On the Effects of Collaborators Selection and Aggregation in Cooperative Coevolution: an Experimental Analysis

Giorgia Nadizar[1][0000−0002−3535−9748] and Eric Medvet[2][0000−0001−5652−2113]

[1] Department of Mathematics and Geosciences, University of Trieste, Italy
[2] Department of Engineering and Architecture, University of Trieste, Italy

Abstract. Cooperative Coevolution is a way to solve complex optimization problems by dividing them in smaller, simpler sub-problems. Those sub-problems are then tackled concurrently by evolving one population of solutions—actually, *components* of a larger solution—for each of them. However, components cannot be evaluated in isolation: in the common case of two concurrently evolving populations, each solution of one population must be coupled with another solution of the other population (the *collaborator*) in order to compute the fitness of the pair. Previous studies have already shown that the way collaborators are chosen and, if more than one is chosen, the way the resulting fitness measures are aggregated, play a key role in determining the success of coevolution. In this paper we perform an experimental analysis aimed at shedding new light on the effects of collaborators selection and aggregation. We first propose a general scheme for cooperative coevolution of two populations that allows to (a) use different EAs and solution representations on the two sub-problems and to (b) set different collaborators selection and aggregation strategies. Second, we instantiate this general scheme in a few variants and apply it to four optimization problems with different degrees of separability: two toy problems and two real prediction problems tackled with different kinds of model (symbolic regression and neural networks). We analyze the outcomes in terms of (a) effectiveness and efficiency of the optimization and (b) complexity and generalization power of the solutions. We find that the degree to which selection and aggregation schemes differ strongly depends on the interaction between the components of the solution.

Keywords: Cooperative coevolution · Collaborator selection · Fitness aggregation · Symbolic Regression · Neuroevolution · Genetic Programming · Evolutionary Strategies

1 Introduction and related works

Cooperative coevolutionary algorithms (CCEAs) are optimization techniques leveraging a divide-and-conquer approach for addressing complex problems where the solution can be decomposed into simpler *components* [16, 17]. The main idea

behind Cooperative Coevolution (CC) is that of evolving, i.e., optimizing, components independently, and than proceeding to aggregate them into the final, complex, solution to the problem. The rationale being that it might be easier to navigate smaller search spaces—those of components—separately, rather than the larger search space of complete solutions, making it simpler to converge to satisfying solutions.

The components identification and/or emergence is itself an interesting research topic, where continuous coadaptation plays a fundamental role [16]. Here, without loss of generality, we consider problems where the solutions have only two hand-designed components. In this case, evolution happens in an ecosystem with 2 species, which, as in the biosphere, are independent, since they are, in principle, related to completely different aspects of the problem. Therefore, each species is evolved in its own population, and the two evolutions are independent from each other.

However, in most practical cases, the components of the solution are completely meaningless on their own, making their independent fitness evaluation practically unfeasible. In fact, oftentimes the components acquire a meaning only when combined to make a complete solution to the problem at hand. Hence, the need for forming *collaborations* to enable the fitness evaluation. To make a clarifying example consider the scenario of a robot being optimized to perform a task, where CCEAs could be used to separately evolve the "body" and the "brain" of an artificial agent: it is immediately clear how neither a body without a brain, nor a brain without a body, can be assessed against the proposed task.

The process of forming collaborations introduces additional degrees of freedom: (1) on how to select the collaborators from the opposite component population (how many and which ones), and (2) on how to define the fitness of a component which has been evaluated in combination with different collaborators, i.e., how to *aggregate* multiple fitness values. Several approaches have been proposed in the literature for both: Ma et al. [4] provide a neat categorization of the existing ones (we refer the reader to the cited survey for further details). Collaborators selection methods mainly differ in terms of greediness: ranging from best collaborator selection [16], the greediest, to random or worst collaborator selection [13, 21], which maintain higher diversity in the population [10, 18]. Such methods can also be generalized or combined in a hybrid manner [14], in order to sample multiple collaborators. For the fitness aggregation schemes, the rationale is similar: the approaches range from the most optimistic ones, considering the fitness of the best individual [5], to pessimistic ones, using the worst fitness found [21], going through more moderate approaches, employing the average or the median fitness in the sample [7].

Clearly, the choice of the collaborator selection and the fitness aggregation have strong implications on the outcome of CC, affecting efficiency, computational costs, generalization ability, to only mention a few. However, the recipe for choosing the right scheme for a given problem remains somewhat unclear. Some studies have indeed tried to tackle this issue, as [3, 12, 15, 21], which have all proposed experimental comparisons of some existing methods. Yet, all these

studies focused on synthetic problems, which generally have different characteristics from the real-world ones. Moreover, to the best of our knowledge, all the existing works have only considered a particular combination of EAs, lacking the generality which CC enables in that direction.

In this context, our goal is that of filling the gap, by first formalizing a general and fully modular scheme of CCEA, and by then employing it for an experimental evaluation. We tailor the CCEA scheme to be completely agnostic with respect to the EAs chosen for evolving the two sub-populations, as long as they follow an iterative structure. Moreover, we devise it to be able to employ a variety of collaborator selection strategies, in combination with different fitness aggregation criteria. Concerning the experimental analysis, we aim at (a) providing a proof of concept of the generality of the formalized schema, and at (b) exploiting it for measuring the effects of different combinations in the modules (EAs, collaborator selection, and fitness aggregation). To this extent, we focus on two toy problems first, and then on two real-world problems: symbolic regression and neuroevolution for data classification.

Our results suggest the following insights. First, the number of collaborators has a clear impact on efficiency and a fuzzier impact on effectiveness of the optimization: in general, few collaborators are enough for co-evolving effective solutions without affecting efficiency too severely. Second, optimistic aggregation of fitness values, i.e., assigning to a collaborator the fintess of the best collaboration pair, is often better than other schemes (namely, median and worst). Third, depending on the interaction of solution components, fitness aggregation may have a strong impact on the structure of evolved solutions, by relying more on one of the two populations for exploring the conjoint search space. Finally, we do not observe any evidence of increased generalization power by using more collaborators or using unfit collaborators.

2 A general scheme for CC

We here describe a CCEA that instantiates a general scheme for CC. We consider the case with only two solution components and we assume that the way they are assembled together, in order to form a complete solution for the problem at hand, is fixed in advance. We deem our results conceptually portable to cases with more components, where emergence is allowed [16].

Our CCEA works by internally exploiting two other EAs. The only requirements we impose on the two EAs is that they are (a) iterative and (b) population-based. We assume that they comply to the structure displayed in Figure 1, according to which an EA corresponds to a function `solve(`f`)` that, given a fitness function $f : P \to \mathbb{R}$ to be minimized[3], should return a solution[3] $p^\star$, such that $q^\star = f(p^\star)$ is minimal. Note that a given EA might internally search a space G different than P and use a genotype-phenotype mapping function $\phi : G \to P$

[3] Our CCEA does not constrain the fitness to be minimized, nor to be a single number; similarly the inner EAs may return many solutions, not just one; we pose here these limitations just for clarity.

for obtaining solutions to be evaluated with f. Our CCEA does not pose any constraint concerning this possibility; in particular, the two search spaces G_1 and G_2 of the inner EAs may be different, as well as the two mapping functions ϕ_1 and ϕ_2—that is, the two EAs may employ different solution representations.

An iterative, population-based EA can be described by providing the bodies for the `init()`, `stop()`, `update()`, and `extractSolution()` functions used by `solve()`. The progress of the EA is stored in its state s, which hosts the population, together with additional information (specific to each EA) about the ongoing optimization. The state is initialized by the `init()` function, and is then iteratively updated within `update()`: these functions respectively perform the initialization of the population, which is then stored in the state, and its update via some forms of genetic operators. The optimization process then iteratively proceeds until a termination condition is met, tested in `stop()`, e.g., an evaluation budget has been exceeded, and finally the solution is extracted from the state s via `extractSolution()` and returned.

```
function solve(f):
    s ← init(f)
    while !stop(s) do
        s ← update(s, f)
    end
    return extractSolution(s, f)
end
```

Fig. 1: General structure of an iterative EA. s is the generic state of the EA. $f : P \to \mathbb{R}$ is the fitness function.

Our CCEA is itself an iterative, population-based EA, and hence we can define it by describing its `init()`, `stop()`, `update()`, and `extractSolution()` functions, which are shown in detail in Figure 2.

The working principle of our CCEA is that it delegates the evolution of the two components of the solution to two inner EAs, that we denote with EA_1 and EA_2, feeding them with *disposable* fitness functions created on-the-fly. Such disposable fitness functions are needed because the inner EAs evolve components of the solution, rather than full solutions. Hence, they cannot be assessed with f, which is defined on P, but need a $f_1 : P_1 \to \mathbb{R}$ and $f_2 : P_2 \to \mathbb{R}$, where P_1 and P_2 are the set of possible components for EA_1 and EA_2, i.e., their solution spaces. Moreover, by building a disposable f_1 (or f_2) on-the-fly, we can enclose the collaborator selection and aggregation steps inside the function, thus making it dependent on the current population of candidate collaborators.

Each disposable fitness function evaluates an individual of the corresponding population (i.e., a component of the solution) by performing these three steps: (1) select a portion of the other population (*collaborator selection*); (2) use f to compute the fitness of each pair resulting by combining the current individual with a collaborator of the selected subset; (3) aggregate the resulting fitness values (*fitness aggregation*). In the algorithms of Figure 2, the collaborator selection

```
function init(f):
    f_1 ← p_1 ↦ 0
    f_2 ← p_2 ↦ 0
    s_1 ← init_1(f_1)
    s_2 ← init_2(f_2)
    P'_1 ← cSel(s_1.pop)
    P'_2 ← cSel(s_2.pop)
    f_1 ← p_1 ↦ cAggr([f(p_1 ⊕ p_2)]_{p_2 ∈ P'_2})
    f_2 ← p_2 ↦ cAggr([f(p_1 ⊕ p_2)]_{p_1 ∈ P'_1})
    s.n ← 0
    s.s1 ← init_1(f_1)
    s.s2 ← init_2(f_2)
    return s
end

function stop(s):
    return stop_1(s.s1) ∨ stop_2(s.s2)
end

function update(s,f):
    P'_1 ← cSel(s.s1.pop)
    P'_2 ← cSel(s.s2.pop)
    f_1 ← p_1 ↦ cAggr([f(p_1 ⊕ p_2)]_{p_2 ∈ P'_2})
    f_2 ← p_2 ↦ cAggr([f(p_1 ⊕ p_2)]_{p_1 ∈ P'_1})
    s.s1 ← update_1(s.s1, f_1)
    s.s2 ← update_2(s.s2, f_2)
    return s
end

function extractSolution(s,f):
    P'_1 ← [p_1 : (p_1, q) ∈ s.s1.pop]
    P'_2 ← [p_2 : (p_2, q) ∈ s.s2.pop]
    (p*_1, p*_2) ← argmax_{(p_1,p_2) ∈ P'_1 × P'_2} f(p_1 ⊕ p_2)
    return p_1 ⊕ p_2
end
```

Fig. 2: The `init()`, `update()`, `stop()`, and `extractSolution()` functions for our CCEA. The $\oplus$ operator composes a solution $p = p_1 \oplus p_2 \in P$ from two components $p_1 \in P_1$ and $p_2 \in P_2$. $\texttt{init}_1()$, $\texttt{init}_2()$, $\texttt{update}_1()$, $\texttt{update}_2()$, $\texttt{stop}_1()$, and $\texttt{stop}_2()$ are the corresponding functions for the first and second EA, searching respectively in P_1 and P_2. $x \mapsto y$ represents a literal for a function that maps an x to an y: hence, $f_1 \leftarrow p_1 \mapsto 0$ means that f_1 becomes a function that maps any p_1 to $f_1(p_1) = 0$.

step is performed by the `cSel()` function, which receives a population, i.e., a multiset of pairs (p, q), p being the solution and q being its fitness, and returns a multiset of solutions p, extracted from the population passed as argument. The fitness aggregation step is performed by the `cAggr()` function, which takes a multiset of real values in input, and outputs a single real value.

Going more into details about each part of the CCEA, the core of the CCEA lies in the `update()` function, which is responsible for performing each evolutionary step. It starts by selecting the collaborators P'_1 and P'_2, using `cSel()`, to evaluate the solutions components, from the first and the second current subpopulation, respectively. Then, `update()` proceeds by defining the disposable fitness functions f_1 and f_2 to evaluate the components: internally, both rely on f to compute the fitness of a pair of solution components. To complete the evolutionary step, `update()` invokes $\texttt{update}_1(s.\texttt{s1}, f_1)$ and $\texttt{update}_2(s.\texttt{s2}, f_2)$ to have the inner EAs perform an evolutionary step and update their states (and populations).

The function `init()`, responsible for starting the CCEA, has a similar structure to the `update()` function. However, since the collaborator selection `cSel()` takes a population $\left[(p^{(i)}, q^{(i)})\right]_i$ of individuals, rather than just a population $\left[p^{(i)}\right]_i$ of components, there is an additional preliminary step where we define two dummy fitness functions, which map all solutions to a 0 value of fitness. At this point, the `cSel()` can effectively be applied to select the individuals from each population, even though here the selection is driven only by randomness.

Then, the two fitness functions are defined, and the initialization is concluded with the invocation of the inner initializers, $\texttt{init}_1(f_1)$ and $\texttt{init}_2(f_2)$.

The last two building blocks of the CCEA are the termination condition check and the extraction of the solution to be returned. The first is performed within the `stop()` function, which checks if any of the two inner EAs has achieved its termination condition. The second, performed by the `extractSolution()` function, computes all the possible combinations among the components, and returns the best one according to the fitness function f. This should increase the likelihood of achieving a good solution in the end, regardless of the criterion used during evolution to sample the collaborators. In fact, some criteria might be helpful at steering the evolutionary search, but they might not be suitable for selecting the final combination of components.

Following the provided schema, we highlight that there is great freedom in selecting (a) the inner EAs, as long as they comply to the structure of Figure 1, (b) the collaborator selection, which can be freely chosen implementing `cSel()` as desired, and (c) the fitness aggregation function, which can be customized in the `cAggr()` function. The latter two are the main focus of this study.

Concerning the collaborator selection, we consider two *sorting* variants and different *proportion rates*. For the sorting we opt for *First* and *Last*, whereas for the proportion rate we experiment with different rates in $r \in \{0.1, 0.25, 0.5, 0.75, 1\}$. Namely, we first order the population in ascending or descending order (depending on first/last), and then we take the top $r|s.\texttt{pop}|$ individuals, corresponding to the desired proportion of the population $s.\texttt{pop}$.

Regarding the fitness *aggregator*, we consider three cases: the *Best*, the *Worst*, and the *Median*. We prefer the median over the mean, as it can be generalized to non-numerical fitness values, and it is also less sensitive to outliers.

In the following, we denote an instance of our CCEA as EA_1+$\text{EA}_2/s/r/a$, where s is the sorting (F for First and L for Last), r is the proportion rate, a is the aggregator (B for Best, M for Median, and W for Worst). For instance, ES+GA/F/0.1/M is the CCEA that uses ES and GA, selects the best 10 % of the other population as collaborators, and takes the median fitness of the resulting solutions.

3 Case studies

We consider four case studies to evaluate the practicability and the generality of the proposed CCEA scheme. Moreover, we use them as a test bed for assessing the impact of different pairs of inner EAs, collaborator selection schemes, and fitness aggregation functions. To this end, we start from two simple toy problems, which serve mostly as a proof-of-concept, and then we move towards two real-world problems, namely Symbolic Regression (SR) and Neuroevolution (NE).

For each of the examined case studies, we split the search of a solution $p \in P$, into the search of two components $p_1 \in P_1$ and $p_2 \in P_2$, such that $\forall p_1, p_2 : p_1 \oplus p_2 \in P$, and we measure the quality of p with a fitness function $f : P \to \mathbb{R}$. We delegate the optimization of p_1 and p_2 to two inner EAs, which search

two genotype spaces, G_1 and G_2 respectively, and make use of two genotype-phenotype mapping functions, $\phi_1 : G_1 \to P_1$ and $\phi_2 : G_2 \to P_2$, to obtain the solution from the genotype. Clearly, the spaces G_1, G_2, P_1, P_2, P and the functions $\phi_1, \phi_2, \oplus, f$ strongly depend on the problem at hand. We hence describe them in detail in the following sections.

3.1 Toy problems

Point aiming. The goal of the point aiming problem is to find a point in $\mathbb{R}^n$ as close as possible, according to the Euclidean distance, to a target point $\boldsymbol{x}^* \in \mathbb{R}^n$. The solution space P is hence $\mathbb{R}^n$, and the fitness of a solution $p = \boldsymbol{x}$ is the Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{x}^*$, $f(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{x}^*\|_2$. We split the search into the search of the first $\lfloor \frac{n}{2} \rfloor$ components ($P_1 = \mathbb{R}^{\lfloor \frac{n}{2} \rfloor}$), and the last $\lceil \frac{n}{2} \rceil$ components ($P_2 = \mathbb{R}^{\lceil \frac{n}{2} \rceil}$), and we rely on a direct encoding for both, i.e., $G_1 = P_1$, $G_2 = P_2$. We compose $p = \boldsymbol{x}$ from $p_1 = \boldsymbol{x}_1$ and $p_2 = \boldsymbol{x}_2$ through simple concatenation.

We note that this problem is fully-separable [11], meaning that it can be solved by optimizing each decision variable independently. These types of problems are, in general, easily tackled by CCEAs [4].

Bimodal point aiming. The bimodal point aiming problem increases the difficulty of the point aiming problem, by introducing a bimodal fitness landscape. Namely, two possible target points, $\boldsymbol{x}^*, \boldsymbol{x}^{**} \in \mathbb{R}^n$, are considered, and the fitness of a candidate solution $\boldsymbol{x}$, is measured as $f(\boldsymbol{x}) = \min\left(\|\boldsymbol{x} - \boldsymbol{x}^*\|_2, \|\boldsymbol{x} - \boldsymbol{x}^{**}\|_2\right)$, i.e., as its Euclidean distance to the closest among $\boldsymbol{x}^*$ and $\boldsymbol{x}^{**}$. This causes the problem to not be fully-separable anymore, as the solution components need to agree on the direction in which they are moving, in order to both get closer to either $\boldsymbol{x}^*$ and $\boldsymbol{x}^{**}$.

3.2 Symbolic regression

We consider SR as our first real-world problem, where the goal is to find a symbolic formula $h : \mathbb{R}^d \to \mathbb{R}$ which best approximates the relation between a data point $\boldsymbol{x} \in \mathbb{R}^d$ and a dependent variable $y \in \mathbb{R}$ expressed in a dataset $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{i=n}$. Hence, the solution consists of a symbolic formula $h : \mathbb{R}^d \to \mathbb{R}$, and we measure its fitness $f(h)$ in terms of mean squared error (MSE) over the training data: $f(h) = \text{MSE}\left(h, \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{i=n}\right)$.

We split the problem into the search of the skeleton of the formula (p_1) and the optimization of the numerical constants which appear therein (p_2), as inspired by [19]. More in details, we define P_1 as the space of parametric symbolic formulae $p_1 : (\mathbb{R}^d, \mathbb{R}^m) \to \mathbb{R}$, and P_2 as $\mathbb{R}^m$. We rely on a tree-based encoding for the formula, as in standard Genetic Programming (GP), hence, G_1 becomes the space of trees where the inner nodes are mathematical operators (namely, $\bullet + \bullet, \bullet - \bullet, \bullet \times \bullet, \bullet \div \bullet, \bullet \div^* \bullet, \log \bullet, \log^* \bullet, \exp \bullet, \sin \bullet, \cos \bullet, \frac{1}{\bullet}, -\bullet, \sqrt{\bullet}, \bullet^2, \bullet^3, \max(\bullet, \bullet)$,

$\min(\bullet, \bullet)$, where $\bullet$ represents an operand, and operators marked with * are protected), and the leaf nodes are either problem features $x_1, \ldots, x_d$ or parameters $c_1, \ldots, c_m$. Conversely, we use a direct encoding for the second component. For composing the solutions, we substitute each parameter c_i in the formula p_1 with the corresponding element of p_2, $c_i = p_{2,i}, \forall i = 1, \ldots, m$.

We refrain from using linear scaling, although it can dramatically increase the performance of SR on real-world problems [20], as it could pollute the results achieved through CC. In fact, our goal here is not that of achieving state-of-the-art performance, but we aim mainly at investigating the effects of different aspects within CC.

We highlight that this problem is far from being separable, as there is a strong intertwining between the two components. This poses additional hurdles on the CCEA, and enables us to observe its behavior under more difficult circumstances.

3.3 Neuroevolution

The second real-world problem we analyze is NE, where the aim is that of finding a suitable Artificial Neural Network (ANN) to solve a task. We here consider binary classification, hence we search the ANN $h : \mathbb{R}^d \to \{y^+, y^-\}$ which best captures the relation between a data point $\boldsymbol{x} \in \mathbb{R}^d$ and its class $y \in \{y^+, y^-\}$ expressed in a dataset $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{i=n}$. The solution thus consists of an ANN h with d input neurons (one per feature), and 2 output neurons (one per class): $h(\boldsymbol{x})$ is y^+ if the first neuron activation is larger than the second neuron activation, or y^- otherwise. We measure fitness with the Balanced Error Rate (BER) over the training data: $f(h) = \text{BER}\left(h, \left\{\left(\boldsymbol{x}^{(i)}, c^{(i)}\right)\right\}_{i=1}^{i=n}\right)$.

Motivated by the work of Gaier and Ha [1], we split the NE task in the search of a suitable architecture for the ANN and in the optimization of the ANN weights. However, to constrain and ease the optimization—we remark our goal is not that of excelling at NE—we take inspiration from the practice of pruning [22], which has been shown to yield to well performing ANNs even in presence of high sparsity [8, 9]. Hence, we fix the architecture of the ANN in terms of number and size of the hidden layers and of activation functions (we always use tanh), but allow for synapses to be present or not, ranging from a completely disconnected ANN to a fully connected one. Therefore, p_1 consists of the weights of the ANN, as if it was fully connected, whereas p_2 consists of a binary mask for the ANN, indicating which synapses are pruned. For composing the final solution p, we simply mask the weights of p_1 using p_2. Concerning the evolution of the components, we encode the weights as real-valued vectors, and the mask as a bit-string. Hence, let w be the maximum number of weights of the ANN, $G_1 = \mathbb{R}^w$ and $G_2 = \{0, 1\}^w$.

4 Experimental analysis

We hereon present the details and the results of our experimental evaluation. We implemented the proposed CCEA in JGEA [6]. For all the problems de-

scribed in the following sections we performed 10 independent evaluations for each considered scenario, unless otherwise specified.

4.1 Toy problems

For both toy problems we considered the same settings. We set the problem size to $n = 25$, resulting in $P = \mathbb{R}^{25}$, $P_1 = \mathbb{R}^{12}$, $P_2 = \mathbb{R}^{13}$. For the point aiming problem we set $\boldsymbol{x}^* = 2 \cdot \mathbf{1}_{25}$, while for the bimodal point aiming problem we set $\boldsymbol{x}^* = 2 \cdot \mathbf{1}_{25}$, $\boldsymbol{x}^{**} = -1 \cdot \mathbf{1}_{25}$. We considered two EAs: a simple form of Evolutionary Strategy (ES) and a Genetic Algorithm (GA). We experimented with both EAs on their own (i.e., without CC) as a baseline, and as inner EAs in the CCEA. For the latter case, we had ES+ES/s/r/a, ES+GA/s/r/a, and GA+GA/s/r/a, with s in {F,L}, r in $\{0.1, 0.25, 0.5, 0.75, 1\}$, and a in {B,M,W}. For both EAs, we evolved a population of size $n_{\text{pop}} = 50$ for $n_{\text{gen}} = 70$ generations. We initialized individuals sampling each component uniformly in $[0, 1]$. Then for the ES, we used a $(1 + \lambda)$ model, where we generate the offspring applying a Gaussian mutation ($\sigma = 0.1$) to the point-wise mean of the parents, selected as the top fourth of the current population. For the GA, we used a $(\mu + \lambda)$ scheme, with $\mu = \lambda$. We relied on tournament selection ($n_{\text{tour}} = 5$) for sampling the two parents, which we combined with geometric crossover followed by a Gaussian mutation ($\sigma = 0.1$) to produce the offspring.

We report the results the experimental evaluation in Figures 3 to 5 for point aiming, and in Figures 6 and 7 for bimodal point aiming. These results constitute a proof of concept for the practicability of our CCEA, as we successfully combined different EAs, and experimented with various collaborator selection methods and fitness aggregation criteria. Focusing more on the single figures, we can gain different insights. First, in Figure 3, we report the median across the 10 runs of the fitness of the best evaluated individual vs. the amount of fitness evaluations performed during evolution. Clearly, since we used the number of generations as termination criterion, some lines stop earlier than others. This highlights the trade-off between the amount of collaborators chosen and the computational effort required: a higher rate r, requires more evaluations to compute an evolutionary step. From the figure we can also note that at the end of evolution all combinations reach comparable fitness levels, meaning that evolution was always able to achieve reasonable solutions regardless of the employed components. These findings are also confirmed by Figure 4, where we plot, for each generation, the median across the 10 runs of the fitness of the best evaluated individual. In fact, we can see the plots almost perfectly overlapping at the end of evolution. Given the insights gained from these two figures, we limit ourselves to showing the progression of fitness along generations for the other problems, in order to save space.

Moving on to Figure 5, we can reason further on the impact of collaborator selection and fitness aggregation on the solutions found by the CCEA. We here report the median fitness of the best individual found at the end of evolution. Concerning the fitness aggregation (one per column), Best generally yields to better results, although comparable to both Median and Last. Focusing on the

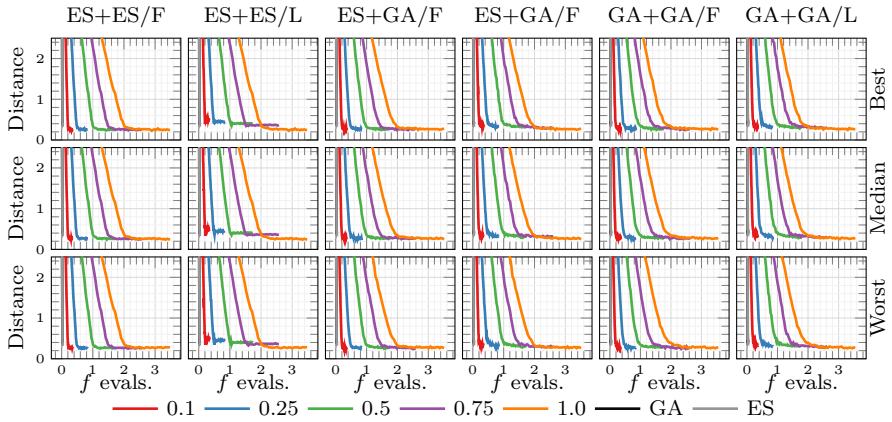

Fig. 3: Fitness (distance from the target) vs. number of fitness evaluations [$\times 10^5$] for the point aiming problem, one line for each rate or baseline EA. The ES line overlies the GA line.

rate r, we observe larger values to be more effective with the Best aggregation scheme, whereas we note a fuzzier impact for Median and Large. Last, regarding the collaborator selection, First performs better than Last, especially for ES+ES. We speculate this could be because both ES populations focus on a specific point in the search space, the mean, and the combination with the another components tends to move it causing instability.

For the bimodal problems we obtained similar results, see Figures 6 and 7. In fact, we notice higher differences between plots during evolution, which were smoothed out at the end of evolution. This leads us to conclude that both toy problems considered were easy enough for evolution to eventually converge to a reasonable solution, regardless of the components involved.

4.2 Symbolic Regression

For the SR problem we considered the Boston Housing dataset [2], which consists of 506 examples and 13 features. We used 5 different 80 %/20 % splits of the dataset, which we respectively used as training and test sets, thus resulting in $5 \cdot 10 = 50$ independent evolutionary runs. We set the number of numerical constants to $m = 10$, hence $P_2 = \mathbb{R}^{10}$. We employed GP as EA_1, and ES as EA_2, but we also considered, for reference, a variant with GP on its own, where the terminal nodes could be the problem features or a constant among $\{0.1, 1, 10\}$. We set $n_{\text{pop}} = 100$ and $n_{\text{gen}} = 500$. For GP we used a $(\mu + \lambda)$ scheme, with *ramped half-and-half* initialization and tournament selection ($n_{\text{tour}} = 10$). We used either subtree crossover or subtree mutation to compute the offspring, chosen with $p_{\text{xo}} = 0.8$ and $p_{\text{mut}} = 1 - p_{\text{xo}} = 0.2$. For the ES we used the same configuration as described in section 4.1.

We report the results for the SR problem in Figures 8 to 10. In Figure 8, we show the median fitness of the best evaluated individual along generations.

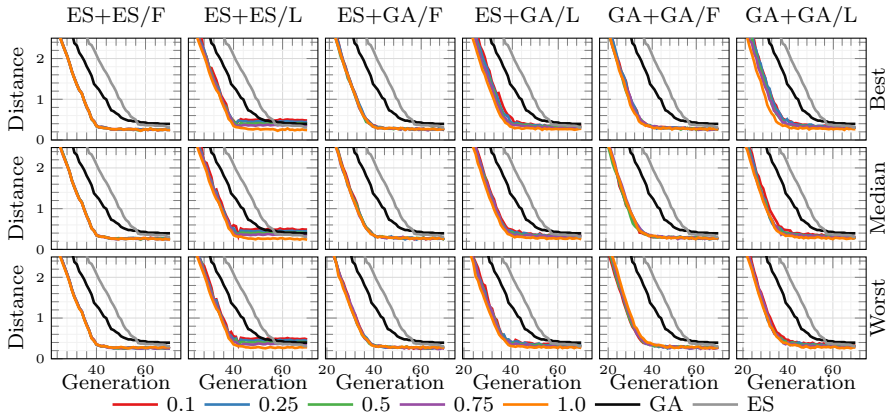

Fig. 4: Fitness (distance from the target) vs. generation for the point aiming problem, one line for each rate or baseline EA.

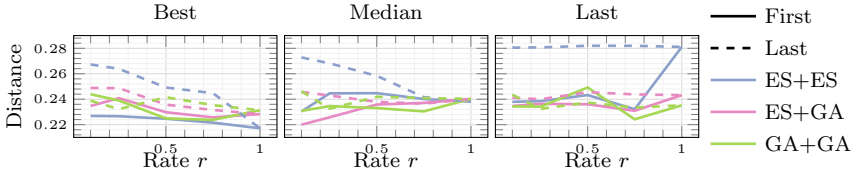

Fig. 5: Fitness at the end of evolution vs. rate r for the point aiming problem.

Concerning the fitness aggregation, we see an ordering: Best is better than Median, which is better than Worst. Regarding the collaborator selection, instead, the results are not too different, and, in fact, different rates and First and Last achieve similar fitness results at the end of evolution.

More insights can be gained from Figure 9, where we report the fitness (MSE on the training set) of the best solutions found at the end of evolution, together with the MSE of said solutions on the test set. From what we see in this figure, the previous findings appear confirmed, also in terms of generalization: the collaborator selection plays a secondary role compared to the fitness aggregation, where the Best is clearly better than Median or Worst. We can also note, once again, that the amount of collaborators, i.e., the rate r, does not influence the performance of the CCEA, hence we deem convenient to use a lower rate to constrain the amount of computational resources used. From another point of view, these results seem to suggest that the optimistic expectation that coupling each component with unfit components of the other population results in a more robust solution is not met.

Last, we investigated the structure of the final solutions found by the CCEA for SR. In Figure 10 we plot the median amount of constants in the best evaluated individual along generations. From here, the differences between columns, i.e., between fitness aggregation schemes, are evident: the more pessimistic the

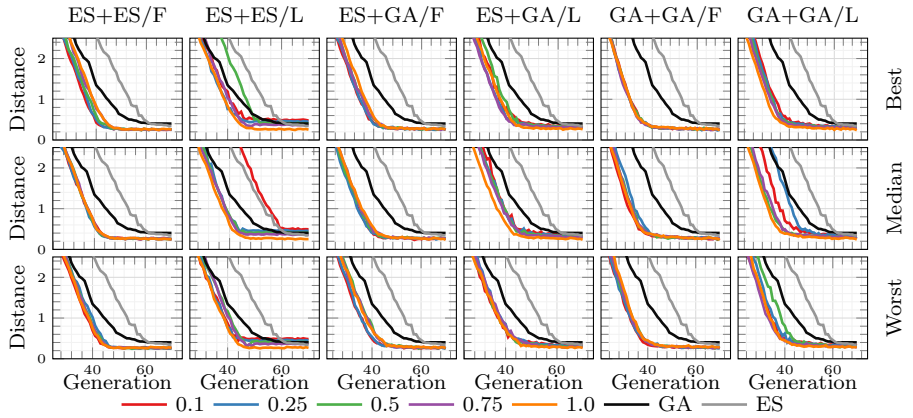

Fig. 6: Fitness (distance from the target) vs. generation for the bimodal point aiming problem, one line for each rate or baseline EA.

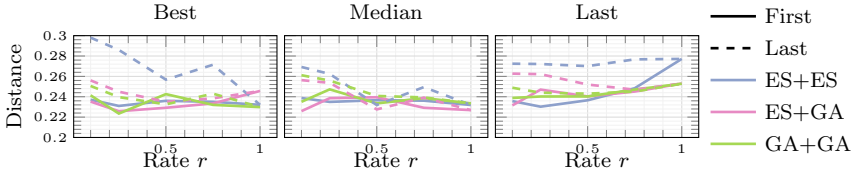

Fig. 7: Fitness at the end of evol. vs. rate r for the bimodal point aiming problem.

fitness aggregation, the less cooperation between populations is preferred. In fact, evolution tends to try to stabilize the fitness of the tree component by avoiding the combination with constants, as few poor performing ones could hinder the final fitness of a tree, good though it might be in combination with other values.

4.3 Neuroevolution

For the NE problem we used the German Credit dataset [2], consisting of 1000 examples and 20 features. As in SR, we considered 5 splits for the dataset. However, here, we performed 5 experiments on each split, rather than 10, totaling $5 \cdot 5 = 25$ independent evaluations. We chose an ANN with 3 hidden layers, each of size 26, to match the size of the input. In fact, some of the 20 features were categorical, hence we transformed them with one-hot encoding. This architecture resulted in a maximum number of weights $w = 2160$, hence $P_1 = R^{2160}$ and $P_2 = \{0, 1\}^{2160}$. We relied on ES as EA_1 and on a GA as EA_2. As a baseline, we also considered ES on its own, leaving all the weights unpruned. We set $n_{\text{pop}} = 100$ and $n_{\text{gen}} = 700$. For the ES we used the configuration described in Section 4.1. For the GA we used a similar scheme as for the toy problems, but given the different search space we had some key differences. Namely, we initialized the individuals sampling each component from $\{0, 1\}$ with equal probability, and we computed the offspring with either uniform crossover or bitflip mutation, chosen

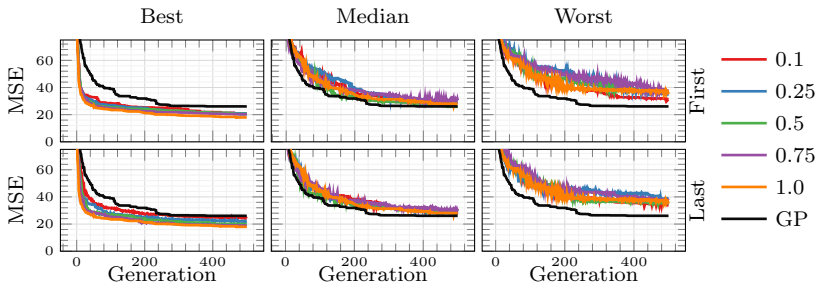

Fig. 8: Fitness (MSE) vs. generation for the SR problem (Boston Housing dataset), one line for each rate or baseline EA.

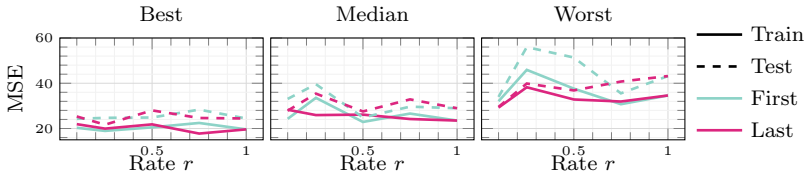

Fig. 9: Fitness (MSE on training set) at the end of evolution and MSE on the test set vs. rate r for the SR problem.

with equal probability ($p_{\text{xo}} = p_{\text{mut}} = 0.5$). Again, we used tournament selection, with $n_{\text{tour}} = 10$.

We report the results in Figures 11 and 12. In Figure 11 we display the median of the fitness of the best evaluated individual along generations. Here we can clearly notice a difference for the third column: with the Worst aggregation, evolution is not able to converge for rates $r > 0.1$. In fact, instead of achieving high selective pressure, the pessimist fitness aggregation is steering evolution in the wrong direction. We speculate this does not happen for a low rate of collaborators as this corresponds to a smaller pool of individuals among which the worst fitness is extracted, meaning that it is more unlikely to have a low value if the evaluated component is promising.

The results at the end of evolution, reported in Figure 12 together with the BER resulting from re-evaluation on the test set, are in line with the previous findings, and confirm the poor performance of the CCEA with the Worst fitness aggregation scheme. Conversely, we do not note significant differences neither between the Best and the Median aggregation schemes, nor between the First and Last collaborator selection criteria. Anyway, it appears that a smaller rate of collaborators is not only sufficient, but also beneficial in most cases.

Regarding the solutions structure, for space reasons we avoid reporting any plots. However, the results showed a constant amount of weights pruned, around 50 %, meaning that there was no evolutionary pressure in that sense.

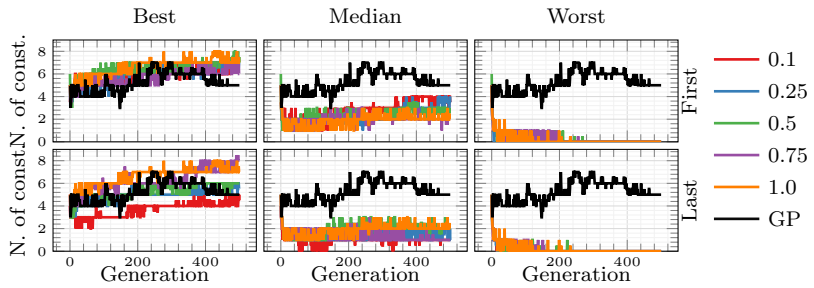

Fig. 10: Amount of constants in the best individual vs. generation for the SR problem, one line for each rate or baseline EA.

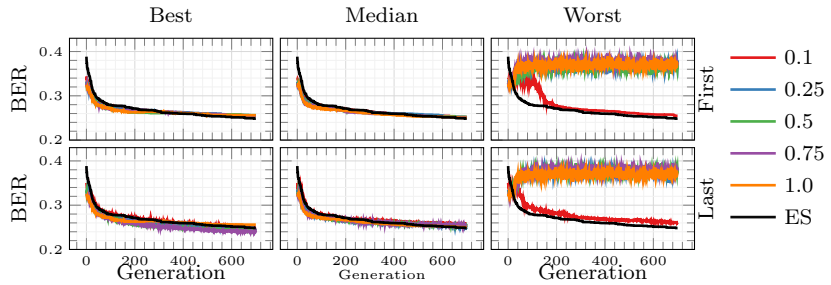

Fig. 11: Fitness (BER) vs. generations for the NE classification problem.

5 Concluding remarks

We considered Cooperative Coevolution (CC), that aims at solving a problem by dividing it in simpler sub-problems, and we conducted an experimental campaign for investigating the impact of key CC components on the effectiveness and efficiency of optimization and on the complexity and generalization power of optimized solutions.

We first proposed a general scheme for a CC evolutionary algorithm (CCEA) for the significant case where the problem can be split in two sub-problems. Our CCEA formulation is general enough to use, as inner EAs for the two sub-problems, any population-based iterative EA with any representation. At the same time, it neatly modularizes the key CC components that we study in this paper, namely (a) how to *select* the collaborators in the other population (including how many of them) and (b) how to *aggregate* the fitness of many collaborator-collaborator pairs to obtain a single fitness value. Then, we instantiated our CCEA with different combinations of inner EAs, selection, and aggregation schemes and we applied it to two toy problems and two real prediction problems

We found that: (a) a small number of collaborators is often enough to have good effectiveness and efficiency; (b) optimistic fitness aggregation schemes (i.e., choosing the best fitness among all the collaborator pairs) often results in better effectiveness; (c) depending on the way the problem is split, fitness aggregation

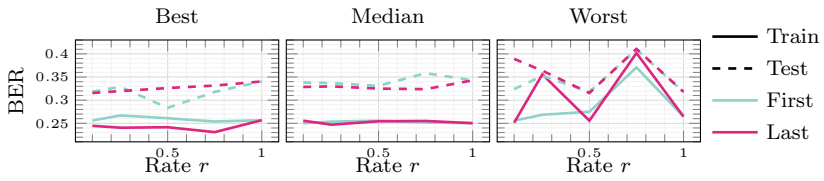

Fig. 12: Fitness (BER on training set) at the end of evolution and BER on the test set vs. rate r for the NN classification problem.

may have a dramatic impact on the structure of evolved solutions. Moreover, we found no evidence of increased generalization power when more or unfit collaborators are selected. We believe that our study may help gaining insights in a sub-field of evolutionary computation, i.e., CC, that is promising, yet not completely characterized.

References

[1] Gaier, A., Ha, D.: Weight agnostic neural networks. Advances in neural information processing systems **32** (2019)
[2] La Cava, W., Orzechowski, P., Burlacu, B., de França, F.O., Virgolin, M., Jin, Y., Kommenda, M., Moore, J.H.: Contemporary symbolic regression methods and their relative performance. arXiv preprint arXiv:2107.14351 (2021)
[3] Luke, S., Sullivan, K., Abidi, F.: Large scale empirical analysis of cooperative coevolution. In: Proceedings of the 13th annual conference companion on Genetic and evolutionary computation, pp. 151–152 (2011)
[4] Ma, X., Li, X., Zhang, Q., Tang, K., Liang, Z., Xie, W., Zhu, Z.: A survey on cooperative co-evolutionary algorithms. IEEE Transactions on Evolutionary Computation **23**(3), 421–441 (2018)
[5] Maniadakis, M., Trahanias, P.: Assessing hierarchical cooperative coevolution. In: 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), vol. 1, pp. 391–398, IEEE (2007)
[6] Medvet, E., Nadizar, G., Manzoni, L.: JGEA: a modular java framework for experimenting with evolutionary computation. In: Proceedings of the genetic and evolutionary computation conference companion, pp. 2009–2018 (2022)
[7] Moriarty, D.E., Miikkulainen, R.: Forming neural networks through efficient and adaptive coevolution. Evolutionary computation **5**(4), 373–399 (1997)
[8] Nadizar, G., Medvet, E., Pellegrino, F.A., Zullich, M., Nichele, S.: On the effects of pruning on evolved neural controllers for soft robots. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1744–1752 (2021)
[9] Nadizar, G., Medvet, E., Ramstad, H.H., Nichele, S., Pellegrino, F.A., Zullich, M.: Merging pruning and neuroevolution: towards robust and efficient

controllers for modular soft robots. The Knowledge Engineering Review **37** (2022)
[10] de Oliveira, F.B., Enayatifar, R., Sadaei, H.J., Guimarães, F.G., Potvin, J.Y.: A cooperative coevolutionary algorithm for the multi-depot vehicle routing problem. Expert Systems with Applications **43**, 117–130 (2016)
[11] Omidvar, M.N., Li, X., Mei, Y., Yao, X.: Cooperative co-evolution with differential grouping for large scale optimization. IEEE Transactions on evolutionary computation **18**(3), 378–393 (2013)
[12] Panait, L., Luke, S.: A comparative study of two competitive fitness functions. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002), pp. 503–511, Citeseer (2002)
[13] Panait, L., Luke, S., Harrison, J.F.: Archive-based cooperative coevolutionary algorithms. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 345–352 (2006)
[14] Peng, X., Liu, K., Jin, Y.: A dynamic optimization approach to the design of cooperative co-evolutionary algorithms. Knowledge-Based Systems **109**, 174–186 (2016)
[15] Popovici, E., De Jong, K.A., et al.: A Dynamical Systems Analysis of Collaboration Methods in Cooperative Co-evolution. In: AAAI Fall Symposium: Coevolutionary and Coadaptive Systems, pp. 26–34 (2005)
[16] Potter, M.A., Jong, K.A.D.: A cooperative coevolutionary approach to function optimization. In: International conference on parallel problem solving from nature, pp. 249–257, Springer (1994)
[17] Potter, M.A., Jong, K.A.D.: Cooperative coevolution: An architecture for evolving coadapted subcomponents. Evolutionary computation **8**(1), 1–29 (2000)
[18] Tan, K.C., Yang, Y., Goh, C.K.: A distributed cooperative coevolutionary algorithm for multiobjective optimization. IEEE Transactions on Evolutionary Computation **10**(5), 527–549 (2006)
[19] Vanneschi, L., Mauri, G., Valsecchi, A., Cagnoni, S.: Heterogeneous cooperative coevolution: strategies of integration between gp and ga. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 361–368 (2006)
[20] Virgolin, M., Alderliesten, T., Bosman, P.A.: Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In: Proceedings of the genetic and evolutionary computation conference, pp. 1084–1092 (2019)
[21] Wiegand, R.P., Liles, W.C., De Jong, K.A., et al.: An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In: Proceedings of the genetic and evolutionary computation conference (GECCO), vol. 2611, pp. 1235–1245, Morgan Kaufmann San Francisco (2001)
[22] Zullich, M., Medvet, E., Pellegrino, F.A., Ansuini, A.: Speeding-up pruning for artificial neural networks: introducing accelerated iterative magnitude pruning. In: 2020 25th International Conference on Pattern Recognition (ICPR), pp. 3868–3875, IEEE (2021)