Survival of the Softest: Evolving Adaptive Soft Robots

Martinus Boom[1][20241544644]

University of Coimbra, Coimbra, Portugal uc2024154464@student.uc.pt

Abstract. The complete progress of the project "Survival of the Softest," which is a component of the University of Coimbra's Evolutionary Computation course, is presented in this report. This work aims to investigate the use of evolutionary computation techniques for the simulation-based optimization of soft robotic agents. The project is broken down into three stages: (1) using a fixed controller to evolve the physical structure of soft robots, (2) using a fixed robot morphology to evolve the controller, and (3) co-evolving structure and control at the same time. Several bio-inspired algorithms were implemented throughout the phases: Genetic Algorithms (GA) were studied in Phase 1, Evolutionary Strategies (ES) and Differential Evolution (DE) were studied in Phase 2, and DE was ultimately chosen for further development. Phase 3 combined two distinct populations in a co-evolutionary manner. Experimental setups and an analysis of the results demonstrate how well evolutionary approaches optimize soft robots.

Keywords: Soft Robotics · Evolutionary Algorithms · Genetic Algorithm · Differential Evolution · Co-evolution · Neural Controller · EvoGym

1 Introduction

Soft robotics uses flexible, deformable materials to provide a variety of adaptive locomotion solutions, so they are more versatile than traditional rigid robots, especially in dynamic environments and challenging terrain. A strong framework for modeling and developing soft robots made up of different voxel types, each with unique physical traits and behaviors, is offered by the EvoGym platform.

The outline of the three progressive stages of the evolutionary design process is presented in this report. In the first stage, the robot's structure is changed while the controller remains fixed. The development of a neural network-based controller given a fixed robot morphology is the subject of the second phase. Finally, both elements are combined in a co-evolutionary setup in the third phase. In order to test various facets of robotic adaptation, each phase focuses on particular locomotion tasks within EvoGym.

2 Methodology

2.1 Phase 1: Evolving Structure with Fixed Controller

I decided to focus on the Genetic Algorithm (GA) and explore its functionality through the application for robot structure evolution.

A structure consists of a 5×5 voxel grid where each cell holds one of four voxel types (empty, rigid, soft, actuator). The GA kicks off with an initial population of correct structures which must be connected because disconnected designs get eliminated and are replaced.

The algorithm executes the following operations during every generation: All individuals undergo fitness evaluation and the algorithm preserves the best results through the process of elitism. The creation of offspring occurs through parent selection which leads to crossover and mutation processes. Any structure that becomes disconnected must be eliminated and duplicate structures need to be taken out to sustain population diversity. Incomplete populations receive additional valid structures which are generated to occupy the remaining available slots.

The evaluation of fitness takes place after the new population formation. A better structure gets selected to replace the current best when found. The process advances through a predetermined number of generations while recording the time-based data for both best and average fitness.

Phase 1.1: Controller Selection The aim here was to identify the most suitable predefined controller (`Alternating gait`, `Sinusoidal wave`, or `Hopping motion`) for each scenario (`Walker-v0` and `BridgeWalker-v0`) using a GA with fixed parameters: population size 30, 50 generations, one-point crossover, flip mutation, tournament selection, crossover rate 0.8, mutation rate 0.2, and 10

The controller with the highest average best fitness (across the fixed seeds) was selected per scenario and a stagnation point was also recorded.

Phase 1.2: GA Operator Combination Testing With the best controller fixed for each scenario, a full grid search was conducted over all combinations of crossover, mutation, and selection operators. The options tested were:
Crossover operators:

- *One-point*: a point is randomly chosen on the structure grid, and all voxels up to that point are taken from one parent, the rest from the other.
- *Two-point*: two random indices are chosen on the flattened structure. The segment between them is copied from one parent, and the rest from the other.
- *Uniform*: each voxel is chosen independently from either parent with 50% probability.

Mutation operators:

- *Flip*: each voxel is inverted — actuator becomes empty and empty becomes actuator or random valid voxel.
- *Swap*: two random positions in the structure are selected and their values exchanged.
- *Scramble*: a random segment in the flattened structure is selected, and its values are randomly shuffled.

Selection methods:

- *Tournament*: a fixed-size group is randomly sampled from the population, and the individual with highest fitness in that group is selected.
- *Roulette*: selection is probabilistic and proportional to fitness. The probability of choosing individual i is:

$$P_i = \frac{f_i - f_{\min}}{\sum_j (f_j - f_{\min})} \tag{1}$$

where f_i is the fitness of individual i, and $f_{\min}$ is subtracted to ensure non-negative values.

Population size, Crossover, mutation, and elitism rates were kept the same as in phase 1.1.

Phase 1.3: Statistical Analysis The results from Phase 1.2 were used in this step. For each scenario, the best fitness value obtained per seed and operator combination was collected. The goal was to determine whether differences between combinations were statistically significant and, for each scenario, to select the configuration with the highest mean fitness if its superiority was confirmed by the statistical tests.

Phase 1.4: Hyperparameter Tuning Finally, a grid search was performed using only the best operator configuration found in Phase 1.3. The parameters tested were:

- Crossover rate: 0.6, 0.8, 1.0
- Mutation rate: 0.1, 0.2, 0.3
- Elitism rate: 0.05, 0.1, 0.2

As in phase 1.3, the results provided from this hyperparameter tuning phase where also used for Statistical Analysis.

2.2 Phase 2: Evolving Controller with Fixed Structure

In this phase, the robot structure was fixed to the one provided with the project and only the neural controller was evolved. Each controller consisted of a fully connected neural network with real-valued weights.

The choice was made to implement and compare five optimization approaches for evolving the controller's weights: Random Search (**RS**), Evolutionary Strategies (**ES**), Enhanced Evolutionary Strategies (**ES+**), Differential Evolution **DE/rand/1/bin**, and Differential Evolution **DE/best/1/bin**.

Phase 2.1: Algorithm Comparison Each algorithm was implemented as follows:

- **Random Search (RS)**: In each generation, a new solution (weight vector) was sampled from a standard normal distribution. No selection or memory was used: only the current sampled individual was evaluated and compared.
- **Evolutionary Strategies (ES)**: A (μ, λ) strategy was adopted. A population of μ parents was maintained. In each generation, λ offspring were generated by applying *Gaussian mutation* to randomly selected parents:

  $$w' = w + \mathcal{N}(0, \sigma^2) \tag{2}$$

  Each weight in the vector had a fixed probability of being mutated. The next generation was formed by selecting the top μ offspring. The standard deviation σ was adapted using a multiplicative rule: if fitness improved, σ was decreased (exploitation); otherwise, it was increased (exploration).
- **Enhanced Evolutionary Strategies (ES+)**: Similar to ES, but using a $(\mu + \lambda)$ strategy. After generating offspring, both parents and offspring were pooled together, and the best μ individuals were selected for the next generation. This promoted stronger elitism.
- **DE/rand/1/bin**: This is a Differential Evolution strategy where, for each target vector, three distinct individuals x_1, x_2, x_3 were sampled at random from the population. A mutant vector was computed as:

  $$v = x_1 + F \cdot (x_2 - x_3) \tag{3}$$

  where F is a fixed mutation factor. The mutant and target vectors then underwent *binomial crossover*, where each gene had a probability CR of being inherited from the mutant.
- **DE/best/1/bin**: Like DE/rand/1/bin, but using the current best individual x_{best} as the base vector instead of x_1. The mutation strategy became:

  $$v = x_{\text{best}} + F \cdot (x_2 - x_3) \tag{4}$$

  Binomial crossover was then applied with the target vector.

The fitness of the best individual was recorded per generation. After all seeds, the final fitness values were used for statistical comparison.

Phase 2.2: Statistical Analysis The fitness results collected in Phase 2.1 were used to compare the algorithms statistically for each scenario (`DownStepper-v0` and `ObstacleTraverser-v0`), and the algorithm with the highest mean fitness was selected when supported by statistical significance.

Phase 2.3: Hyperparameter Tuning (F and CR) A grid search was performed on the Differential Evolution variants selected in Phase 2.2. The parameters tested were:

- Scaling factor F: 0.3, 0.5, 0.7
- Crossover rate CR: 0.5, 0.7, 0.9

Given that **DE** consistently outperformed the other algorithms in both scenarios during Phase 2.2 (results shown later), two additional variants were investigated for DE/rand/1/bin:

- **DE/rand/2/bin**: A variant where five individuals x_1, x_2, x_3, x_4, x_5 were sampled and mutation was done using:

$$v = x_1 + F \cdot (x_2 - x_3) + F \cdot (x_4 - x_5) \tag{5}$$

  This variant introduces more diversity during exploration.
- **DE/jDE**: A self-adaptive DE variant where each individual carried its own mutation factor F and crossover rate CR. These parameters were evolved over time using a learning rate τ:

$$F \leftarrow \begin{cases} U(0.1, 1.0) & \text{with probability } \tau_1 \\ F & \text{otherwise} \end{cases} \tag{6}$$

$$CR \leftarrow \begin{cases} U(0.0, 1.0) & \text{with probability } \tau_2 \\ CR & \text{otherwise} \end{cases} \tag{7}$$

  This allowed DE to adapt its parameters automatically during evolution.

Additionally, one more variant was explored for DE/best/1/bin:

- **DE/best/2/bin**: A more aggressive variant, where four individuals x_1, x_2, x_3, x_4 were sampled and the mutation rule was:

$$v = x_{\text{best}} + F \cdot (x_1 - x_2 + x_3 - x_4) \tag{8}$$

  This allowed more information to be combined in the mutation.

Each configuration was tested across all seeds. The final fitness of each run was recorded and compared statistically as in Phase 2.2.

Phase 2.4: Final Tuning of `POP_SIZE` and `NUM_GENERATIONS` In this final phase, the Differential Evolution variant, as well as its F and CR values, were fixed based on the statistical results from Phase 2.3. With those settings defined for each scenario, a grid search was performed over different population sizes and generation counts.

- `POP_SIZE`: 30, 50, 70
- `NUM_GENERATIONS`: 20, 40, 60

Each combination was evaluated for every seed. The final best fitness per configuration was extracted and statistically compared. The configuration with the highest average fitness was selected.

2.3 Phase 3: Co-Evolution of Structure and Controller

Finally, in this last phase, both the robot structure and controller are evolved together using a step-based Co-Evolutionary Algorithm. Unlike earlier phases, where only one component was optimized at a time, this approach updates both in alternating steps.

Each individual is a pair: one structure and one controller, as defined in Phases 1 and 2. Two separate populations are used, one for structures and one for controllers. The evaluation is done by testing all structure-controller pairs, forming a fitness matrix.

Each generation follows these steps:

1. **Fitness Evaluation:** All structure-controller pairs are evaluated, the best structure and controller are selected according to row and column fitness, and the best-performing pair is stored.
2. **Elitism:** A portion of the top individuals in each population is kept.
3. **Phase-based Evolution:** The algorithm alternates between controller and structure updates:
   - In the **controller phase**, structures are fixed. Controllers evolve using tournament selection, uniform crossover that mixes weights layer-wise using a random mask while ensuring shape compatibility through padding or cropping, and Gaussian mutation applied to selected parameters.
   - In the **structure phase**, controllers are fixed. Structures evolve using tournament selection, uniform crossover on voxel grids, and scramble mutation.
4. **Compatibility:** Controllers are adjusted to match new structures.
5. **Population Maintenance:** Disconnected or duplicate individuals are removed, and new valid individuals are added.

This process allows both components to evolve together while maintaining compatibility.

Phase 3.1: Co-Evolution Execution This step runs the Co-Evolutionary Algorithm described earlier for each scenario and seed. A population of 30 structures and 30 controllers is initialized and evolved over 50 generations using a crossover rate of 0.8, mutation rate of 0.3, elitism rate of 0.1, and a tournament size of 5, while recording the best fitness per generation, the cumulative best fitness, and the final best structure-controller pair found.

Phase 3.2: Hyperparameter Tuning For this subphase it´s performed a grid search over combinations of the main hyperparameters used in the coevolution algorithm:

- Mutation rate: 0.1, 0.2, 0.4
- Crossover rate: 0.6, 0.8, 1.0

These runs reuse the full coevolution logic and differ only by adjusting global hyperparameters before execution.

Phase 3.3: Statistical Analysis At this point, the goal was to determine whether the hyperparameter choices in Phase 3.2 had a statistically significant impact on final fitness. The configuration with the best average performance was then selected and used as the basis for future experiments and reference setups.

3 Experimental Setup

To ensure a consistent and fair evaluation of all algorithmic variants, the following experimental protocol was adopted.

Reproducibility All experiments across Phases 1, 2, and 3 were executed using a fixed set of five random seeds: `[42, 123, 999, 2025, 7]`.

Experimental Protocol

- **Same computational budget**: number of evaluations is identical across all comparisons.
- **Multiple runs**: each configuration was run 5 times using the fixed seeds.
- **Diverse initial population**: all initializations were randomized but constrained for feasibility.
- **Logging:** For every run and generation, detailed logs were stored in structured CSV files. These include individual and population fitness values, parameter configurations, statistical results, and summaries for each phase and scenario. The output is organized by phase folders and seeds, with consistent naming conventions to support later analysis and reproducibility.

Statistical Measures The experimental analysis includes the following:

- **Performance metrics:** Final fitness values were used to evaluate the effectiveness of each configuration.
- **Significance testing:** Statistical tests were applied to determine whether the observed differences between algorithms or configurations were significant. Since the outcome variable (fitness) is continuous and the predictor variable (algorithm or configuration) is categorical with more than two independent groups, the testing approach followed standard methodology:
  1. First, the **Shapiro-Wilk test** was applied to each group to assess normality.
  2. If all groups followed a normal distribution, a **one-way ANOVA** was used.
  3. If normality was violated in any group, the non-parametric **Kruskal-Wallis test** was applied instead.

4 Results and Discussion

All plots and result visualizations referred to in this section are available in the Appendix (see Figures 1–30).

4.1 Phase 1: Structural Evolution

Controller Selection (Phase 1.1) Figure 1 and Figure 2 show the final fitness per controller, where in `Walker-v0` `Hopping` had the highest mean followed by `Alternating` and `Sinusoidal`, and in `BridgeWalker-v0` `Hopping` and `Alternating` were close with `Sinusoidal` again lower. Figures 3 and 4 show the progression over generations, with `Hopping` leading early in `Walker-v0` and `Alternating` overtaking `Hopping` later in `BridgeWalker-v0`, where stagnation occurred around generation 30–40.

GA Operator Combination Testing (Phase 1.2) Figures 5 and 6 show the final fitness for each operator combination, with several configurations in `Walker-v0` exceeding fitness 4.5 while most in `BridgeWalker-v0` stayed below 2.0. The corresponding curves in Figures 7 and 8 show early convergence in some configurations and more gradual improvement in others, with clearer differences in `Walker-v0`. The best configuration based on the highest mean fitness was `uniform crossover`, `swap mutation`, and `tournament selection` for `Walker-v0`, and `uniform crossover`, `scramble mutation`, and `roulette selection` for `BridgeWalker-v0`.

Hyperparameter Tuning (Phase 1.4) Figures 9 and 10 show the final fitness across hyperparameter settings with visible variation in both scenarios. Figures 11 and 12 show that some configurations stagnated early while others continued to improve across generations with consistent performance across seeds. The best hyperparameters based on the highest mean fitness for `Walker-v0` were crossover rate = 1.0, mutation rate = 0.3, and elitism rate = 0.1, while for `BridgeWalker-v0` the best were crossover rate = 1.0, mutation rate = 0.1, and elitism rate = 0.1.

Statistical Significance No statistically significant differences were found in either phase or scenario. In Phase 1.2, the Kruskal-Wallis test gave $p = 0.382$ for `Walker-v0` and $p = 0.555$ for `BridgeWalker-v0`. In Phase 1.4, ANOVA gave $p = 0.270$ for `Walker-v0` and the Kruskal-Wallis test gave $p = 0.950$ for `BridgeWalker-v0`. Although the best configurations in each case showed higher mean fitness across seeds, the differences were not statistically significant.

4.2 Phase 2: Controller Evolution

Algorithm Comparison (Phase 2.1) Figure 13 and Figure 14 show the final fitness distribution across all algorithms, where `DownStepper-v0` favored DE variants while in `ObstacleTraverser-v0` the differences between DE and ES methods were less pronounced. Figures 15 and 16 show the fitness evolution excluding RS, with DE methods showing faster improvement and higher final values while ES and ES+ progressed more slowly.

Hyperparameter Tuning (Phase 2.3) Figures 17 to 20 show how different F and CR values influenced the final fitness. The best configuration based on highest mean fitness was `DE_best2` with $F = 0.3$ and $CR = 0.5$ for `DownStepper-v0`, and `DE_rand` with $F = 0.5$ and $CR = 0.9$ for `ObstacleTraverser-v0`. Figures 21 to 24 show the fitness evolution across generations, with most configurations improving early and few continuing later.

Population and Generation Tuning (Phase 2.4) Figures 25 and 26 show the final fitness obtained using different combinations of population size and generation count. The best configuration based on highest final fitness was `DE_best2` with population = 50 and generations = 60 for `DownStepper-v0`, and `DE_rand` with population = 70 and generations = 60 for `ObstacleTraverser-v0`, using the F and CR values selected in Phase 2.3. Figures 27 and 28 confirm the trend, showing stronger and more consistent improvement when using larger budgets.

Statistical Significance The analysis in Phase 2.2 (based on the results from Phase 2.1) found statistically significant differences between algorithms, with `DE_best` and `DE_rand` achieving the highest mean fitness for `DownStepper-v0` and `ObstacleTraverser-v0`, respectively. The Kruskal-Wallis test yielded $p = 0.0127$ and $p = 0.0140$, confirming the differences were significant.

In 2.3, statistical significance was found within each DE variant. For `DownStepper-v0`, the ANOVA test on `DE_best2` gave $p = 0.037$, and for `ObstacleTraverser-v0`, the Kruskal-Wallis test on `DE_rand` gave $p = 0.041$, both below the 0.05 threshold. This supports that the configurations with highest mean fitness were also statistically better than the alternatives.

In 2.4, significance was also confirmed in both scenarios. For `DownStepper-v0`, ANOVA gave $p = 0.0047$, and for `ObstacleTraverser-v0`, the Kruskal-Wallis test gave $p = 0.0273$, indicating that population size and number of generations had a significant impact on final fitness.

4.3 Phase 3: Co-evolution

Co-Evolution Execution (Phase 3.1) Figure 29 shows the final fitness values per seed in the `GapJumper-v0` and `CaveCrawler-v0` scenarios. In `GapJumper-v0`, results were more consistent across seeds, while in `CaveCrawler-v0` the variation was wider, indicating less stable performance. Figure 30 shows the evolution of mean fitness over generations in both scenarios, with `GapJumper-v0` achieving higher fitness and faster convergence. Most improvement occurred in the first 30 generations, with the curve flattening after that point in both environments. This suggests that the alternation between structure and controller evolution was effective early, but further progress may require additional diversity or tuning.

Hyperparameter Tuning (Phase 3.2) This phase was implemented to test combinations of mutation and crossover rates in co-evolution, but only a subset of runs was completed. The data was incomplete and not consistent across seeds, so Phase 3.3 (Statistical Analysis) could not be done, and this likely contributed to the weak results seen in Phase 3.1.

Conclusion

A selection of operators and parameters during Phase 1 resulted in few configurations that produced better results, yet statistical analysis did not identify any meaningful distinctions. The evaluation process established a method to assess structures while monitoring performance during multiple runs.

The outcome of Phase 2 revealed significantly different results. The application of DE -based methods resulted in better fitness outcomes through the successful tuning of configuration parameters which included F and CR as well as population size and number of generations. The differences between scenarios received statistical confirmation through significance tests.

Phase 3 showed restricted performance results. The execution of hyperparameter tuning remained incomplete therefore statistical analysis became impossible. Early stagnation happened frequently during the experiment which resulted in low final fitness values that hindered proper conclusions.

The upcoming research activities need to finish the unexecuted co-evolution experiments and test every possible combination. The solution to stagnation and the improvement of joint evolution will benefit from enhanced diversity alongside modifications to the update cycle.

References

1. DataCamp: Genetic Algorithm Tutorial in Python. `https://www.datacamp.com/tutorial/genetic-algorithm-python`. Last accessed 11 May 2025
2. GeeksforGeeks: Crossover in Genetic Algorithm. `https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/`. Last accessed 11 May 2025
3. Cratecode: Roulette Wheel Selection Explained. `https://cratecode.com/info/roulette-wheel-selection`. Last accessed 11 May 2025
4. GeeksforGeeks: Gaussian Fit in Python. `https://www.geeksforgeeks.org/python-gaussian-fit/`. Last accessed 11 May 2025
5. Brownlee, J.: Evolution Strategies from Scratch in Python. `https://machinelearningmastery.com/evolution-strategies-from-scratch-in-python/`. Last accessed 11 May 2025
6. Brownlee, J.: Differential Evolution from Scratch in Python. `https://machinelearningmastery.com/differential-evolution-from-scratch-in-python/`. Last accessed 11 May 2025
7. Algorithm Afternoon: Self-Adaptive Differential Evolution. `https://algorithmafternoon.com/differential/self_adaptive_differential_evolution/`. Last accessed 11 May 2025

A Appendix

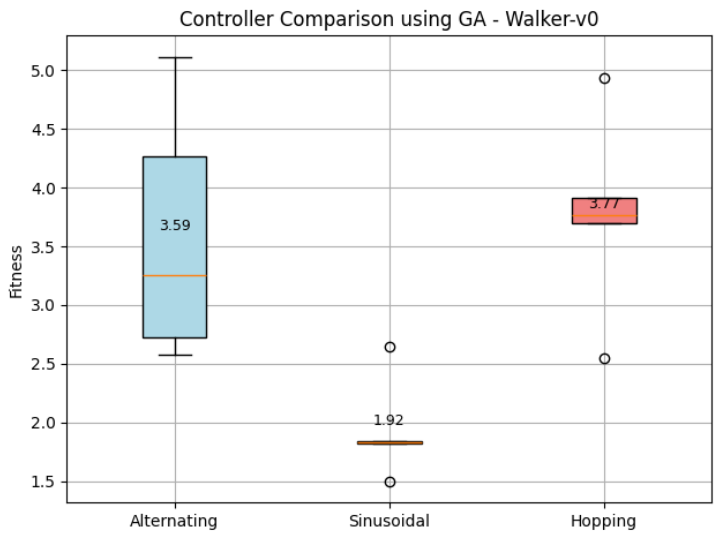

Fig. 1. Controller fitness distribution using GA for `Walker-v0`. (Phase 1.1)

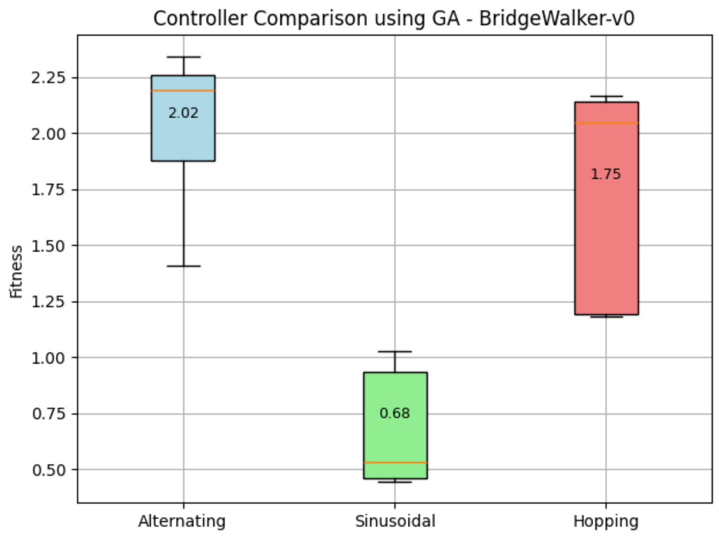

Fig. 2. Controller fitness distribution using GA for `BridgeWalker-v0`. (Phase 1.1)

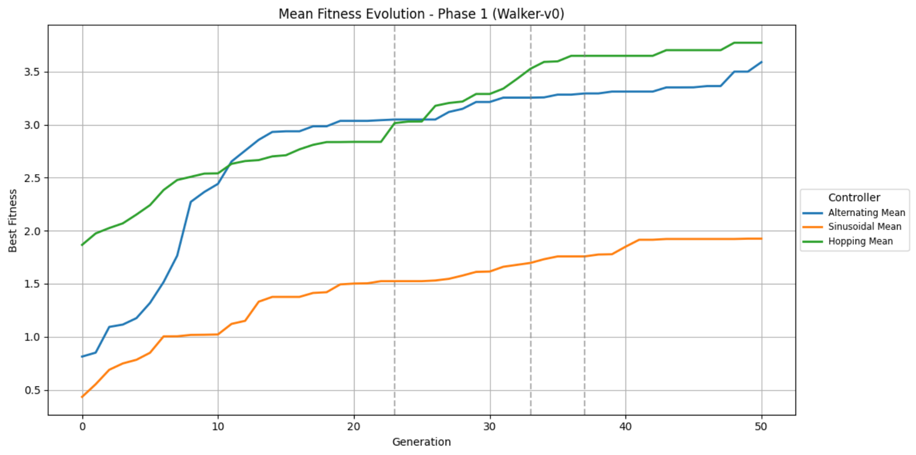

Fig. 3. Mean fitness evolution over generations for all controllers on `Walker-v0`. (Phase 1.1)

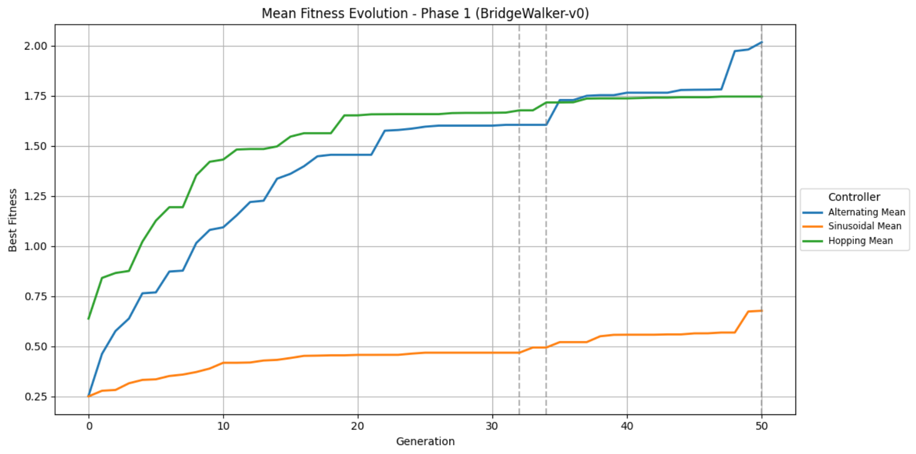

Fig. 4. Mean fitness evolution over generations for all controllers on `BridgeWalker-v0`. (Phase 1.1)

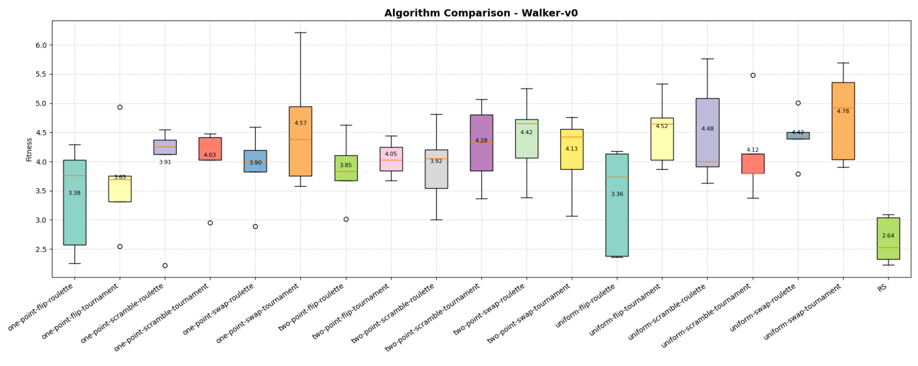

Fig. 5. Final fitness comparison of GA operator combinations on `Walker-v0`. (Phase 1.2)

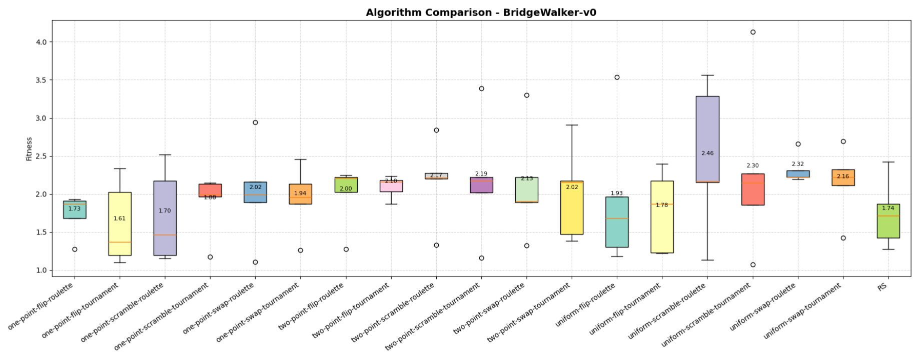

Fig. 6. Final fitness comparison of GA operator combinations on `BridgeWalker-v0`. (Phase 1.2)

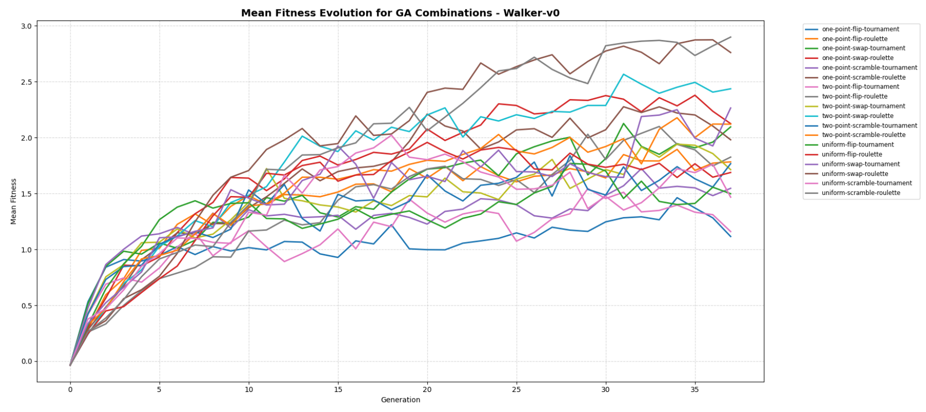

Fig. 7. Mean fitness evolution for all GA operator combinations on `Walker-v0`. (Phase 1.2)

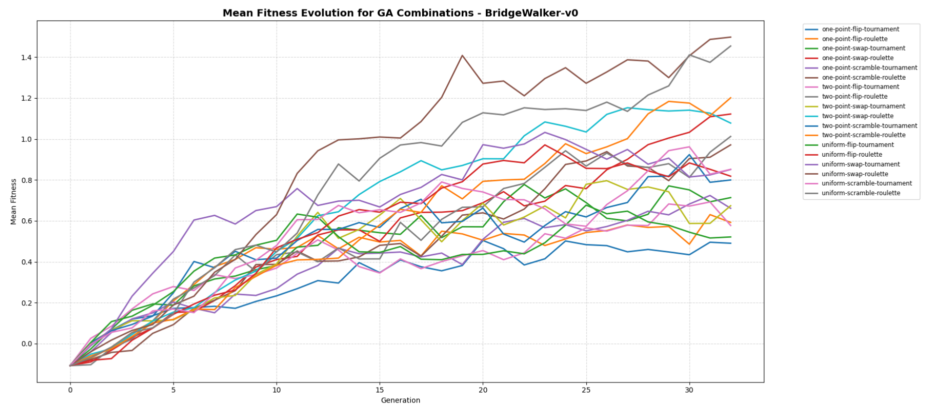

Fig. 8. Mean fitness evolution for all GA operator combinations on `BridgeWalker-v0`. (Phase 1.2)

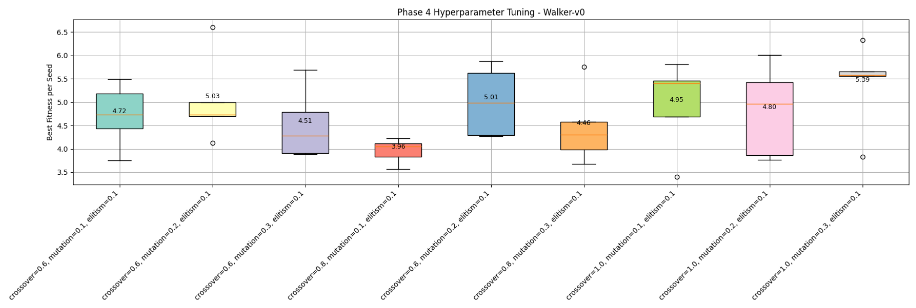

Fig. 9. Final fitness comparison of hyperparameter settings for `Walker-v0`. (Phase 1.4)

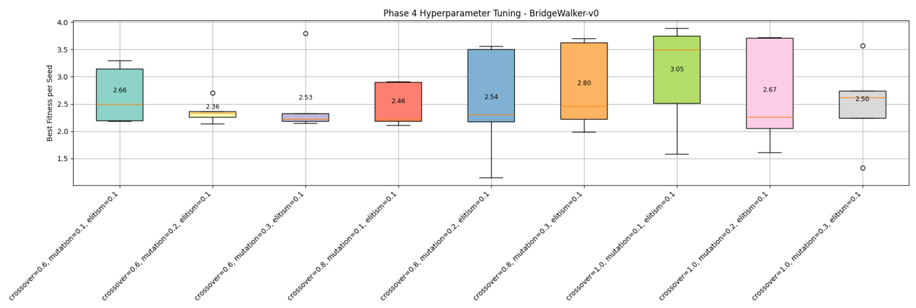

Fig. 10. Final fitness comparison of hyperparameter settings for `BridgeWalker-v0`. (Phase 1.4)

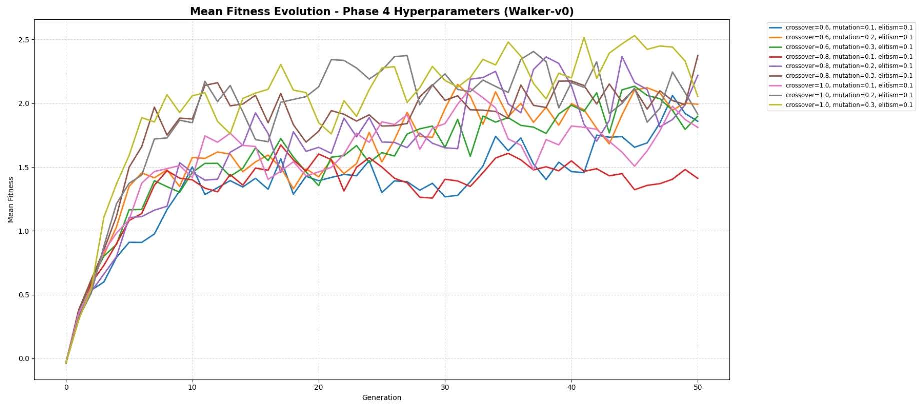

Fig. 11. Mean fitness evolution for different hyperparameter combinations on `Walker-v0`. (Phase 1.4)

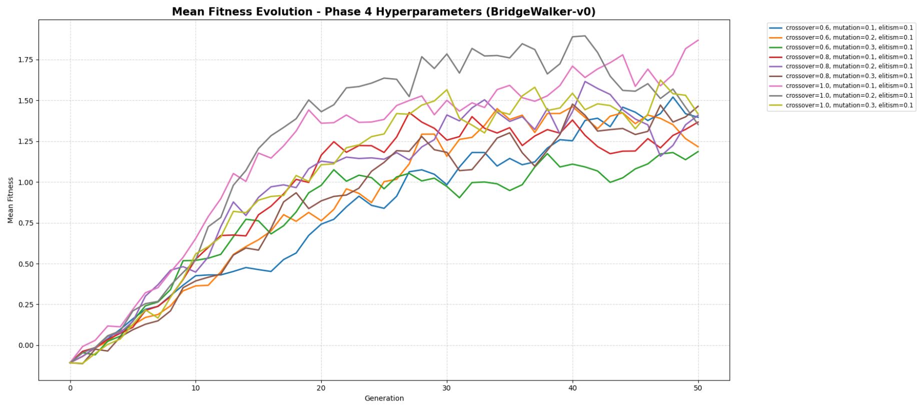

Fig. 12. Mean fitness evolution for different hyperparameter combinations on `BridgeWalker-v0`. (Phase 1.4)

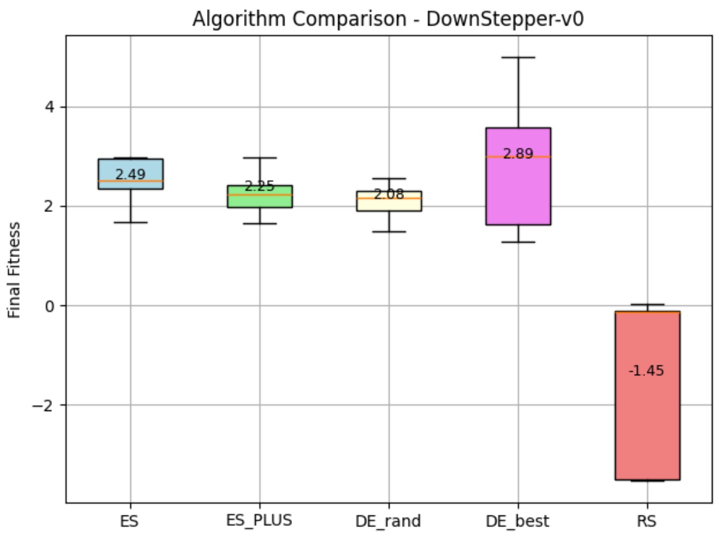

Fig. 13. Fitness distribution of all controller optimization algorithms on `DownStepper-v0`. (Phase 2.1)

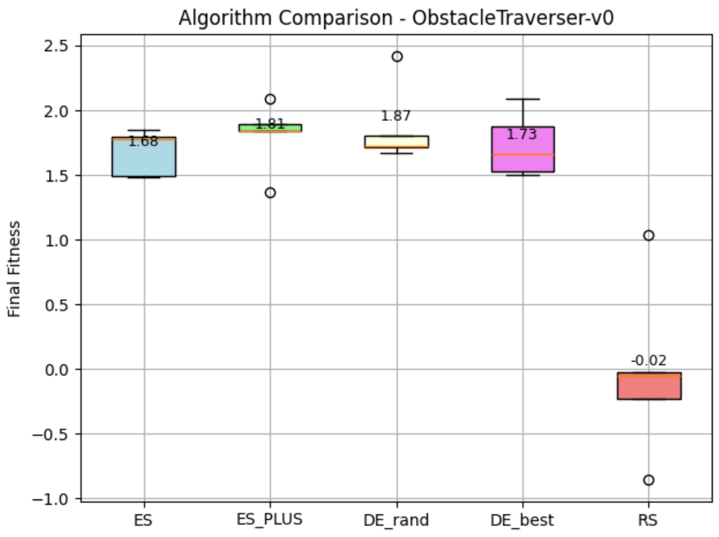

Fig. 14. Fitness distribution of all controller optimization algorithms on `ObstacleTraverser-v0`. (Phase 2.1)

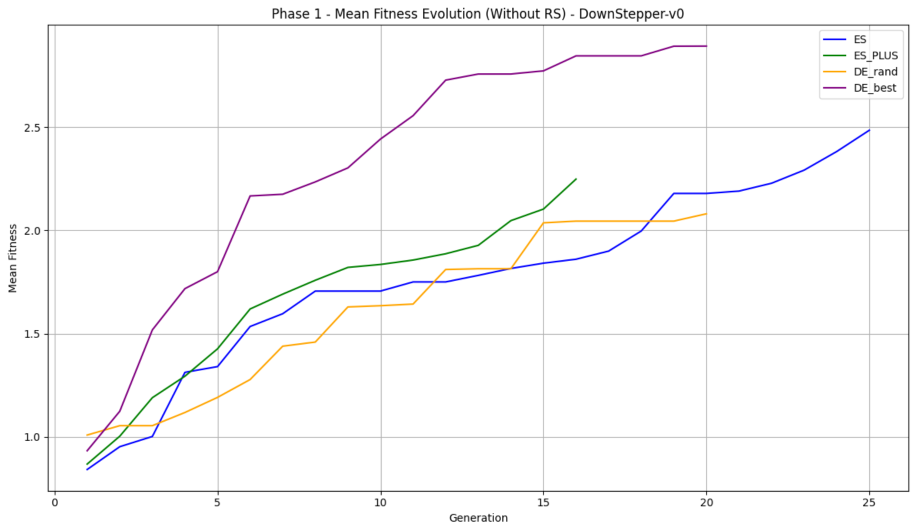

Fig. 15. Mean fitness evolution over generations (excluding RS) on `DownStepper-v0`. (Phase 2.1)

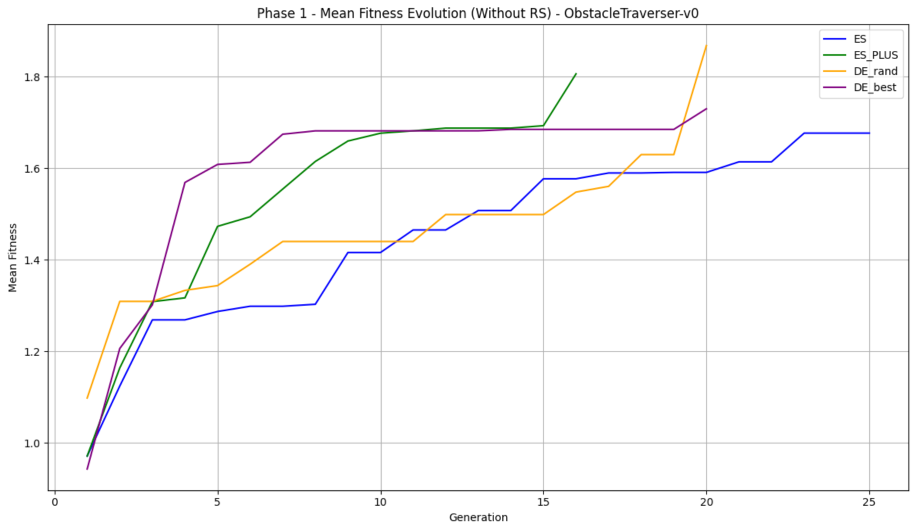

Fig. 16. Mean fitness evolution over generations (excluding RS) on `ObstacleTraverser-v0`. (Phase 2.1)

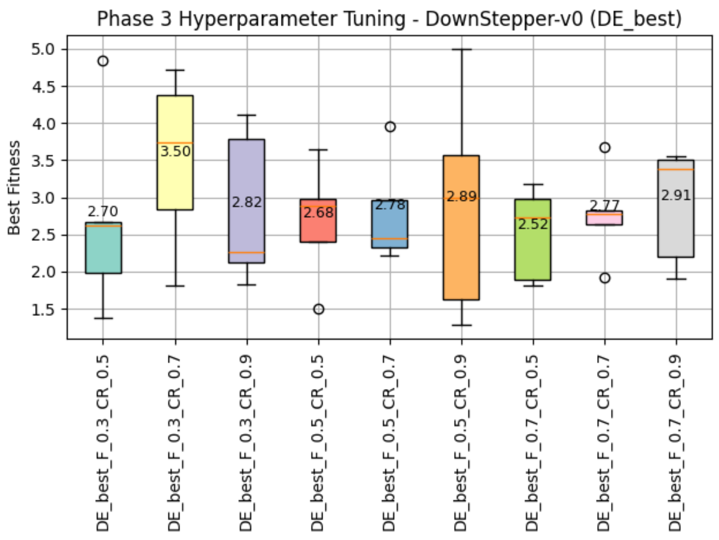

Fig. 17. Fitness distribution for different hyperparameter settings of `DE/best/1/bin` on `DownStepper-v0`. (Phase 2.3)

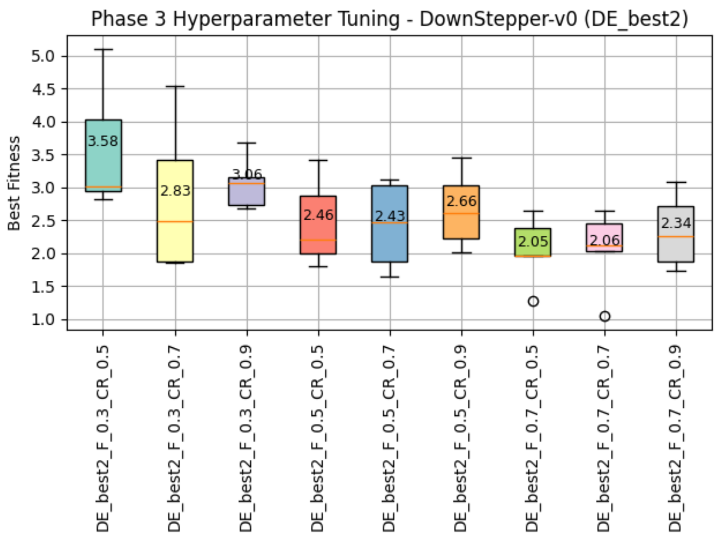

Fig. 18. Fitness distribution for different hyperparameter settings of `DE/best/2/bin` on `DownStepper-v0`. (Phase 2.3)

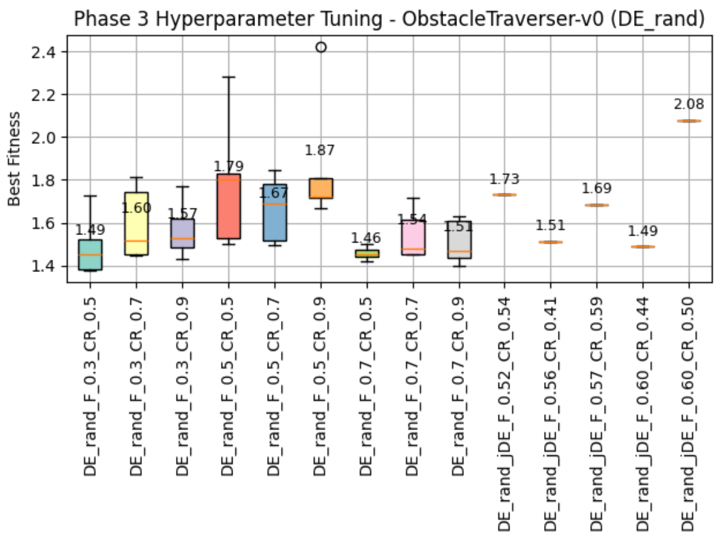

Fig. 19. Fitness distribution for different hyperparameter settings of `DE/rand/1/bin` on `ObstacleTraverser-v0`. (Phase 2.3)

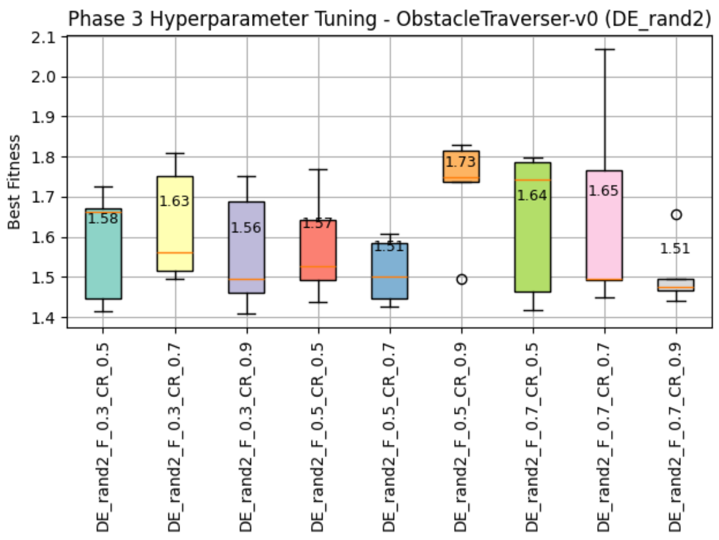

Fig. 20. Fitness distribution for different hyperparameter settings of `DE/rand/2/bin` on `ObstacleTraverser-v0`. (Phase 2.3)

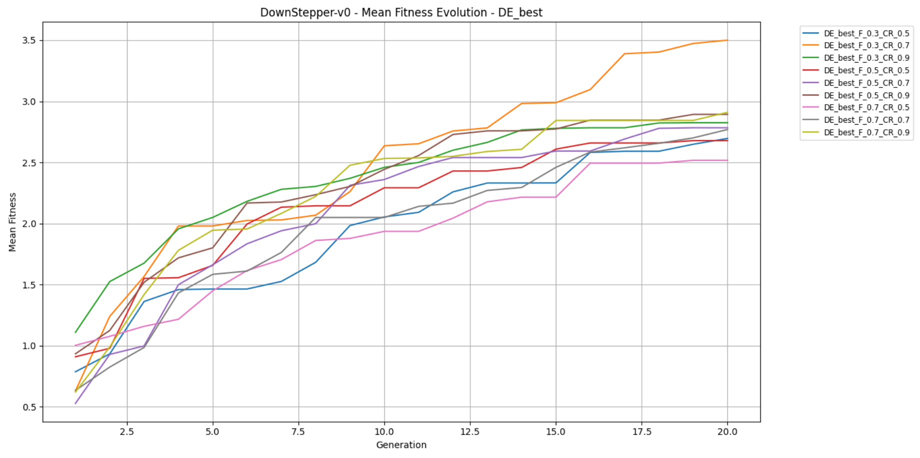

Fig. 21. Mean fitness evolution per generation for `DE/best/1/bin` on `DownStepper-v0`. (Phase 2.3)

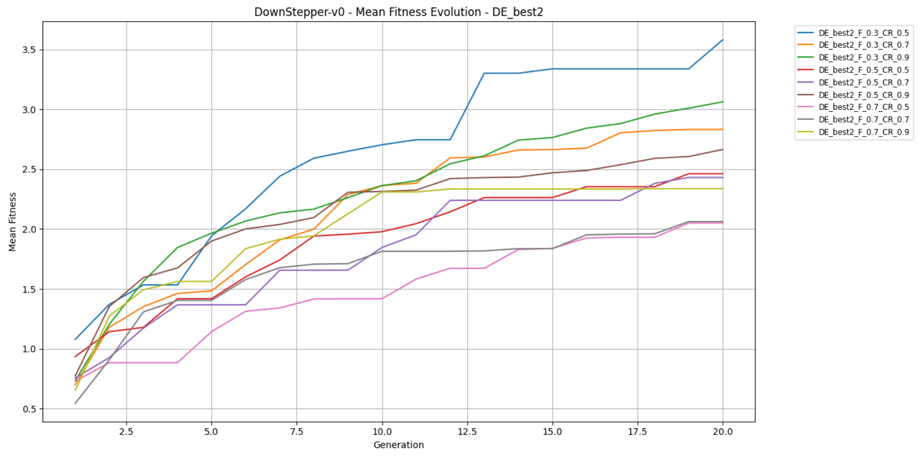

Fig. 22. Mean fitness evolution per generation for `DE/best/2/bin` on `DownStepper-v0`. (Phase 2.3)

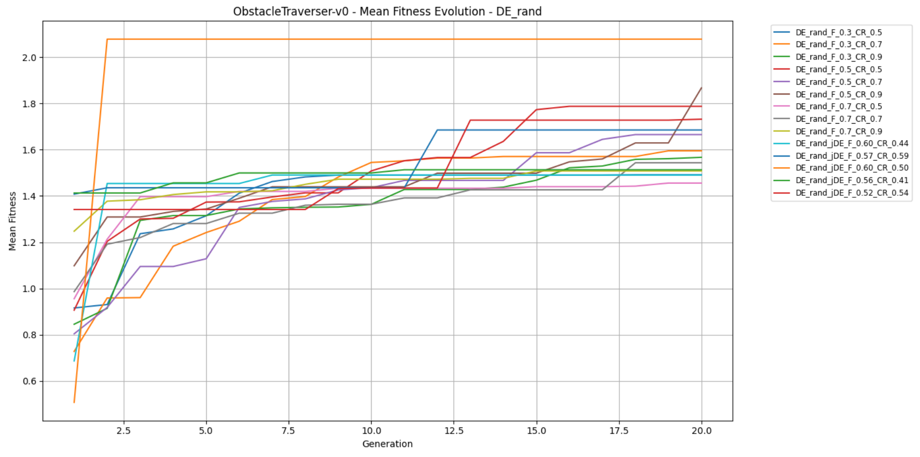

Fig. 23. Mean fitness evolution per generation for `DE/rand/1/bin` on `ObstacleTraverser-v0`. (Phase 2.3)

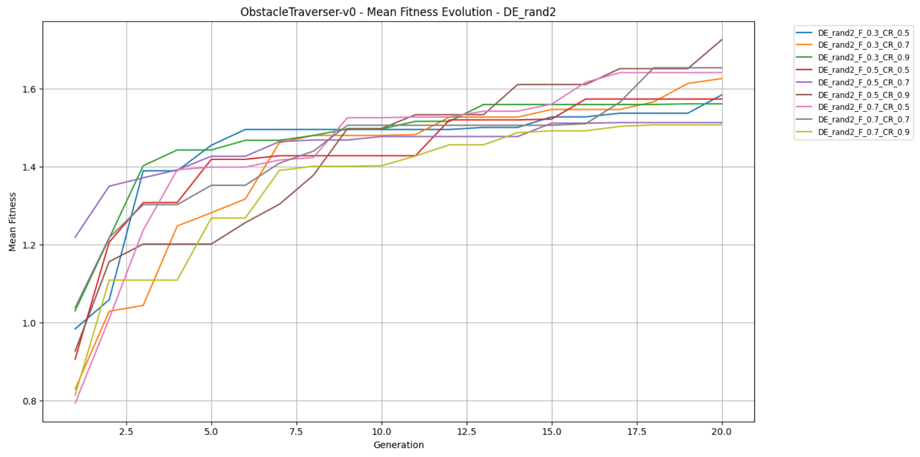

Fig. 24. Mean fitness evolution per generation for `DE/rand/2/bin` on `ObstacleTraverser-v0`. (Phase 2.3)

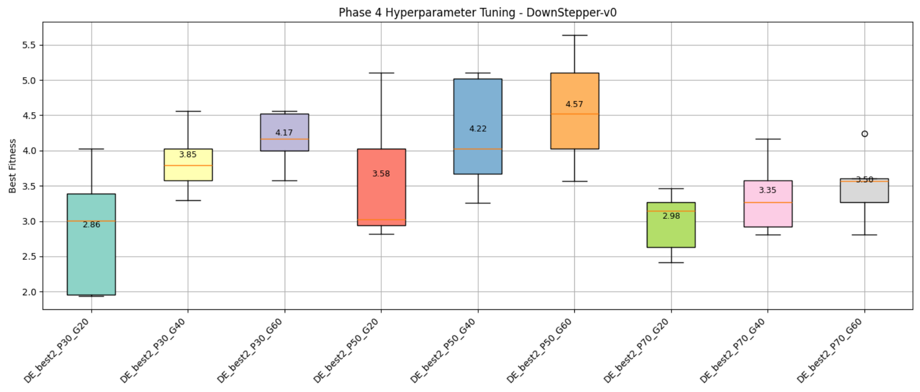

Fig. 25. Final fitness comparison for population and generation settings on `DownStepper-v0`. (Phase 2.4)

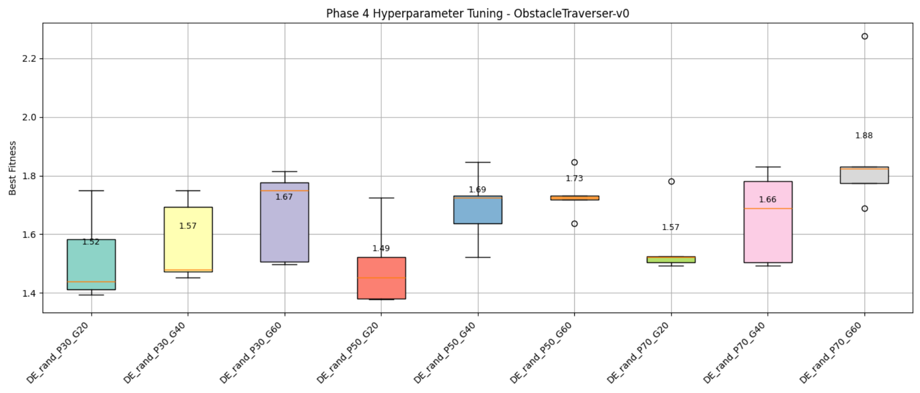

Fig. 26. Final fitness comparison for population and generation settings on `ObstacleTraverser-v0`. (Phase 2.4)

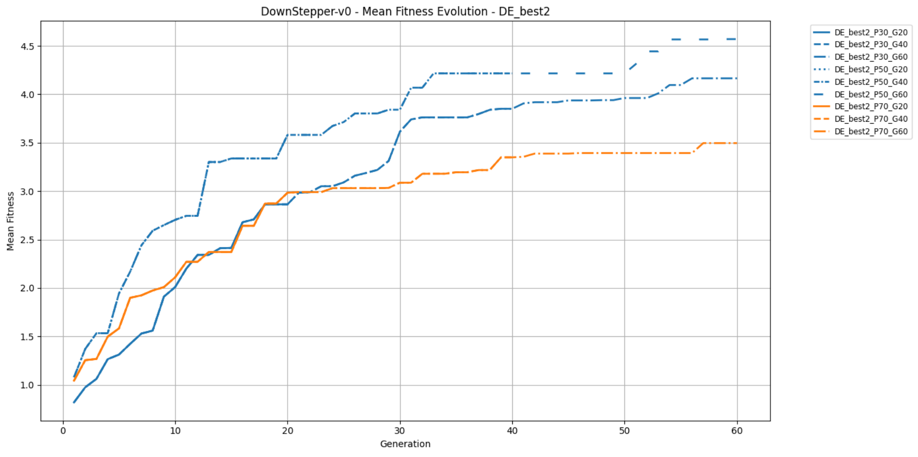

Fig. 27. Mean fitness evolution for different population and generation configurations of `DE/best/2/bin` on `DownStepper-v0`. (Phase 2.4)

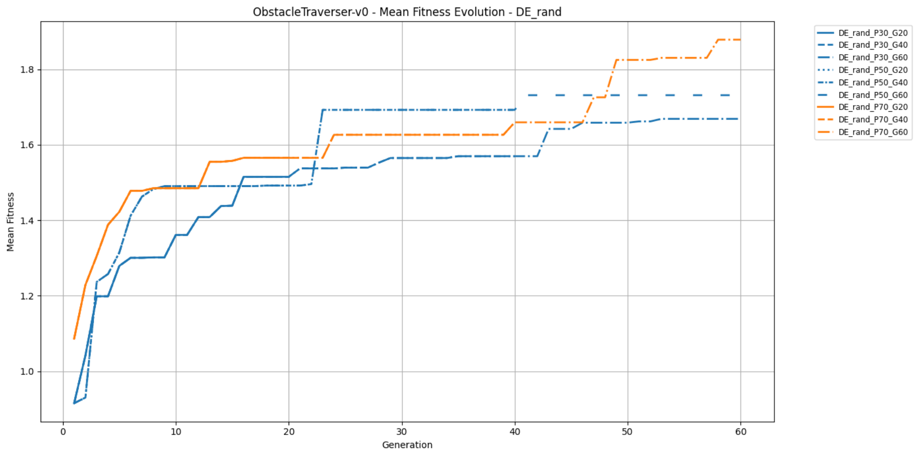

Fig. 28. Mean fitness evolution for different population and generation configurations of `DE/rand/1/bin` on `ObstacleTraverser-v0`. (Phase 2.4)

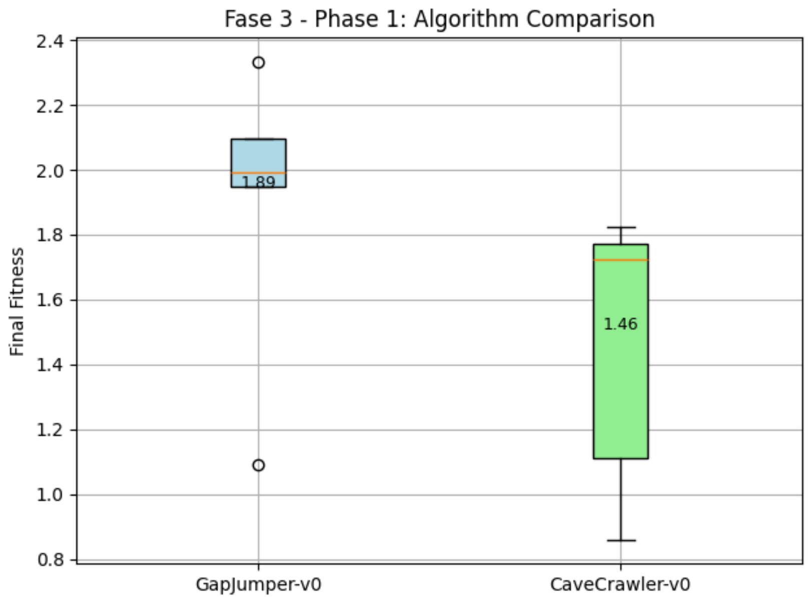

Fig. 29. Final fitness per seed for the co-evolution runs across scenarios. (Phase 3.1)

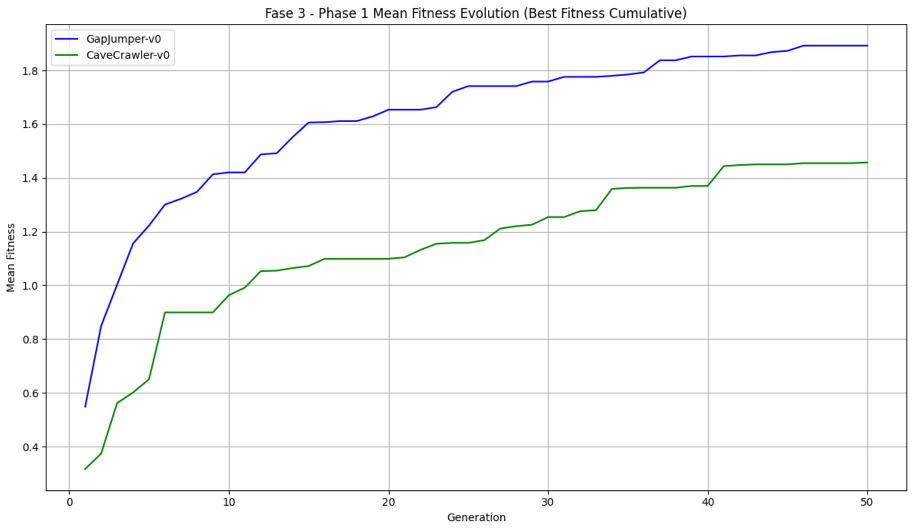

Fig. 30. Mean fitness evolution over generations in co-evolution. (Phase 3.1)