a vector of sequential actions. However, in order to reflect reality, each of these actions is associated with a cost. In a real scenario, this can be the energy consumed to run that action or its realization cost.

For example, an agent can be represented as $a1 \rightarrow a3 \rightarrow a4 \rightarrow a2$: for running such sequence, the cost is simply the summation of the costs associated with each action. This agent successfully finishes a mission if the challenges presented in the environment are solved with that exact sequence; on the other hand, if the agent fails in a singe node (zone), the mission is considered a failure.

4.5 Agent Fitness

As seen in Section 2.5, we define the agent fitness function as follows:

$$F_A = \frac{\text{Mission State}}{\text{Action Cost}} \tag{8}$$

In this case there is no difference w.r.t. the general fitness function introduced earlier in Section 2.5. Thus, the fitness is equal to zero if the mission is a failure, and (1/Action Cost) if it is a success. As for the "Action Cost", each possible action is mapped to a fixed cost.

4.6 Results

In the case study, we used a population of 30 individuals for both environments and agents. Table 1 shows the produced archive of the agents and environments, obtained using the SMAO solution concept. The archive size was set to five solutions. We can see that the captured solutions reached a highest score of 93.1% and 80.3% of the maximum possible value of fitness for the agents and environments, respectively.

Table 1: Archive of agents and environments, with mean ± variance fitness over 30 runs, obtained using SMAO. The fitness is expressed as a percentage of the maximum possible value of fitness, as defined in Eqs.7-8. It should be noted that in each element of the archive the list of agents and environments are not sorted, such that there is no direct coupling between the entities, but only the best elements are stored.

Solution ID	Agent Fitness [%]	Environment Fitness [%]
1	93.1 ± 5.2	80.3 ± 2.2
2	89.3 ± 7.3	76.2 ± 1.2
3	84.2 ± 6.1	66.3 ± 2.2
4	77.3 ± 7.1	64.3 ± 5.2
5	59.3 ± 3.3	50.6 ± 2.5

5 Conclusion

In this deliverable we have introduced the co-evolutionary scheme used in the Phoenix project to generate an archive of environment and agent solutions, which helps the virtual loop to converge faster and more effectively. The scheme produces an archive of "extreme" (efficient) agents and "extreme" (challenging) environments. In the document, we presented the main elements of the co-evolutionary scheme, namely: the agent and environment representations, their related fitness functions, the link between agents and environment (defined the concept of "mission"), and different "solution concepts", i.e. different ways in which the fitness functions can be composed to reflect different user application goals and define different stopping criteria.

Furthermore, as a proof of concept, we presented a case study where all the components of the scheme were set, and showed the results with one of the proposed solution concepts. For future steps, we plan to conduct further analysis of different solution concepts, and generalize the case study to a complete Phoenix problem, also including hardware in the loop.

References

[1] Phoenix Project, *Deliverable 2.1 – System architecture and interface*, 2016.

[2] ——, *Deliverable 3.1 – Knowledge elicitation techniques analysis*, 2016.

[3] ——, *Deliverable 3.4 – Translation tool to/from human-friendly representation and formalization*, 2017.

[4] ——, *Deliverable 3.5 – Parser to extract information from formalized knowledge and interpret phoenix output*, 2017.

[5] A. Hallawa, A. Yaman, G. Iacca, and G. Ascheid, "A Framework for Knowledge Integrated Evolutionary Algorithms," in *Applications of Evolutionary Computation*, Springer, 2017.

[6] D. Garcia, A. E. Lugo, E. Hemberg, and U.-M. O'Reilly, "Investigating coevolutionary archive based genetic algorithms on cyber defense networks," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ACM, 2017.

[7] Phoenix Project, *Deliverable 4.1 – Agent and Environment Abstraction*, 2016.

[8] A. Hallawa, S. Schlupkothen, G. Iacca, and G. Ascheid, "Energy-efficient environment mapping via evolutionary algorithm optimized multi-agent localization," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ACM, 2017.

[9] Phoenix Project, *Deliverable 5.1 – Definition of the agents' instincts and related algorithms*, 2016.

[10] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *International Conference on Parallel Problem Solving from Nature*, Springer, 1994.

[11] A. Yaman, A. Hallawa, M. Coler, and G. Iacca, "Presenting the ECO: Evolutionary Computation Ontology," in *Applications of Evolutionary Computation*, Springer, 2017.

[12] A. H. Stephan Schlupkothen and G. Ascheid, "Evolutionary Algorithm Optimized Centralized Offline Localization and Mapping," in *International Conference on Computing, Networking and Communications: Wireless Ad hoc and Sensor Networks*, IEEE, 2018.