Lamarckian Co-design of Soft Robots via Transfer Learning

Kazuaki Harada
the University of Tokyo
Tokyo, Japan
rahadal-v-l2228@g.ecc.u-tokyo.ac.jp

Hitoshi Iba
the University of Tokyo
Tokyo, Japan
iba@iba.t.u-tokyo.ac.jp

ABSTRACT

In the realm of robot design, co-design aims to optimize both the structure and the controller of a robot concurrently. One approach integrates genetic algorithms to optimize the soft robot's structure with deep reinforcement learning for the controller. A significant challenge in this approach is the inheritance of the controller due to the mismatch of the sensors and actuators of the robots across generations. In this study, we propose a Lamarckian co-design method to inherit the controller optimized by deep reinforcement learning through transfer learning. In experimental evaluations through the Evogym benchmark, we demonstrate that our proposed method achieves an average reduction of 41.7% in the optimization time for robots compared to existing methods and concurrently leads to an average performance improvement of 118.5%. Furthermore, we show that combining the inheritance of controllers with the crossover of structure genomes from two robots allows for additional reductions in optimization time and improvements in performance in several tasks.

CCS CONCEPTS

• **Computing methodologies** → **Evolutionary robotics**; **Reinforcement learning**.

KEYWORDS

soft robotics, co-design, transfer reinforcement learning, evolution gym

ACM Reference Format:
Kazuaki Harada and Hitoshi Iba. 2024. Lamarckian Co-design of Soft Robots via Transfer Learning. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3638529.3654180

1 INTRODUCTION

Creating intelligent agents capable of autonomous operation in complex environments is a significant focus in artificial life research. Soft robots, composed of pliable materials (e.g., polymers, rubber, silicone) resembling living organisms, hold potential industry impact due to enhanced adaptability compared to rigid robots, facilitating safe collaboration with humans [11]. However, designing soft robots is more challenging than rigid robots, as predicting the dynamics of their behavior and determining adaptability to an environment is hindered until controller optimization [22].

Natural life concurrently optimized morphology and brain over time. Co-design, applying this evolutionary process, automatically optimizes a robot's structure and controller [1]. Existing studies proposed co-design algorithms, combining structure optimization (e.g., Bayesian Optimization, Evolutionary Computation) and control optimization (e.g., Evolutionary Computation, Reinforcement Learning (RL)), demonstrating effectiveness in soft robot design through simulation experiments [1–4, 21]. However, designing soft robots for complex tasks remains challenging [1].

Evolution Gym (Evogym) [1] serves as a benchmark for soft robot co-design algorithms, enabling robots to tackle various tasks. Bhatia et al. [1] tested existing algorithms on Evogym, finding the most effective method combined Genetic Algorithm (GA) for structure optimization and Deep Reinforcement Learning (DRL) for controller optimization. Nevertheless, this approach faces challenges related to extensive training time and difficulties in handling complex tasks [1]. We attribute these challenges to the necessity of training each robot individually from the start, arising from the differences in sensors and actuators among individuals. This variability makes it challenging to inherit controllers from parents to children. Although there exist strategies for inheriting controllers enhanced via GA [19], there remains a lack of well-established procedures for inheriting controllers refined using DRL.

In this study, we propose a lamarckian co-design method for obtaining more performant soft robots in less computational time by inheriting controllers optimized through DRL. We train newly generated robots in each generation using Proximal Policy Optimization (PPO), a state-of-the-art DRL method. Consequently, robots exhibiting superior performance during the training phase function as parent individuals, generating offspring that receive both the controller and the physical structure. To address the challenge of sensor and actuator differences between parent and child robots, we utilize a Transfer Learning (TL) approach.

Contributions include:

- Development of a Lamarckian co-design algorithm for controller inheritance optimized by DRL.
- Significant evolution speed improvement for soft robots.
- Improvement in soft robot performance on challenging tasks.
- Validation of the proposed method's effectiveness even with crossover-generated child structures.

The paper initiates with an exploration of related work in Section 2. Following that, Section 3 introduces the proposed method, and Section 4 provides a detailed description of the experimental setup. In Section 5, we analyze the obtained results. Subsequent to the results, Section 6 delves into the implications and interpretations of the findings. Finally, Section 7 concludes the paper by summarizing

GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0494-9/24/07.
https://doi.org/10.1145/3638529.3654180

key insights, addressing method limitations, and outlining avenues for future work.

2 RELATED WORKS

2.1 Robot Co-design

The pioneering attempt to co-optimize the structure and controller can be traced back to Sims' work [14]. Sims designed virtual creatures with rigid parts for structures, representing their controller as Artificial Neural Networks (ANNs) in the form of directed graphs. Employing GA with task achievement as the fitness criteria, Sims successfully evolved virtual creatures capable of walking, swimming, or jumping. Subsequent to Sims' work, co-design studies for rigid robots have gained prominence [5, 10, 23–25]. However, rigid robots faced challenges in tackling complex tasks due to their limited degrees of freedom [1].

Consequently, attention shifted towards the co-design of soft robots, constructed from flexible materials mimicking real organisms. Soft robots, a novel category, consist of flexible materials like polymers, rubber, and silicone [9]. Compared to traditional rigid robots, soft robots offer advantages such as impact absorption during collisions, facilitating safe collaboration with humans. Their high degree of freedom, attributed to the ability to continuously alter their structure shape, enables effective adaptation to diverse environments [11].

2.2 Evolution Gym

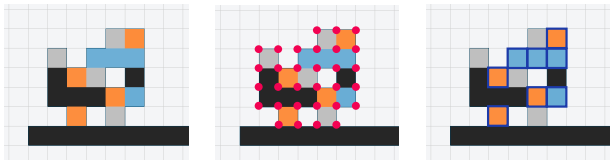
Several attempts have been made to co-design soft robots in simulations [2–4, 21]; however, these studies exhibit some limitations. In these methods, robot control is often constrained to periodic and simplistic patterns, and the assigned tasks are frequently limited to simple activities such as walking on flat surfaces. Consequently, adapting soft robots to complex environments becomes challenging. Additionally, the difficulty in comparing algorithm performance arises from each study conducting experiments in its own uniquely developed environment [1].

To tackle these issues, Bhatia et al. [1] introduced Evogym, a simulation environment designed for co-designing soft robots. In Evogym, soft robots are constructed in a 2D space by assembling four types of voxels made from soft materials. These soft robots possess the ability to deform their structure based on environmental observations, enabling them to execute various tasks such as locomotion, object manipulation, climbing, and more. In this section, we delve into the specifics of the simulation environment provided by Evogym.

The robots in Evogym consist of four voxel types: black rigid voxels, gray soft voxels, light blue horizontal actuators, and orange vertical actuators. An example of a robot's structure is presented in Figure 1a.

At each time step, the robot takes actions by extending or contracting actuators based on observations from sensors. The sensors include:

- **Position Sensor**: Located at voxel vertices (indicated by red circles in Figure 1b), this sensor provides the relative position from the robot's center of mass to itself.
- **Velocity Sensor**: This sensor offers the velocity of the robot's center of mass.



(a) structure (b) position sensors (c) actuators

Figure 1: An example of soft robot in Evogym.

- **Rotation Sensor**: This sensor supplies the rotation angle of the robot.
- **Object Sensor**: In tasks involving object manipulation, this sensor provides two pieces of information: 1) the relative position of the target object's center of mass from the robot's center of mass, and 2) the velocity of the target object.

The availability of sensors varies based on the task, but the position sensor is consistently present for all tasks. Since position sensors are positioned at the vertices of the voxels forming the robot, their quantity depends on the robot's structure, while the robot is equipped with only one sensor each for velocity, rotation, and object. The blue-outlined colored voxels in Figure 1c serve as actuators. Each actuator receives a real-valued input within the range of $[0.6, 1.6]$ from the controller and adjusts its extension or contraction based on the input magnitude.

Since Evogym is a relatively recent development, there are still few research examples utilizing Evogym. In the following two subsections, we introduce two prior studies that serve as baseline algorithms for comparison with our research.

2.3 GA × PPO

Bhatia et al. [1], the creators of Evogym, conducted experiments on Evogym using three structure optimization algorithms: GA, CPPN-NEAT (the method optimizing Compositional Pattern Producing Network (CPPN) [15] through NeuroEvolution of Augmented Topologies (NEAT) [17]) , and Bayesian optimization, each combined with PPO [13]. Through these experiments, they found that the most effective approach for structure optimization is GA.

In this GA-based approach, the robot's genome is represented by an integer matrix of the same size as the robot's structure. In this matrix, the element 0 indicates the absence of a voxel at that location, while elements 1 through 4 correspond, respectively, to a rigid voxel, soft voxel, vertical actuator, and horizontal actuator. For example, the genome of the robot in Figure 1a is represented as follows.

$$\begin{bmatrix} 0 & 0 & 0 & 2 & 3 \\ 2 & 0 & 4 & 4 & 4 \\ 1 & 3 & 2 & 0 & 1 \\ 1 & 1 & 1 & 3 & 4 \\ 0 & 3 & 0 & 2 & 0 \end{bmatrix}$$

The algorithm operates as follows: In each generation, newly born robot individuals within the population undergo PPO learning from scratch, and the results of this learning are assigned as fitness. After learning, only elite individuals are preserved while the rest are eliminated. The eliminated individuals are then replaced by generating new individuals through mutation from the elite individuals.

Mutation occurs with a certain probability for each element of the genome matrix, and the mutated elements are assigned random values from 0 to 4.

2.4 CPPN Genome Encoding

CPPN is an ANN that takes coordinates as input and produces pixel values at corresponding points, generating regular geometric patterns. CPPN evolves to acquire more optimal and complex structures through the NEAT algorithm [17], grounded in GA. This process involves iteratively applying mutations (changes in edge weights or network structures) and crossovers, acquiring more optimal and complex neural network structures.

In several co-design methods for soft robots, CPPN is frequently used as the encoding scheme for genomes [2–4]. In these studies, CPPN's output is interpreted as the type of voxel at the given coordinates, generating the robot's morphology. Moreover, by representing controllers as CPPNs, the inheritance of controllers from parent to child becomes possible.

Tanaka et al. [19] proposed a co-design method that inherits both the structure and controller from parents to their children using the Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) [16]. In HyperNEAT, CPPN receives two coordinates as input, and its output is interpreted as the weights between nodes corresponding to the two points, resulting in the generation of an ANN from CPPN. Tanaka et al. represented the robot's genome using a single CPPN. The CPPN generates two ANNs, where one is responsible for generating the robot's structure, and the other controls the generated structure. Based on fitness assigned depending on task achievement, the NEAT algorithm is applied to the CPPNs to create the next generation's CPPNs. This achieves co-design by inheriting both the structure and controller from parents to their children.

However, the approach of encoding robot controllers using CPPN involves optimization solely through GA, without utilizing gradient-based learning. In a comparative experiment conducted by Saito et al. [12] using evogym, three different optimization methods, namely CPPN-NEAT, HyperNEAT, and PPO, were employed for controller optimization. The findings revealed that PPO demonstrated superior performance in the majority of tasks.

2.5 Transfer Reinforcement Learning

RL serves as a valuable tool for training robots to execute complex control tasks without relying on training data. In RL, an agent receives a state $s_t \in \mathcal{S}$ from the environment at each time step t. Subsequently, the agent decides on an action $a_t \in \mathcal{A}$, and at the next time step $t+1$, it receives a reward r_{t+1} based on the state-action pair (s_t, a_t). Here, $\mathcal{S}$ and $\mathcal{A}$ represent the state and action spaces, respectively, indicating the sets of possible states and actions available to the agent. The agent's goal is to acquire an optimal policy $\pi^*(s)$—a rule for action selection based on states—that maximizes the cumulative reward, given by the sum $\sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in [0, 1]$ is a constant known as the discount factor. The value function $V^\pi(s_t)$ estimates the expected cumulative reward at time $\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ for a given state s_t. The agent learns the optimal policy by iteratively estimating the value function $V^\pi(s)$ through interactions with the environment based on the policy $\pi(s)$ and adjusting the policy $\pi(s)$ using the value function $V^\pi(s)$ [18].

Transfer Reinforcement Learning (TRL) is a method where knowledge acquired by one agent through RL aids in the learning of other agents. The primary objectives of TRL are as follows [8].

- **Learning Speed Improvement**: Achieving a reduction in the number of iterations required to learn the optimal policy.
- **Asymptotic Improvement**: Attaining a more optimal policy compared to scenarios without knowledge transfer.
- **Jumpstart Improvement**: Obtaining an improved policy in the initial stages of learning compared to scenarios without knowledge transfer.

Taylor et al. [20] proposed a method called Behavior Transfer for TRL between two RL tasks T_1 (state space $\mathcal{S}_1$, action space $\mathcal{A}_1$) and T_2 (state space $\mathcal{S}_2$, action space $\mathcal{A}_2$), where generally $\mathcal{S}1 \neq \mathcal{S}2$ and $\mathcal{A}1 \neq \mathcal{A}2$. In Behavior Transfer, the policy $\pi_{(1,final)}$ obtained from RL in T_1 is transformed by a manually designed policy transform function ρ. This transformed policy $\rho(\pi_{(1,final)})$ is then used to initialize the policy π_2 for T_2 (i.e., $\pi_{(2,initial)} = \rho(\pi_{(1,final)})$).

2.6 Learning in GA

Evolutionary theory encompasses three main theories based on Darwin, Lamarck, and Baldwin. While Darwin emphasized natural selection as the driving force of evolution, Lamarck advocated for the importance of learning, proposing that traits acquired through an individual's learning process are heritable. Baldwin, while denying the direct inheritance of acquired traits, suggested that individuals proficient in learning tend to survive more easily, thereby making acquired traits appear heritable.

GA also varies in implementation depending on whether they are based on Darwinian, Lamarckian, or Baldwinian theories. For Darwinian evolution, natural selection is performed after evaluating the fitness of each individual in the population, followed by applying crossover and mutation to the surviving individuals in a repeated process of altering individuals. In the case of Baldwinian evolution, individuals undergo learning before evaluating fitness, and the results of this learning are recorded as fitness. Additionally, in Lamarckian evolution, the learning outcomes are further inherited by newly generated individuals [6].

As outlined in subsection 2.3 and 2.4, existing co-design algorithms for soft robots fall into two categories. One group involves optimizing the robot's controller using DRL without inheriting the controller's learned outcomes [1], thus resembling Baldwinian evolution as learning occurs but outcomes aren't inherited. The other group optimizes the controller using CPPNs [2–4, 19], resembling Darwinian evolution as no learning process is involved.

3 PROPOSED METHOD

In this study, we propose a Lamarckian co-design algorithm for soft robots wherein the optimized controller obtained through DRL is inherited from parents to children. The aim is to reduce co-design time and enable the robot to perform complex tasks that existing methods cannot achieve. Additionally, to demonstrate the effectiveness of inheriting learned controllers when generating robot

Algorithm 1 Proposed Method

Input: population size p, max generations n

for $k = 1$ to p **do**
  initialize structure S_k and controller C_k.
end for
for *generation* $= 1$ to n **do**
  for $k = 1$ to p **do**
    if S_k is a newborn **then**
      train C_k for S_k.
      let f_k be the best score achieved by (S_k, C_k).
    end if
  end for
  perform natural selection
  for $k = 1$ to p **do**
    if k has died **then**
      mutate or **crossover** elites' structure to configure S_k.
      inherit parents' controller and configure C_k.
    end if
  end for
end for
$l = \arg\max_k (f_k)$
return S_l, C_l

Figure 2: Available cutting lines for a robot with a 5×5 shape.

structures by crossing the genomes of two robot structures (a feature not used in Bhatia et al.'s method [1]), we propose a simple method for crossing structure genomes.

The algorithm for the proposed method is presented in Algorithm 1. The processes of mutation and natural selection followed the same method employed by Bhatia et al., as described in Section 2.3. For reproducibility, the code used in the experiments is available online[1].

3.1 Structure Genome Crossover

To encode the structure using genomes, we employed the same method utilizing integer matrices as introduced in Section 2.3. For genome crossover between two parents, we initially cut the two genomes along a randomly selected line (available cutting lines are depicted in Figure 2), then merged them to produce two new genomes. The offspring's genome was randomly selected from these two new genomes. An illustrative example of crossover is presented in Figure 3.

3.2 Controller Inheritance

We performed controller inheritance using TRL. The actor-network, governing control in PPO, comprises a fully connected input layer,

[1]https://github.com/kazukazupi/LamarckianSoftRobot.git

(a) parent1 **(b) parent2** **(c) child**

Figure 3: An example of crossover. The child is configured by crossing over between parent1 and parent2.

two intermediate layers, and an output layer. The input layer's node count corresponds to the robot's state space dimension, the output layer's node count corresponds to the action space dimension, and both intermediate layers have a fixed node count of 64. Given that the state and action spaces depend on the robot structures, the actor-network structures differ between parents and children. Therefore, to inherit learned controllers from parents to children, an appropriate policy transform function is necessary. In this study, behavior transfer was achieved by copying the weights of connections in the actor-network.

The Case of a Child Born from Mutation. Through mutation, the offspring may gain or lose voxels or change voxel types, potentially resulting in the acquisition or loss of actuators/position sensors. To govern the inheritance of edge weights in the actor-network, the following rules were applied (Figure 4):

- Initialize weights of edges connected to nodes linked to newly acquired actuators (or position sensors) through mutation.
- If the child has an actuator (or position sensor) at the same position as the parent, copy the weights of edges connected to the node linked to that actuator (or position sensor) from the parent.
- Copy the weights of edges between nodes in the intermediate layers.

The Case of a Child Born from Crossover. Through crossover, the child inherits actuators and position sensors from both parents. The inheritance of edge weights in the actor-network follows these rules (Figure 5):

- If an actuator (or position sensor) originates from either parent, copy the weights of edges connected to the node linked to the actuator (or position sensor) from the parent.
- If a child's position sensor is located where both parents have one, randomly choose one parent and copy the weights of edges connected to the node to the corresponding part of the child.
- For edge weights in the intermediate layer, randomly choose one parent for each edge and copy the weight of the edge located in the same position.

Furthermore, the weights of the critic-network are shared between parent and child following the same rules. As the critic-network has a consistent output layer size of one node, irrespective of the robot, the edges connecting to the output layer are treated similarly to edges in the actor-network's intermediate layer.

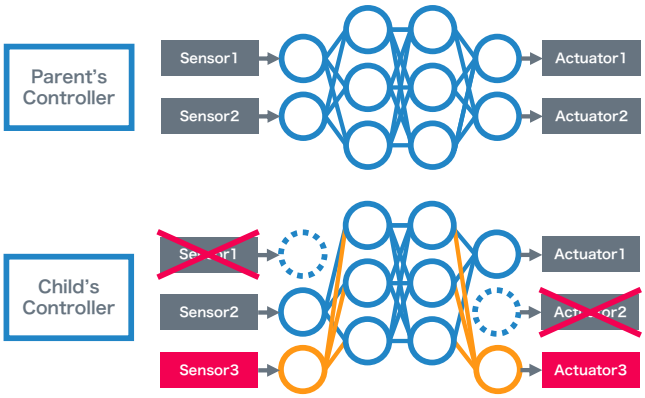


Figure 4: An example of controller inheritance through mutation. The lower part of the figure depicts the controller of a child robot, which, through mutation, lost Sensor1 and Actuator2 but gained Sensor3 and Actuator3. Blue edge weights and blue node biases are copied from the parent, while orange edge weights and orange node biases are initialized.

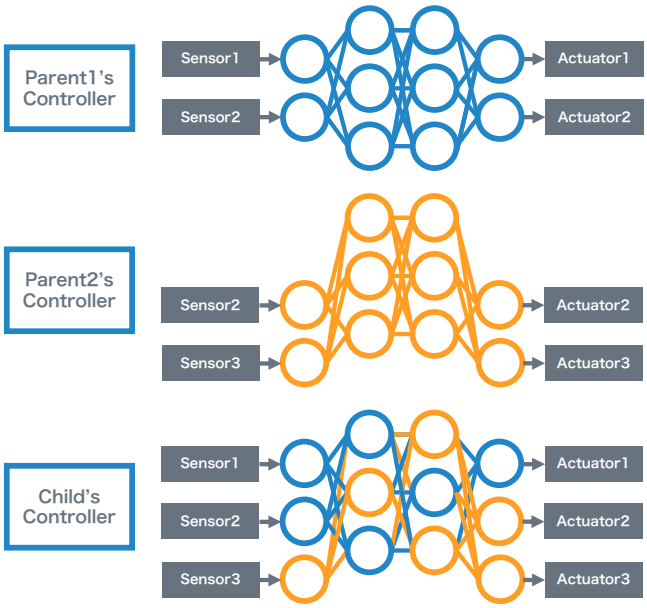


Figure 5: An example of controller inheritance through crossover. Blue edge weights and blue node biases are copied from the parent1, while orange edge weights and orange node biases are copied from the parent2.

4 EXPERIMENTAL SETUP

We conducted experiments on Evogym [1] involving various tasks, comparing three algorithms: (i) the proposed method with controller inheritance and crossover, (ii) the proposed method with controller inheritance but without crossover, and (iii) the existing method without crossover or control transfer. In (ii) and (iii), robots reproduced only through mutation. In contrast, in (i), half of the individuals experienced mutation, and the remaining half underwent crossover for reproduction. For crossover, two randomly selected distinct elite individuals were chosen as parents.

Table 1: GA Hyper Parameters

Parameter Name	Value
robot size	5×5
population size	25
survival rate	$0.6 \rightarrow 0.0$
probability of gene mutation	0.1
crossover rate	0.5

Table 2: PPO Hyper Parameters

Parameter Name	Value
use gae	True
learning rate	2.5×10^{-4}
use linear learning rate decay	True
clip parameter	0.1
value loss coefficient	0.5
entropy coefficient	0.01
num steps	128
num processes	4
evaluation interval	50
train iters	1000

4.1 Hyper Parameters

The hyperparameters for the genetic algorithm are presented in Table 1, while the key hyperparameters for PPO are shown in Table 2. The *survival rate* was linearly reduced from 0.6 to 0.0. However, irrespective of how low the survival rate became, a minimum of two individuals were retained as elite individuals. The *probability of gene mutation* indicates the probability that each voxel of the robot will change into a different type of voxel during the mutation process. The *crossover rate* represents the proportion of robots within the population that are born through crossover. The number of robots to be trained (synonymous with the number of generations) was set for each task based on the reference [1]. These values are detailed in Section 4.2. The implementation of PPO utilized the version developed by Kostrikov [7]. For a fair comparison, all parameters, except for the presence of controller inheritance and crossover, were set to be identical across the three algorithms.

4.2 Tasks

We employed six tasks for the experiments, outlined as follows. Refer to [1] for task details.

- **BridgeWalker-v0:** Easy task of traversing an unstable terrain made of soft material, with reward based on horizontal distance moved. A total of 250 robots are trained for this task.
- **BidirectionalWalker-v0:** Medium-difficulty task of changing movement direction towards a randomly switching goal point, with reward based on distance moved towards the goal. A total of 750 robots are trained for this task.

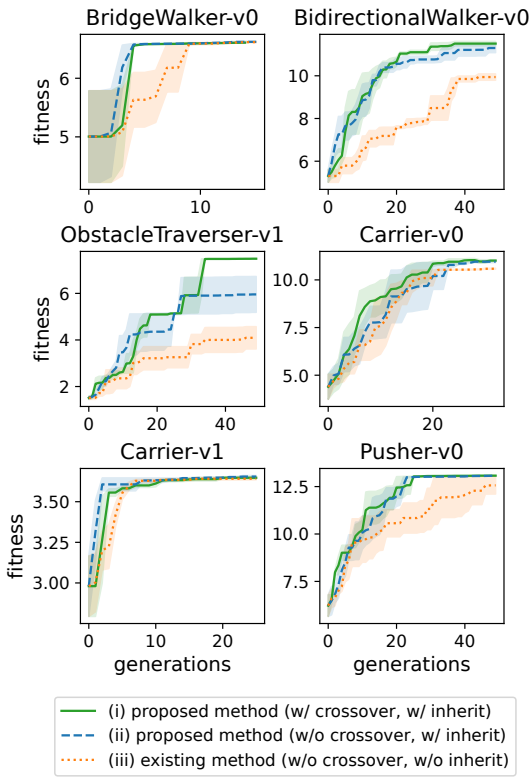


Figure 6: The impact of inheritance and crossover on evolution. We plot the fitness of the best-performing robot in each generation. All curves represent the average over three different runs, and the variance is indicated by shaded regions.

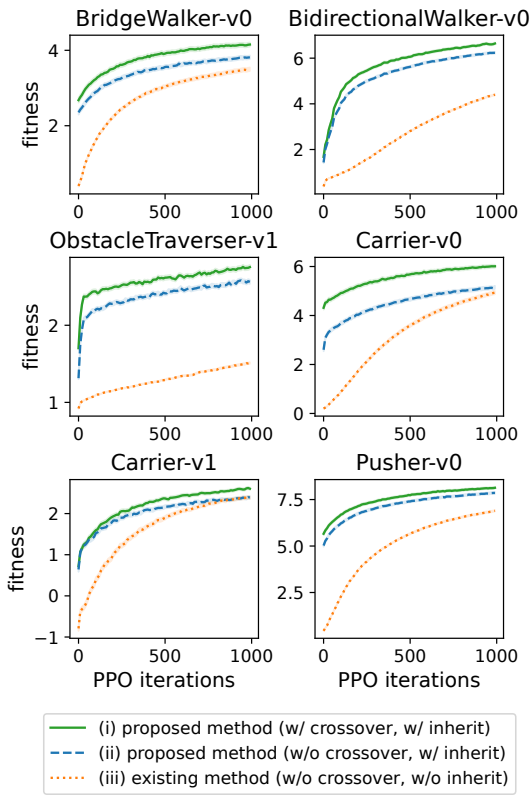


Figure 7: The impact of inheritance and crossover on PPO learning. We plotted the trajectory of the robot's cumulative reward. All the curves are averaged over all of the robots in 3 different random seeds, and the variance is indicated by shaded regions. Please note that the shaded regions may appear small due to the small magnitude of the error.

- **ObstacleTraverser-v1:** High-difficulty task of traversing an uneven path, with fitness determined by total horizontal distance moved. A total of 750 robots are trained for this task.
- **Carrier-v0:** Easy task of transporting objects on a flat path, with fitness determined by total distance the robot transports the target object, penalized if the object is dropped. A total of 500 robots are trained for this task.
- **Carrier-v1:** High-difficulty task of transporting objects on an uneven path. A total of 330 robots are trained for this task.
- **Pusher-v0:** Easy task of pushing an object forward on a flat path, with fitness determined by total distance the robot pushes the target object. A total of 750 robots are trained for this task.

5 RESULTS

The best performance curves of three algorithms across six selected tasks are presented in Figure 6. The following improvements were observed using controller inheritance:

- A reduction in the number of individuals required for robots to attain the fitness of the best-performing robots obtained by the existing method. The average reduction was 41.8% without crossover and 37.3% with crossover.
- An average improvement in the fitness of the final evolved robots of 11.2% without crossover and 11.8% with crossover.
- Excluding tasks BridgeWalker-v0 and Carrier-v1, which were already completely accomplished by the existing method, improvements of 16.7% and 28.1%, respectively, were observed.

Additionally, crossover contributed to additional performance enhancement in ObstacleTraverser-v1. While there was no significant difference in optimization speed with controller inheritance in Carrier-v0, the use of crossover expedited the optimization process.

Learning curves depicting the trajectory of the robot's cumulative reward for each method are presented in Figure 7. In existing methods, robots were scarcely able to accomplish tasks before learning. In contrast, the proposed method enabled robots to exhibit some level of task achievement even before learning. Moreover, at the end of the learning process, the performance of robots in the proposed method surpassed that of the existing method.

6 DISCUSSION

We posit that the superiority of the proposed methods (i) and (ii) over the existing method (iii) can be attributed to the following reasons:

- **Improvement in Learning**: The inheritance of the control system significantly reduced the learning cost for robots, facilitating the acquisition of more complex behaviors.
- **Unimpeded Structure Evolution**: The inheritance of the control system did not impede the morphological evolution of the robots.

In the subsequent subsections, we elaborate on these points.

6.1 Controller Evolution Analysis

The result indicated in Figure 7 underscores that inheriting controller, leading to jump-start and asymptotic improvement, contributes to the efficiency of learning.

In certain instances, the proposed method led to robots demonstrating distinct control strategies in comparison to the existing method. A case in point is illustrated in Figure 8. The best robot from the proposed method rotates its structure to avoid obstacles and traverses more effectively compared to the best robot from the existing method, which jumps high but comes to a stop midway. This trend was consistently observed across all runs of the ObstacleTraverser-v1 task.

6.2 Structure Evolution Analysis

A major concern in establishing this methodology was the potential adverse impact of inheriting controllers on the evolution of body structures. In this method, the differences in the body structures of parent and child robots, caused by mutations, are determined randomly. Consequently, a child robot acquiring or losing voxels, changes the number of initialized parameters but also diverges in optimal control strategies from the parent. This suggests that inheritance may be more beneficial for small structural changes.

However, it was observed that regardless of the method used, the robots' structures converged to similar structures or structures with similar functions. Evolved robot structures in various tasks are compared in Figure 9. In BridgeWalker-v0, robots evolved structures with many voxels in the lower part to increase ground friction or structures without voxels in the lower right part to facilitate a forward-leaning posture. In ObstacleTraverser-v1, robots evolved structures with multiple actuators connected in series, enabling them to strongly push off the ground for leaping over obstacles. In Carrier-v0, robots evolved structures resembling an upper recess for holding objects and a lower structure suitable for walking, either in the form of animal-like legs or by concentrating actuators in the lower part. In Carrier-v1, robots evolved structures with an upper structure acting as a container and a lower structure with a flat contact surface for stable object transportation on stair-like terrain. In Pusher-v0, robots evolved structures resembling two legs of an animal or a forward protrusion, suitable for kicking the object far away.

6.3 The Effect of Crossover

The incorporation of crossover, along with controller inheritance, contributed to enhanced co-design efficiency and the ultimate performance of evolved robots in Carrier-v0 and ObstacleTraverser-v1 tasks. We attribute this improvement to the combining structures from distinct elite individuals, leading to the development of innovative structures.

In Figure 10, *parent1* possesses an upper structure designed for holding objects but is equipped with only horizontal actuators in the lower part, making it challenging to move forward. On the other hand, *parent2* has a lower structure with vertical actuators, and its contour is well-suited for maintaining a forward-leaning posture, enabling strong forward propulsion. However, its upper structure is not designed for holding objects. *Child* inherits the upper structure responsible for holding objects from *parent1* and the lower structure for propulsion from *parent2*, resulting in a configuration well-suited for transporting objects over a distance.

7 CONCLUSION

In this paper, we introduced a lamarckian co-design methodology for soft robots, incorporating TL to inherit the parent's controller. Extensive simulation experiments conducted on Evogym demonstrated the efficacy of the proposed approach. Results indicated that our method accelerated the optimization process of robots compared to existing methodologies, yielding superior performance in challenging tasks. Furthermore, experiments incorporating simple crossover for robot replication underscored the effectiveness of transferring learned outcomes, with indications that crossover can enhance the co-design process.

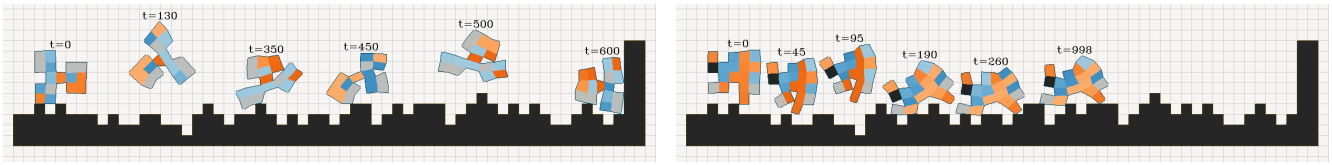
However, it is crucial to mention that the crossover method implemented for the genomes in this research adopted a straightforward approach involving the selection of cutting lines from among eight predetermined alternatives. Employing a more adaptable crossover technique could possibly broaden the search space for body configurations. Furthermore, the genome in this investigation explicitly encoded the body. Consequently, it is vital to examine if the suggested method continues to be efficient when employing indirect body encoding strategies, like CPPN, which are widely used in the literature.

Moreover, this study primarily focuses on improving co-design efficiency for a single task for a single robot. To advance soft robotics or artificial life in real-world scenarios, future research should address methods capable of optimizing robots for tasks involving a combination of multiple objectives or multi-agent tasks where coordination among robots is crucial. In such scenarios, the proposed sharing of learning outcomes among robots holds promise for enhancing learning efficiency.

REFERENCES

[1] Jagdeep Bhatia, Holly Jackson, Yunsheng Tian, Jie Xu, and Wojciech Matusik. 2021. Evolution Gym: A Large-Scale Benchmark for Evolving Soft Robots. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 2201–2214. https://proceedings.neurips.cc/paper_files/paper/2021/file/118921efba23fc329e6560b27861f0c2-Paper.pdf

[2] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. 2013. Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Generative Encoding. *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference.* https://doi.org/10.1145/2463372.2463404



(a) proposed method (w/o crossover, w/ inherit) (b) existing method (w/o crossover, w/o inherit)

Figure 8: Comparison of behaviors of the best robots. We illustrate the behaviors of the best individuals obtained in a single run in ObstacleTraverser-v1 for both the proposed method without crossover and the existing method.

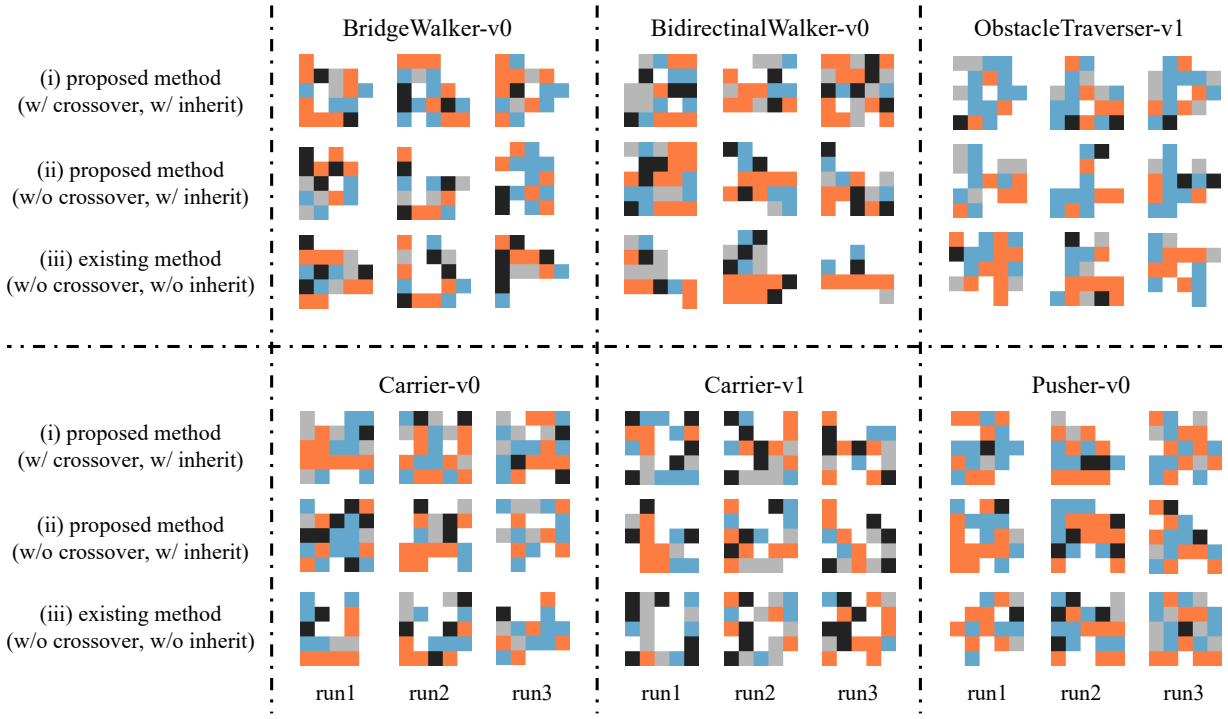
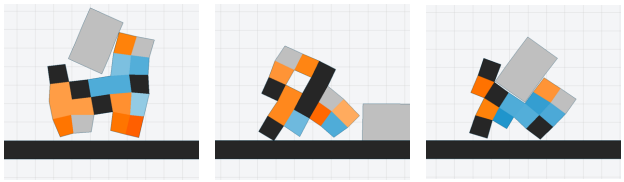


Figure 9: Comparison of evolved bodies. For each method and each run, the structures of robots that achieved the most optimal fitness in each trial were illustrated.



(a) parent1 (b) parent2 (c) child

Figure 10: An example of crossover in Carrier-v0 in the experiments. A child (fitness:7.97) is generated by combining parent1 (fitness:4.52) with parent2 (fitness:4.77).

[3] Francesco Corucci, Nick Cheney, Francesco Giorgio Serchi, Josh Bongard, and Cecilia Laschi. 2017. Evolving Soft Locomotion in Aquatic and Terrestrial Environments: Effects of Material Properties and Environmental Transitions. *Soft Robotics* 5 (11 2017). https://doi.org/10.1089/soro.2017.0055

[4] Francesco Corucci, Nick Cheney, Hod Lipson, Cecilia Laschi, and Josh Bongard. 2016. Evolving swimming soft-bodied creatures.

[5] David Ha. 2019. Reinforcement Learning for Improving Agent Design. *Artificial Life* 25, 4 (11 2019), 352–365. https://doi.org/10.1162/artl_a_00301 arXiv:https://direct.mit.edu/artl/article-pdf/25/4/352/1896269/artl_a_00301.pdf

[6] Andreas Holzinger, David Blanchard, Marcus Bloice, Katharina Holzinger, Vasile Palade, and Raul Rabadan. 2014. Darwin, Lamarck, or Baldwin: Applying Evolutionary Algorithms to Machine Learning Techniques. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, Vol. 2. 449–453. https://doi.org/10.1109/WI-IAT.2014.132

[7] Ilya Kostrikov. 2018. PyTorch Implementations of Reinforcement Learning Algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail.

[8] Alessandro Lazaric. 2012. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning: State-of-the-Art*. Springer, 143–173.

[9] Chiwon Lee, Myungjoon Kim, Yoon Kim, Nhayoung Hong, Seungwan Ryu, and Sungwan Kim. 2017. Soft robot review. *International Journal of Control, Automation and Systems* 15 (01 2017). https://doi.org/10.1007/s12555-016-0462-3

[10] Deepak Pathak, Christopher Lu, Trevor Darrell, Phillip Isola, and Alexei A Efros. 2019. Learning to Control Self-Assembling Morphologies: A Study of Generalization via Modularity. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_

files/paper/2019/file/c26820b8a4c1b3c2aa868d6d57e14a79-Paper.pdf
[11] Daniela Rus and Michael Tolley. 2015. Design, fabrication and control of soft robots. *Nature* 521 (05 2015), 467–75. https://doi.org/10.1038/nature14543
[12] Takumi Saito, Haruki Nishimura, and Mizuki Oka. 2022. Comparative studies of evolutionary methods and RL for learning behavior of virtual creatures. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*. 1059–1065. https://doi.org/10.1109/SSCI51031.2022.10022282
[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. (2017). arXiv:1707.06347[cs.LG].
[14] Karl Sims. 1994. Evolving virtual creatures. *In Proceedings of the 21st annual conference on Computer graphics and interactive techniques (SIGGRAPH '94). Association for Computing Machinery* (1994), 15–22. https://doi.org/10.1145/192161.192167
[15] Kenneth O Stanley. 2007. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines* 8 (2007), 131–162.
[16] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. 2009. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life* 15, 2 (04 2009), 185–212. https://doi.org/10.1162/artl.2009.15.2.15202 arXiv:https://direct.mit.edu/artl/article-pdf/15/2/185/1662702/artl.2009.15.2.15202.pdf
[17] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation* 10, 2 (2002), 99–127. https://doi.org/10.1162/106365602320169811
[18] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning An Introduction*. The MIT Press.
[19] Fabio Tanaka and Claus Aranha. 2022. Co-evolving morphology and control of soft robots using a single genome. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*. 1235–1242. https://doi.org/10.1109/SSCI51031.2022.10022230
[20] Matthew E Taylor and Peter Stone. 2005. Behavior transfer for value-function-based reinforcement learning. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. 53–59.
[21] Merel van Diepen and Kristina Shea. 2019. A Spatial Grammar Method for the Computational Design Synthesis of Virtual Soft Locomotion Robots. *Journal of Mechanical Design* 141, 10 (05 2019), 101402. https://doi.org/10.1115/1.4043314 arXiv:https://asmedigitalcollection.asme.org/mechanicaldesign/article-pdf/141/10/101402/6402274/md_141_10_101402.pdf
[22] Merel van Diepen and Kristina Shea. 2022. Co-Design of the Morphology and Actuation of Soft Robots for Locomotion. *Journal of Mechanical Design* 144, 8 (06 2022), 083305. https://doi.org/10.1115/1.4054522 arXiv:https://asmedigitalcollection.asme.org/mechanicaldesign/article-pdf/144/8/083305/6886716/md_144_8_083305.pdf
[23] Tingwu Wang, Yuhao Zhou, Sanja Fidler, and Jimmy Ba. 2019. Neural Graph Evolution: Automatic Robot Design. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BkgWHnR5tm
[24] Zhiquan Wang, Bedrich Benes, Ahmed H. Qureshi, and Christos Mousas. 2022. Co-design of Embodied Neural Intelligence via Constrained Evolution. arXiv:2205.10688[cs.AI].
[25] Allan Zhao, Jie Xu, Mina Konaković Luković, Josephine Hughes, Andrew Speilberg, Daniela Rus, and Wojciech Matusik. 2020. RoboGrammar: Graph Grammar for Terrain-Optimized Robot Design. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–16.