CE Project

Diogo Pinto[1 [2024156583]] and Giovanni Maffeo[2,3 [2024232210]]

Abstract. This project investigates evolutionary algorithms for robot co-optimization, applying Genetic Algorithms for structure, Evolutionary Strategies for controllers, and a combined Coevolutionary approach. We performed hyperparameter tuning and statistical testing to validate results. The methods consistently outperformed random baselines, and the project emphasized the importance of fitness design, operator choice, and statistical rigor. It served as a practical and challenging application of theoretical concepts.

Keywords: Genetic Algorithm · Evolutionary Strategy · Coevolution.

1 Introduction

1.1 Representation

- **Evolve Structure:** Grid of voxels defining the robot's body.
- **Evolve Controller:** Neural network weights controlling a fixed structure.
- **Evolve Both:** Combines structure and controller from separate populations.

2 Evolving the structure

2.1 Context

In the first part of the implementation, Genetic Algorithm was implemented, since the problem in hands has a discrete space, the structure of the robot can be described in vectors of integers so it wouldn't make sense to apply for example Evolutionary Strategies where noise is added.

2.2 Structure Connectivity

The first challenge we encountered during the implementation of the structure evolution algorithm arose when using the `get_full_connectivity` method, which is used to perform the fitness evaluation. Whenever a disconnected robot is received, this method raises an exception.

Since our representation in the Evolving the structure phase, described in Section 1.1, is directly modified by the operators presented in the following sections, this issue is particularly relevant.

To address it, we implemented a function named `is_connected`, which performs a Breadth-First Search (BFS) treating the structure as a graph. This

Algorithm 1 Breadth-First Search (BFS)

```
Input: Graph G = (V, E), starting node s
Unmark all vertices
Initialize empty queue Q
Mark s and insert s into Q
while Q is not empty do
    Remove v from Q
    for each neighbor w of v do
        if w is not marked then
            Mark w
            Insert w into Q
        end if
    end for
end while
```

allows us to verify whether the structure is fully connected: if the BFS reaches all nodes (in this case, all voxels), the structure is considered connected.

The pseudocode of the BFS used for connectivity validation is presented below:

The following operators were implemented as part of the evolutionary process:

1. **Parent Selection:** We used tournament selection, where a subset of individuals of size `PARENT_SELECTION_COUNT` is randomly sampled from the population, and the two best individuals among them are selected as parents.
2. **Crossover:** We implemented one-point, two-point, and uniform crossover methods. The specific crossover function used in each run is defined by the `CROSSOVER_TYPE` parameter and is applied with probability `CROSSOVER_RATE`.
3. **Mutation:** A random voxel in the structure is altered with probability `MUTATION_RATE`, changing its type to another valid one.
4. **Survivor Selection:** After offspring are generated, the best `SURVIVORS_COUNT` individuals from the previous generation are preserved, and the remaining individuals are filled with the new offspring to form the next generation.

2.3 Implementation

Following the description of the operators and hyperparameters presented in Section **??**, the implementation can be summarized as follows: first a random population is initialized and for a given amount of generations, two parents are selected to generate new individuals through crossover and mutation. At the end of each generation, the best individuals from the previous generation are preserved through survivor selection, and new individuals generated by crossover and mutation are added to form the next population.

Several crossover types were implemented, namely one-point, two-point, and uniform crossover. The most appropriate approach to validate these variations

would be through statistical testing, as done for the hyperparameters in Section 5. However, due to the high execution time required, we conducted offline preliminary testing for this component. Although the results lack statistical significance, we ultimately considered the uniform crossover to be the most effective, as it consistently returned better outcomes in our tests.

Additionally, whenever a crossover occurs and a new child is generated, its connectivity is evaluated. If the resulting robot is not fully connected, the algorithm attempts to generate a new child up to a predefined maximum number of tries. If no valid robot is produced within this limit, one of the parents is returned instead.

For the mutation function, each position of the robot structure has a probability (mutation rate) of suffering a mutation, and then if it has a probability, it's selected one of the possible voxels that is different from the current one and it's analyzed if the structure is still connected, if it's not then, tries again with the other options and if nothing works then the mutation is canceled and the father is returned.

Algorithm 2 Genetic Algorithm (GA)

```
Initialize population P with N random individuals
for generation = 1 to G do
    Evaluate fitness of all individuals in P
    Select p two best random parents from a subsection of P
    Apply crossover to parents to create X offspring
    Apply mutation to offspring
    Select new population P from previous individuals and P − X offspring
end for
return Best individual found
```

3 Evolving the controller

In this section of the project, Evolutionary Strategies was implemented, since other kinds of algorithm like GA doesn't work here, because what were are trying to evolve is the paramets vectors of a neural network and those are in a continuous space, not discrete.

We initialize a random population and then for a specific amount of iterations we evolve by applying the $(\mu + \lambda)$ algorithm. We choose this one specifically for being an elitism approach and also providing a good diversity of solutions. In this algorithm, noise is added to the vector parameters and for the testing phase we decided to variate the two most important parameters, which is the mutation rate (how often a mutation will occur) and also the sigma (how hard will the mutation) and the values will be further discussed later in the document. It wasn't possible to test with more parameters that could potentially increase the performance because of the time constraint we had to run the project.

Algorithm 3 Evolution Strategy $(\mu + \lambda)$

```
Initialize population of μ individuals
for generation = 1 to G do
    for μ do
        Generate λ offspring through mutation
        Evaluate fitness of all new individuals
        Select the best μ individuals from parents + λ offspring
    end for
end for
return Best individual
```

4 Coevolution

For the last part of the implementation, it was decided to implement an approach where we evolve a structure and for each one we then evolve the controller.

First it was created a random population of individuals with random structures and consequently random controllers (neural networks) created for matching the robot's structure. In the second phase it was created new individuals by executing the code that it was previously created in the evolving structure phase, with the GA algorithm, through uniform crossover (because it was the one that achieved better results) and mutation. After that, it was created the controllers from scratch for those new individuals and the code for evolving the controller was executed for each individual, using the evolutionary strategy that we already had, for a giver number of generations. It is important to note that since we had time constraints, the code was only ran for 30 generations for each controller, and therefore, to try to achieve acceptable results so early, it was decided that was better, through previous "offline" test, to utilize high values for mutation rate and sigma. After each generation of evolving the structure, a tournament selection was performed to keep the best individuals of the current population in the next generation.

5 Tests and results

We performed offline testing to identify which parameters most affected algorithm performance and to define suitable value ranges. Based on this, a hyperparameter search was conducted for the selected parameters. For each combination of hyperparameters the algorithms were ran 5 times with different seeds to analyze the performance and allowing for the statistical tests. Below, we present an example of the results obtained for each run with a given seed, as well as for the entire combination.

Algorithm 4 Coevolution

Initialize population of P individuals
for generation = 1 to G of structure **do**
  Evaluate fitness of all individuals
  Choose the two best parents randomly of a given subset of P
  Generate x structure offspring through crossover and mutation
  for each offspring **do**
    for generation = 1 to G of controllers **do**
      Generate λ offspring by mutation
      Evaluate fitness of all new individuals
      Select the best μ individuals from parents + λ offspring
    end for Update offspring's weights
  end for
  Select the best μ individuals from parents + λoffspring
end for
return Best individual

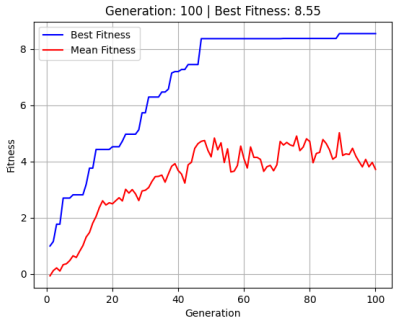

Fig. 1: Example Run Plot

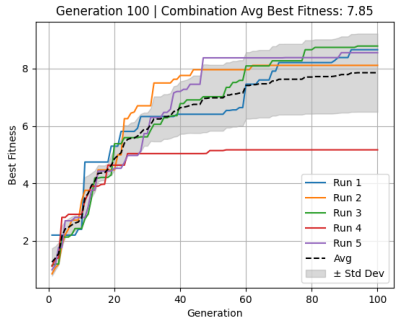

Fig. 2: Example Combination Plot

After selecting the best hyperparameters, we ran tests for each combination of scenario and controller. Since each scenario defines a specific task-controller pair, we reused the same plots for both individual runs and complete combinations.

To identify the best hyperparameters and the most effective scenario-controller combinations, we applied statistical tests, following the methodology discussed in the theoretical lectures. Finally, we compared our results with those obtained using random search. Due to time constraints, we did not implement or run additional baseline algorithms.

The result structure in the project repository is organized as follows:

```
outputs/
  evolve_structure/
    ea_search/
      hyperparams_test/ # Stores results for each run, combination and
          statistical tests
```

```
    controller_scenario_test/ # Same structure as above
  random_search/
    ... # Equivalent test and result structure

evolve_controller/
  ... # Similar organization
evolve_both/
  ... # Similar organization
```

l

Statistical tests were applied to compare our approach with random search (3.1, 3.2 and 3.3), identify the best controller and scenario combinations for each algorithm and evaluate hyperparameters. All results are available in the source code following the structure above.

5.1 Evolving the structure tests

In this section we present our results for the evolving structure phase.

Hyperparameter In the following section we present the results of the hyperparameters search. To verify the statistical validity of comparing hyperparameter combinations, we first applied the Shapiro–Wilk normality test. Combinations 6 and 8 returned p-values lower than 0.05, indicating non-normal distributions. As a result, we proceeded with the non-parametric Kruskal–Wallis test. The test yielded a p-value of 0.5877, suggesting no statistically significant differences between the combinations ($p > 0.05$).

The best robot of this phase, on the first map, reached 8.784 (GIF)of reward with an average reward of the best combination being 7.855. While in the second map the best reward was 4.2340 (GIF) and the average being 3.992.

The hyperparameter combinations tested were:

1. Mutation Rate: 0.03, Crossover Rate: 0.9, Survivors: 3, Parent Selection: 3
2. Mutation Rate: 0.03, Crossover Rate: 0.9, Survivors: 3, Parent Selection: 4
3. Mutation Rate: 0.03, Crossover Rate: 0.9, Survivors: 5, Parent Selection: 3
4. Mutation Rate: 0.03, Crossover Rate: 0.9, Survivors: 5, Parent Selection: 4
5. Mutation Rate: 0.03, Crossover Rate: 0.95, Survivors: 3, Parent Selection: 3
6. Mutation Rate: 0.03, Crossover Rate: 0.95, Survivors: 3, Parent Selection: 4
7. Mutation Rate: 0.03, Crossover Rate: 0.95, Survivors: 5, Parent Selection: 3
8. Mutation Rate: 0.03, Crossover Rate: 0.95, Survivors: 5, Parent Selection: 4
9. Mutation Rate: 0.05, Crossover Rate: 0.9, Survivors: 3, Parent Selection: 3
10. Mutation Rate: 0.05, Crossover Rate: 0.9, Survivors: 3, Parent Selection: 4
11. Mutation Rate: 0.05, Crossover Rate: 0.9, Survivors: 5, Parent Selection: 3
12. Mutation Rate: 0.05, Crossover Rate: 0.9, Survivors: 5, Parent Selection: 4
13. Mutation Rate: 0.05, Crossover Rate: 0.95, Survivors: 3, Parent Selection: 3
14. Mutation Rate: 0.05, Crossover Rate: 0.95, Survivors: 3, Parent Selection: 4
15. Mutation Rate: 0.05, Crossover Rate: 0.95, Survivors: 5, Parent Selection: 3

16. Mutation Rate: 0.05, Crossover Rate: 0.95, Survivors: 5, Parent Selection: 4

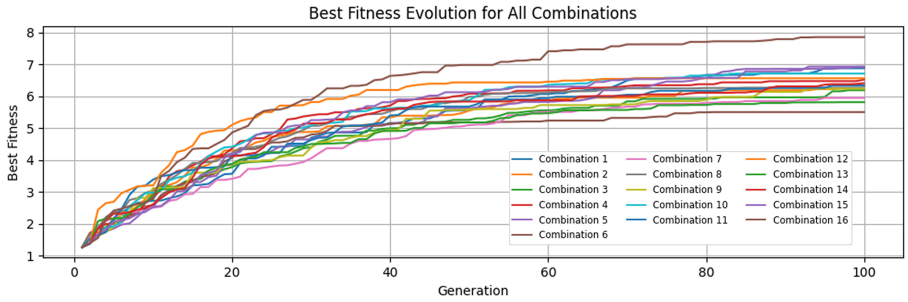

Fig. 3: Fitness evolution for all combinations

Comparison GA and Random Search For each combination of scenario and controller, the Shapiro–Wilk normality test was first applied to determine whether the data followed a normal distribution. Based on the results, if all data were parametric, the Independent t-test was used; otherwise, the Mann–Whitney U test was applied. In all tested combinations, the GA consistently achieved higher fitness values than the random search. The statistical tests confirmed that these differences were significant (p-values < 0.05 in all cases), demonstrating that our implementation clearly outperformed the baseline approach.

Scenario and Controller	**Test**	**p-value**	**Winner**
BridgeWalker-v0 + Sinusoidal wave	Independent t-test	0.0001	GA
BridgeWalker-v0 + Alternating gait	Independent t-test	0.0000	GA
BridgeWalker-v0 + Hopping motion	Mann–Whitney U test	0.0079	GA
Walker-v0 + Sinusoidal wave	Independent t-test	0.0000	GA
Walker-v0 + Alternating gait	Mann–Whitney U test	0.0079	GA
Walker-v0 + Hopping motion	Mann–Whitney U test	0.0079	GA

Table 1: Statistical Comparison GA and Random Search

5.2 Evolving the controller tests

In this section we present our results for the evolving controller phase.

Hyperparameter In this section, we present the results of the hyperparameter search for the Evolutionary Strategies (ES) algorithm. To assess the statistical validity of comparing the different hyperparameter combinations, we first applied the Shapiro–Wilk normality test. Combination 4 returned a p-value below

0.05, indicating a non-normal distribution. Therefore, the dataset was considered non-parametric. Given this, we applied the Kruskal–Wallis test to evaluate whether there were statistically significant differences between the combinations. The test resulted in a p-value of 0.1072, indicating no significant difference between the tested configurations ($p > 0.05$).

In this phase the best robot, for the first map, achieved a fitness of 6.325 (GIF) of reward with an average of the best combination being 6.025. While in the second map the best reward was 4.352 (GIF) and the average of the best combination 3.325.

1. Mutation Rate: 0.1, Sigma: 0.3, Num. Offspring: 3
2. Mutation Rate: 0.1, Sigma: 0.3, Num. Offspring: 5
3. Mutation Rate: 0.1, Sigma: 0.5, Num. Offspring: 3
4. Mutation Rate: 0.1, Sigma: 0.5, Num. Offspring: 5
5. Mutation Rate: 0.3, Sigma: 0.3, Num. Offspring: 3
6. Mutation Rate: 0.3, Sigma: 0.3, Num. Offspring: 5
7. Mutation Rate: 0.3, Sigma: 0.5, Num. Offspring: 3
8. Mutation Rate: 0.3, Sigma: 0.5, Num. Offspring: 5

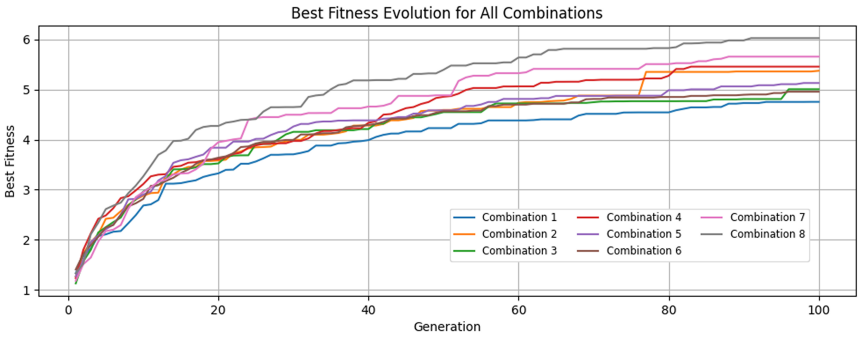

Fig. 4: Fitness evolution for each ES hyperparameter combination.

Comparison ES and Random Search For the comparison between the Evolutionary Strategies (ES) algorithm and the Random Search, the Shapiro–Wilk normality test was first applied to determine whether each dataset followed a normal distribution. Both methods returned p-values greater than 0.05, indicating normality in both cases. Given the parametric nature of the data, an Independent t-test was used to compare the fitness results. The test revealed a statistically significant difference between the two approaches ($p = 0.0000$), with the ES method clearly outperforming the Random Search baseline.

Method	p-value	Normality
Random Search	0.8806	Normal
ES Search	0.4967	Normal

Table 2: Normality test results using Shapiro–Wilk test.

Comparison	Winner	p-value	t-statistic
Random Search vs ES	ES	0.0000	-22.37

Table 3: Statistical comparison between the two algorithms using Independent t-test.

5.3 Coevolution

In this section we present our results for the coevolution phase.

Hyperparameter To assess the suitability of parametric tests, we first applied the Shapiro–Wilk normality test to both combinations. The p-values obtained (0.3771 and 0.7805) were both greater than 0.05, indicating that the data followed a normal distribution. Thus, we then performed pairwise Independent t-tests to compare the performance of the two combinations. And the the difference between the combinations was not statistically significant ($p = 0.0558$, $t = -2.24$).

In this phase, we achieved a robot with 4.287 (GIF) of reward using our algorithm for the first map, with an average of 3.312 using the best combination. While in the second map the best was 1.936 (GIF) and the average 1.870. The following image and table present the parameters selected and the results.

#	Structure Mutation Rate
1	0.1
2	0.3

Fig. 5: Hyperparameter combination used in the Coev grid search.

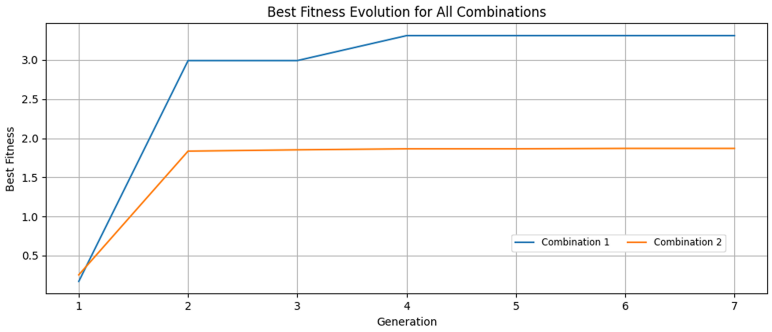

Fig. 6: Fitness evolution for each Coev hyperparameter combination.

Comparison Evolve Both and Random Search We performed the comparison by utilizing the same approach as before.

Method	p-value	Normality
Random Search	0.2199	Normal
Evolve Both	0.7805	Normal

Table 4: Normality test results using Shapiro-Wilk test.

Comparison	Winner	p-value	t-statistic
Random Search vs Coevolution	Coevolution	0.0010	-5.06

Table 5: Statistical comparison between the two algorithms.

6 Conclusion

This project applied evolutionary computation to continuous, discrete, and co-evolutionary problems, highlighting the importance of fitness design, operator choice, hyperparameter tuning, and proper statistical testing, while offering a realistic and enriching challenge.

7 Acknowledgment

Chatgpt was utilized for refactoring all the text of the report and also for creating code in general and especially for bug fixing.