CE Report

Filipe Obrist[2024170686]1

Departamento de Informática, Universidade de Coimbra, Portugal
uc2024170686@student.uc.pt

Abstract. This project investigates evolutionary approaches for optimizing soft robots, progressing through three key stages: evolving morphology, evolving control, and finally co-evolving both simultaneously. Starting with the independent optimization of structure and controller, various evolutionary strategies were tested and compared. The final phase integrates both components, exploring how their interaction impacts overall performance. The study highlights the challenges and opportunities of co-evolution in robotic design.

Keywords: Soft Robots · Evolutionary Algorithms · Co-Evolution.

1 Introduction

In this paper, I use evolutionary algorithms with the aim of solving the robotic issue of morphology and control co-design of soft robots. Using evolutionary principles, I can explore how varied structure-control policy pairs arise while evolving solutions to specific locomotion tasks. In order to achieve the ultimate task (co-operation), my investigation addresses the three progressively complex challenges: Morphology Optimization (2.2), Control Policy Optimization (2.3) and Co-Evolution (2.4).

2 Experiment

2.1 Experimental Protocol

All experiments in this section adhere to the following standardized protocol:

- Each algorithm configuration was executed 5 independent runs with fixed random seeds ([42, 43, 44, 45, 46]) to ensure reproducibility and enable statistical comparison.
- The fitness of solutions was determined exclusively by the environment reward returned by the Gym simulator, reflecting task performance (e.g., locomotion distance).
- Computational resources (e.g., number of evaluations, simulation steps) were held constant across all algorithm variants to ensure fair comparisons.
- For each algorithm, all the scenarios from the tasks where it was applied where runned.
- A **random search** algorithm was included as baseline comparison in all experiments, using the same evaluation budget and seeds as other methods.

2.2 Morphology Optimization

The robot morphology representation as a fixed-length array of integers constrained our choice of evolutionary approaches. This discrete representation inherently excludes algorithms like Differential Evolution (DE) and Covariance Matrix Adaptation ES (CMA-ES), which are designed for continuous optimization spaces. After careful consideration, we focus on three approaches:

Genetic Algorithm (GA) The GA was selected for its proven effectiveness in discrete optimization problems. Key advantages for our task include:

- Natural handling of integer-based representations through direct encoding
- Explicit maintenance of population diversity - crucial for exploring morphological configurations
- Robustness to local optima through tournament selection pressure

The variation operations used were point mutation and uniform crossover.

Evolution Strategy (ES) While typically applied to continuous domains, we adapted ES for several compelling reasons:

- Selection Mechanism Analysis: Unlike GA, ES uses a (μ+λ) survival strategy, where both parents and offspring compete for selection, enabling a different evolutionary pressure.
- Crossover Exclusion: To isolate the effect of recombination, ES excludes crossover entirely while maintaining a similar mutation operator, allowing a focused comparison on the role of selection and variation.

Variances of Genetic Algorithm Given GA's superior initial performance, I developed two targeted variants:

- **High-Mutation GA (HM-GA)**: Enhances exploration in the discrete configuration space.
- **Compact GA (C-GA)**: Maintains evaluation budget while testing intensified selection pressure.

As a curiosity, I also ran the original GA for the three different fixed controllers.

Parameters Table 1 summarizes the parameter configurations for each algorithm used in this task.

Table 1. Parameterization Task 3.1

Algorithm	Generations	Pop size	Mutation	Crossover	Elitism
Random	100	40	-	-	-
GA	100	40	0.1	0.5	0.2
ES	100	40	0.1	-	-
HM-GA	100	40	0.5	0.5	0.2
C-GA	200	20	0.1	0.5	0.2

Results I performed a Friedman test to determine if there were significant differences between the five algorithms across five matched runs. The test revealed a statistically significant difference (p = 0.004), indicating that at least one algorithm performed differently from the others. To identify which algorithms differed, we conducted pairwise Wilcoxon signed-rank tests, applying a Bonferroni correction ($(\alpha) = 0.005$). While several comparisons showed p-values suggesting potential trends (e.g., GA > ES, p = 0.0625), none reached statistical significance at the corrected threshold. This suggests that although the Friedman test detects overall differences, our current sample size (n = 5) may lack the power to resolve specific pairwise distinctions. Despite this, we observe meaningful differences between algorithms, as shown in Figure 1. The graph reveals that the GA with more generations consistently outperforms others. This was unexpected, as the trade-off involved reducing the population size per generation—something I assumed would hinder performance, but instead led to improved results. The graph also highlights the importance of crossover: the Evolution Strategy, which lacks it, performs slightly worse and shows less diversity, as indicated by consistently lower standard deviations. Lastly, we see that a high mutation rate degrades performance. At a mutation rate of 0.5, the process becomes nearly random—since such frequent, large changes prevent gradual improvement and disrupt any accumulated adaptation.

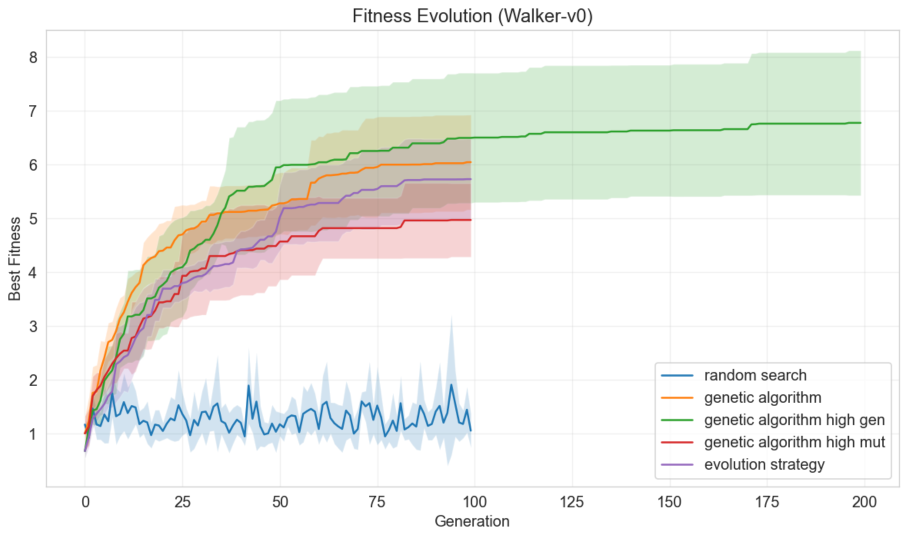

Fig. 1. Fitness evolution of the implemented algorithms in the Walker scenario

Figure 2 confirms that the best fixed controller was indeed the one used (*alternating_gait*).

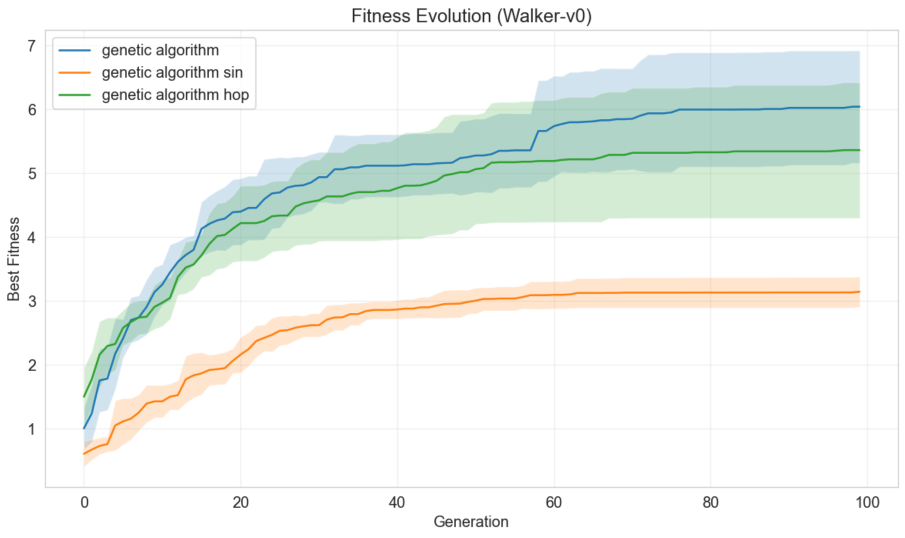

Fig. 2. Fitness evolution of the possible fixed controllers

As an additional observation, I preserved the final morphologies of the best-evolved robots from each algorithm. Interestingly, they all converged toward similar structural designs, suggesting common optimal solutions (Figure 3).

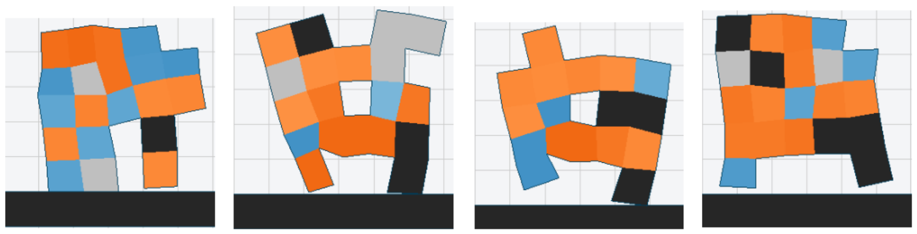

Fig. 3. Morphologies of the best robots found

Note: Results for the BridgeWalker scenario were omitted, as they mirrored those from the Walker scenario. This suggests that the two environments may not have differed enough to influence the evolutionary process meaningfully.

2.3 Control Policy Optimization

In this section, we evolve our neural controller by optimizing its weights. Since the representation now consists of floating-point values instead of integer arrays, we can explore different evolutionary algorithms. To maintain continuity, we bring over the best-performing algorithm from the structure evolution task-Genetic Algorithm. Alongside GA, we test Differential Evolution (DE), which uses geometrical mutation and binomial crossover, making it well-suited for continuous optimization due to its directional exploration and adaptability. We also explore CMA-ES, an evolutionary strategy that adapts a multivariate normal distribution over time, making it promising for fine-tuning in high-dimensional, non-linear landscapes. After observing performance issues in one scenario using DE—likely due to declining diversity as standard deviation shrank—I implemented a variant of DE (DE_Tuned) with increased mutation factor (F = 0.7) to boost exploration and reduced crossover rate (CR = 0.5) to limit disruptive changes in noisy environments. *Note: Genetic Programming (GP) was initially considered as a potential approach. However, preliminary investigation revealed it would be computationally expensive for our specific application while offering no significant performance advantages, as also observed in similar VSR control studies [1]*

Parameters Table 2 summarizes the parameter configurations for each algorithm used in this task.

Table 2. Parameterization Task 3.2. IS - Initial Sigma; F - Mutation Scaling Factor; CR - Crossover rate (Binomial)

Algorithm	Generations	Pop size	Mut_std	Cross	Elitism	F	CR	IS
Random	100	40	-	-	-	-	-	-
GA	100	40	0.1	0.5	0.2	-	-	-
DE	100	40	-	-	-	0.5	0.7	-
DE_Tuned	100	40	-	-	-	0.7	0.5	-
CMA-ES	100	40	-	-	0.2	-	-	0.3

Results As in the previous task, a statistically significant difference was found among the five algorithms ($p = 0.014$), although no pairwise comparison reached statistical significance. Still, notable performance trends emerged, as illustrated in Figure 4. In the first scenario (DownStepper), the Genetic Algorithm (GA) once again outperformed the others, with Differential Evolution (DE) following closely behind. This was somewhat surprising, as DE is typically considered more suitable for continuous optimization problems like neural weight evolution due to its geometric mutation and smooth exploration capabilities. Nevertheless, GA's robust selection and crossover mechanisms proved highly effective in this context. On the other hand, the CMA-ES approach underperformed significantly—even trailing behind the random baseline. This unexpected result is likely due to implementation or parameterization issues, such as a poor initial step size or premature convergence due to high noise sensitivity. Given this, CMA-ES was excluded from the next scenario.

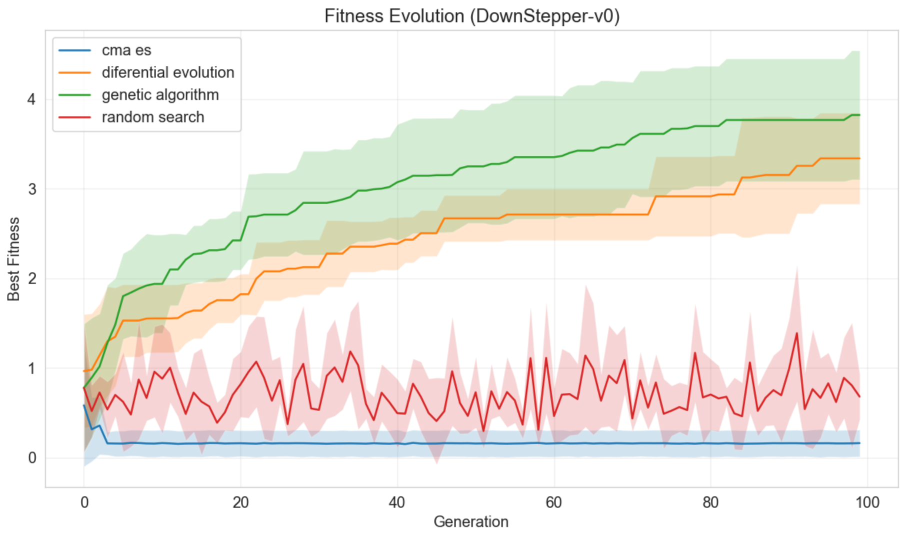

Fig. 4. Fitness evolution of the implemented algorithms in the DownStepper scenario

In the ObstacleTraverser scenario, GA continued to dominate, consistently outperforming both DE and the tuned variant (DE_Tuned), as shown in Figure 5. Although the DE_Tuned version was designed to promote greater exploration by increasing the mutation factor and reducing crossover pressure, the improvements were marginal. This suggests that simply adjusting DE's hyperparameters may not be sufficient in highly dynamic or noisy environments where strong selection and recombination play a more dominant role.

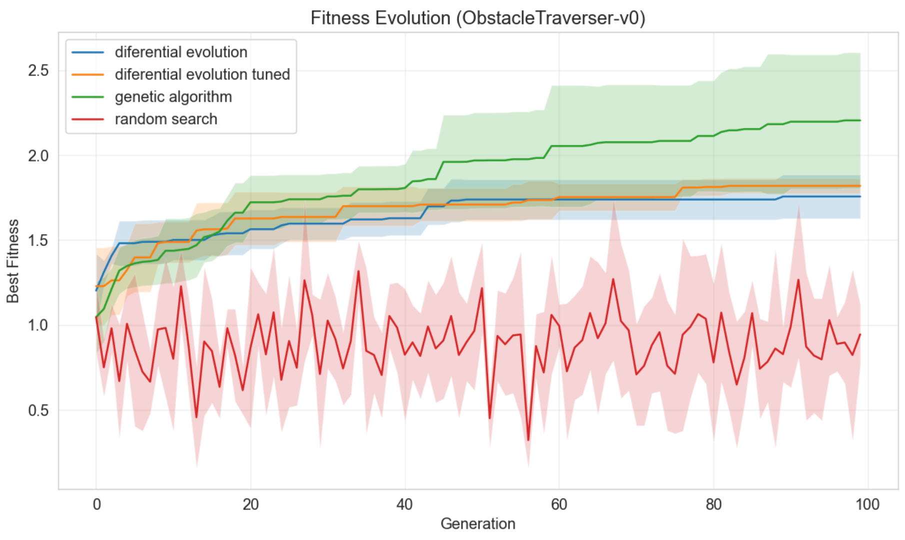

Fig. 5. Fitness evolution of the implemented algorithms in the Obstacle scenario

2.4 Co-Evolution

Finally, we arrive at the last phase of this project: co-evolution. Co-evolution of morphology and control has been shown to unlock more adaptive and functional robot designs [2], by allowing both dimensions to adapt in response to each other. The general framework is straightforward: evolve a population of structures, evolve a population of controllers, pair them, evaluate them, and use their combined performance to guide evolution. In this task, the main focus was on exploring different pairing strategies. The first approach implemented was Random Pairing Co-Evolution, where each structure is randomly paired with a controller. Each pair is evaluated together, and the top-performing combinations are preserved for the next generation. The remaining individuals are used to generate offspring through variation operators. The second approach was Alternating Co-Evolution, where the populations are evolved in turns: one generation focuses on evolving controllers while keeping the structures fixed, and the next focuses on evolving structures while keeping the controllers fixed. This strategy promotes stability and allows each population to adapt to the current state of the other. It's particularly effective when one of the domains is harder to optimize than the other. Given that in our case the controllers are significantly harder to evolve than the structures, I developed a biased variation of the alternating strategy, giving controllers more evolutionary time (in a 1:3 ratio). This bias allows the more complex domain (controllers) to converge better while still allowing structural innovation. Lastly, I implemented the Best-of-Opposite Co-Evolution approach. Though computationally demanding, this method attempts to maximize pairing effectiveness by evaluating each individual against the best member of the opposite population. Due to its high evaluation cost, it was only feasible to test this method in the GapJumper scenario. Nevertheless, this approach is designed to promote strong coordination between structure and controller by always optimizing against the current best partner. In all approaches above, Genetic Algorithms (GA) were used to evolve the structures, while Differential Evolution (DE) was used for the controllers. While the previous tasks favored GA for both domains, I opted to diversify the methodology to explore DE's potential in continuous parameter optimization.

Note: Other pairing strategies, such as K-Random Pairing, were explored during development, but due to time constraints, they are not included in this report.

Parameters The parameters in this task remain consistent across all approaches, with a population size of 40, 100 generations, and an elitism rate of 0.2. However, we can still differentiate the methods by reporting their time and number of evaluations per generation, as summarized in Table 3.

Table 3. Times and evaluations numbers in task 3.3

Algorithm	Time	Evaluations per generation
Random	Fast	Pop_size
Alternating	Medium	Pop_size
Bias-Alternating	Medium	Pop_size
Best-Of-Opposite	Very Slow	4 x Pop_size

Results The overall results were somewhat disappointing; however, several meaningful conclusions can still be drawn. In this case, no statistically significant differences were found among the algorithms (($p = 0.095$) for GapJumper and ($p = 0.074$) for CaveCrawler). Nevertheless, clear trends and performance distinctions can be observed in Figure 6. In the GapJumper scenario, the expected outcome occurred: the Best-of-Opposite approach consistently outperformed all other algorithms across generations. Interestingly, the Alternating_Bias approach was a very close second. Considering that Best-of-Opposite is by far the most computationally and time-intensive method, Alternating_Bias could arguably be seen as the more practical dominant algorithm, depending on the evaluation criteria. This also highlights the importance of allocating more generations to evolving controllers rather than structures. In Figure 6, we can clearly observe the performance gap between the standard Alternating approach and its biased variation. Unexpectedly, Random Pairing also delivered strong results, slightly outperforming the standard Alternating method. While overall fitness values remained relatively low, suggesting that no approach consistently achieved the jump in the GapJumper task, it's reasonable to believe that extending the number of generations could eventually yield stronger solutions.

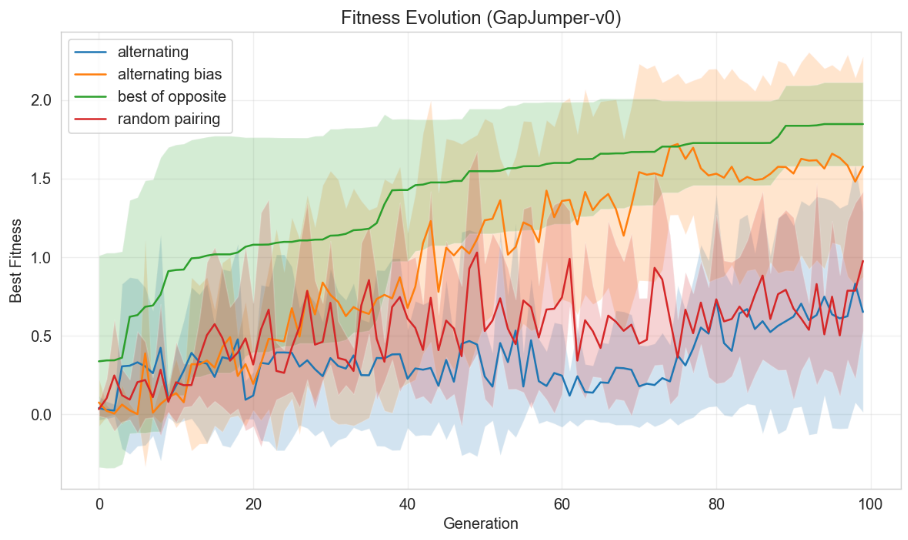

Fig. 6. Fitness evolution of the implemented algorithms in the GapJumper scenario

The CaveCrawler scenario produced similar trends, as shown in Figure 7. While the Best-of-Opposite approach was not included due to its computational demands, we can reasonably assume it would exhibit similar superiority to what was observed in the first scenario.

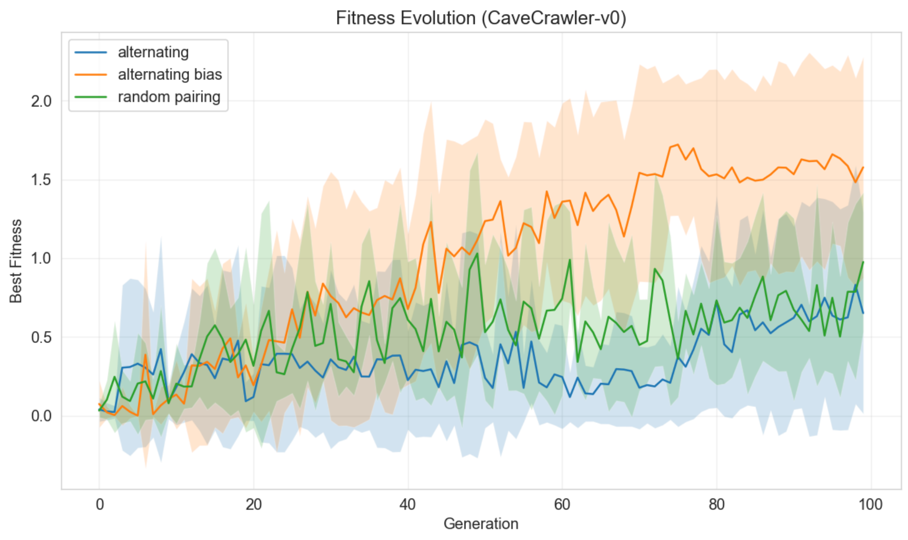

Fig. 7. Fitness evolution of the implemented algorithms in the CaveCrawler scenario

2.5 Conclusion

This project explored the evolution of soft robots in three progressive stages: structure evolution, controller optimization, and full coevolution. Each phase increased in complexity, highlighting the strengths of different evolutionary strategies. Genetic Algorithms proved reliable for discrete morphology, while Differential Evolution showed potential in continuous weight optimization. In coevolution, pairing strategies played a crucial role—Alternating Bias stood out as a strong balance between performance and efficiency. Future work could involve increasing the number of runs for stronger statistical support, adapting fitness shaping to task goals, and extending the number of generations to allow further convergence. Exploring hybrid pairing and adaptive evolutionary parameters may also yield improved results.

References

1. Scala, F., Medvet, E.: Genetic Programming for Teaching and Learning Control in Soft Robots. In: Genetic Programming Theory and Practice XX, Springer (2023)
2. Cheney, N., MacCurdy, R., Clune, J., Lipson, H.: Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Simulator. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 167–174. ACM (2013)