On the Origins of Robot Morphologies

Mike Zhang
shao.zhang@student.ethz.ch

Mikhail Terekhov
mterekhov@student.ethz.ch

Abstract

The problem of automatic robot morphology design is commonly handled by evolutionary search. We propose a novel approach by formulating this evolutionary search as a Markov Decision Process and using Reinforcement Learning for efficient search through the combinatorial space of robot morphologies. To this end, we implement an evolutionary training environment and Double-DQN and A3C agents for this environment. We find that the evolutionary environment is unlike most common benchmark environments due to the difficulty of accurately evaluating a morphology's fitness. However, we show that for hand-crafted morphology spaces, our trained agents are able to find good performing morphologies.

1 Introduction

A robot is composed of both its physical design (*morphology*) and a controller. The morphology is generally first designed by humans and a controller is designed later. Ideally, a joint design and optimization of the morphology and controller would result in better performance, leading to the field of evolutionary robotics [1, 2, 3, 4], inspired by its namesake process whereby embodied organisms change both in physical morphology and neurological control over generations. In the context of evolutionary robotics, reinforcement learning is used to train a controller for evaluating a morphology's performance, also known as its *fitness*. Learning the control policy is analogous to animals learning during their lifetime, while the morphology optimization is analogous to the evolutionary process changing a species over generations. The major issue that any work in evolutionary robotics must address is the combinatorial nature of the morphological design space. There is no gradient for a change in morphology such as "adding a limb to the robot". To work around this, combinatorial optimization methods such as evolutionary search with random mutations are commonly employed to search through the space of morphologies.

In this project, we take a different approach. The incremental nature of applying mutations to a given morphology allows us to frame *evolution itself* as a Markov Decision Process (MDP). Morphologies become states, mutations become actions, and the change in performance after a mutation is the reward. Note, that existing reinforcement learning algorithms are well-suited for discrete action spaces, such as the Atari games controller [5], meaning our approach is compatible with non-differentiable changes in morphology. Seeing evolution as an MDP allows us to train an evolutionary agent to determine useful mutations to apply as opposed to applying random mutations as it is done in existing morphology optimization methods. Moreover, an MDP point-of-view naturally allows the evolutionary agent to consider the long-term effects of mutations in the genealogy, and not only just immediate improvements in performance. An example evolutionary trajectory that can be produced by such an MDP is presented in fig. 1.

The specific contributions of our work are:

- Implementation of a 'Gym'-like training environment for evolution formulated as a MDP.
- Study and propose solutions to the issue of tractable approximate fitness evaluation.

Foundations of Reinforcement Learning HS2021, Department of Computer Science, ETH Zürich.

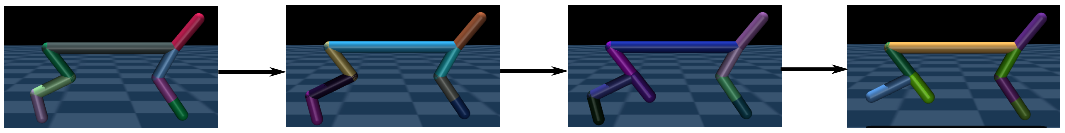


Figure 1: An example trajectory of the evolution MDP. At each step a single mutation is applied (from left to right): change of the length of a limb, addition of a limb, removal of a limb.

- Demonstrate that RL agents trained in this evolutionary environment are capable of learning how to change a robot morphology to achieve better performance on a desired task, empirically justifying our evolutionary MDP formulation.

2 Related Work

Most recent works in robot morphology optimization [2, 3, 4, 6] are based on evolutionary search. A smaller set of recent work [7, 8] perform joint morphology and control optimization using RL, but these works can only handle parametric morphology changes, such as "change the length of the limb". In contrast to the aforementioned methods, Pan et al. [9] propose to handle parametric and nonparametric changes using Bayesian Optimization. However, their method was specifically proposed for hand morphology optimization for grasping tasks with a very limited morphology space.

To the best of our knowledge, our proposed method is the only that formulates morphological evolution as an MDP. As such, there are no existing baselines for direct comparison. In the following, we highlight recent works which are the most relevant to our method.

In *Neural Graph Evolution (NGE)* [2], control policies are parameterized by Graph Neural Networks (GNN) which allows for the policy of the parent to be passed to its children, enabling policy inheritance. The fitness of each morphology is determined using RL training. A learned heuristic function is used to prune morphologies to speed up the search.

In *Task-Agnostic Morphology Evolution (TAME)* [3], the authors formulate an information-theoretic fitness function to measure how would a morphology perform on a variety of tasks in a given environment. Intuitively, this fitness measures if a morphology can predictably reach a wide variety of states. The key advantage of TAME over NGE is that its fitness evaluation does not require any time consuming RL training.

RoboGrammar [6] formulates a grammar by which robot morphologies are built up from. This grammar imposes a structure on the morphology space which gives enough of a prior allowing the authors to efficiently compute a model predictive controller for evaluating fitness. Graph search augmented with a learned heuristic is employed for searching through the morphology space.

From NGE and TAME, we draw the representation of morphologies as trees/graphs and experiment with both methods' fitness functions. In contrast to NGE and RoboGrammar, instead of using learning to augment search, we propose to learn the entirety of the search process.

3 Methodology

3.1 Evolutionary Training Environment

Taking inspiration from common RL benchmark interfaces such as the OpenAI Gym [10] and the Deepmind Control Suite [11], we implement a "gym-like" training environment for our evolutionary MDP which at every step applies a mutation selected from some policy and returns the mutated morphology. The morphology is represented as a tree where the nodes are the limbs of the robot connected by joints/edges. Nodes store information such as the length of the limb while edges store information such as the type of actuation connecting two limbs.

Let $m_i \in \mathbb{M}$ be the morphology from some morphology space $\mathbb{M}$ obtained after i-th mutation. Each mutation is represented as $a_i : \mathbb{M} \to \mathbb{M}$, and deterministic state transitions are

$$m_{i+1} = P(m_i, a_i) = a_i(m_i). \tag{1}$$

In this project, we use the following representation of mutations. Since morphologies are represented as trees $m = (V, E)$, we say that a fixed finite set of actions A can be applied to any node $v \in V$ of the morphology. Set A can include deletion of a node with all its subnodes, modification of some attributes, such as the length of the edge to the parent, addition of another child, etc. Overall, each morphology gets its own set of allowed actions $\mathcal{A}_m$. The set of all possible actions is thus

$$\mathcal{A} = \bigcup_{m \in \mathbb{M}} \mathcal{A}_m, \qquad \mathcal{A} \subset \mathbb{M}^{\mathbb{M}}. \tag{2}$$

This tree representation allows for exploration of sufficiently rich morphology spaces, while maintaining problem tractability. It is also nicely compatible with graph neural networks for learning.

We further introduce the notion of *fitness* $f_i = f(m_i) \in \mathbb{R}$ that corresponds to performance of the morphology m_i at a given task. After applying k mutations to an initial morphology m_0, we will obtain a final morphology m_k. Our evolutionary RL framework tries to optimize the intermediate mutations to maximize f_k. We introduce the reward function

$$r(m, a) = f(a(m)) - f(m) \qquad r : \mathbb{M} \times \mathcal{A} \to \mathbb{R}. \tag{3}$$

When using finite time horizon k and a discount factor $\gamma = 1$, this formulation leads to the desired total reward:

$$R(a_0, \cdots a_{k-1}) = \sum_{i=0}^{k-1} r_i = \sum_{i=0}^{k-1} (f_{i+1} - f_i) = f_k - f_0. \tag{4}$$

With a fixed starting morphology m_0, maximizing R is equivalent to maximizing f_k. In practice, we use $\gamma < 1$, introducing some bias to make the learning task easier. The Evolutionary MDP is then

$$\mathcal{M} = (\mathbb{M}, \mathcal{A}, P, r, m_0, \gamma) \tag{5}$$

3.2 Fitness Evaluation

The best measure of fitness would be the performance of a morphology with its optimal controller for a given task. However, finding an optimal controller is intractable. We thus have to rely on heuristics. Our formulations of fitness are non-deterministic (fitness of a fixed morphology can vary from evaluation to evaluation.) Non-deterministic fitness, together with high computational cost of its evaluation are the main challenges in the project. We will now describe the fitness functions we use.

TAME fitness. We use the *Task-Agnostic Objective for Fitness* introduced in equation (1) in Hejna et.al [3], which presents a detailed explanation of this fitness formulation. Note that the TAME fitness evaluation it does not require any RL training. Instead, an *action-predicting classifier* $q_\phi(a \mid s_T, m)$ is used. In the TAME morphology optimization algorithm, this classifier is trained repeatedly on the growing genealogy of searched morphologies. We cannot do the same, since this would require to progress with all evolutionary unrolls at the same time, in the manner of evolutionary search used in TAME. This approach cannot be combined with RL training, which requires performing unrolls one at a time, or in small batches. To work around this issue, we pre-train the action-predicting classifier by running the TAME algorithm, and then use the final trained classifier for evaluating fitness in the evolutionary MDP.

NGE fitness. A method of finding a lower bound on a morphology's fitness is to use the performance of the control policy trained by NGE as the fitness. The disadvantage of NGE fitness is that each evaluation requires an RL training process. An *inner RL loop* is executed for each morphology. For simple 2D walkers, training a reasonable policy (not necessarily the optimal policy) takes around $10'000$ environment steps, and this corresponds to a minute of runtime on a single core. Because of this we consider NGE's feature of policy inheritance that allows us include the control policy a part of the MDP's state and training each single morphology for less steps. This will, however, introduce bias in the reward signal: if a morphology occurs further in the evolutionary unroll, its policy will have been trained for longer, and likely the fitness will be higher. Similarly, this can introduce delays in the reward signal as a good morphology may be reached but is not reflected in the reward due to a poor control policy. Despite this, we opt to use policy inheritance with NGE fitness or training the evolutionary agents would not be feasible in a reasonable amount of time.

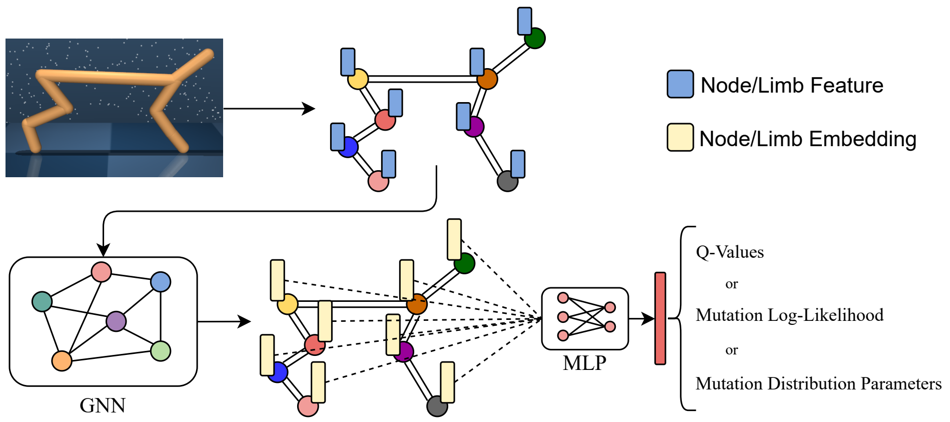


Figure 2: Neural network architecture used for all RL agents. All node embeddings are passed through the same MLP, outputting values corresponding to the allowable mutations for each node.

Pre-defined fitness. For early tests of our RL implementations, we designed "toy problems", where the reward is not associated with any kind of task performance. Instead, the reward for each morphology is hand-crafted and deterministic. This allows for fast fitness evaluation and fast feedback on the performance of our agents.

3.3 Applying Deep Reinforcement Learning

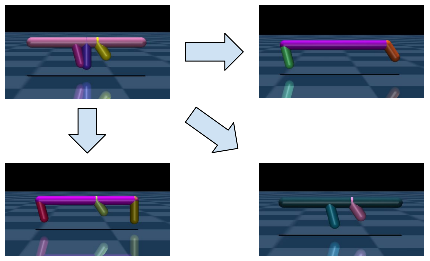
Because the morphology spaces can be extremely large, we apply Deep RL using neural networks to parameterize our agents. For a given morphology, the agent decides which node of the morphology to mutate, and what mutation from a set of allowable mutations will be applied. As the number of nodes may change due to mutations, the dimension of the action space changes accordingly. This violates the standard assumption of constant action space dimension in most implementations of Deep RL algorithms, such as Stable Baselines [12] which therefore cannot be used in our work. Therefore, we have implemented our own versions of some popular RL algorithms, using `PyTorch` and `PyG`[1] for Graph Neural Networks. Our open-source implementation for this work can be found at [2].

A graph neural network consisting of layers of graph convolutions [13] is used to compute contextual node embeddings from the initial node features connected by the tree of the morphology. All node embeddings are passed through the same multi-layer perception which gives the final outputs of the network. Depending on the RL algorithm the final outputs could be estimated Q values, log-likelihood of the actions, or parameters of a distribution for continuous actions. For morphology level outputs such as value estimation, we use pooling to get a single output from the node level outputs. We implemented two Deep RL algorithms for training the evolutionary agent.

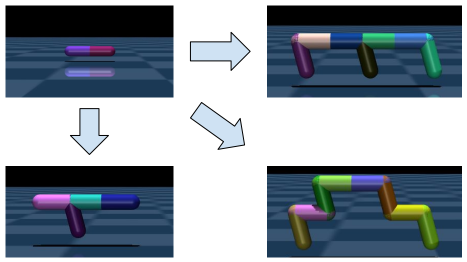
Deep Q-learning. As a simple first algorithm, we implemented Deep Q Networks (DQN) following Minh et.al [5] training with an annealed ϵ-greedy exploration policy. Moreover, we implemented Double-DQN [14] as we suspect Q overestimation to be likely when using non-deterministic fitness evaluation methods.

Asynchronous Advantage Actor Critic (A3C). The bottleneck for training is not in propagating through the neural network, but rather in fitness evaluation and computing the reward at each step. Thus, asynchronous methods are perfect for the task, allowing for several processes to carry out unrolls in parallel and update a shared model. We implemented the A3C algorithm described in [15] with Generalized Advantage Estimation (GAE) [16].

[1]`https://pytorch-geometric.readthedocs.io/en/latest/`
[2]`https://gitlab.ethz.ch/darwin/morphology`



(a) The *Bad Walker* morphology space. The initial morphology m_0 is shown on the top left, and m^*, one of the "interesting" morphologies, better suited for forward motion, is on the top right.



(b) The *Constructor* morphology space. The initial morphology m_0 is shown on the top left, and the others are examples of morphologies that can be obtained within 8 mutations from m_0.

Figure 3: Morphology spaces used in this project.

4 Results

4.1 Experiment Design

Two morphology spaces were designed to test the RL implementations. These spaces are small, where we have a prior of what a good morphology should be to simplify the learning problem. For each space, we have also constructed a pre-defined fitness function for early testing.

The first morphology space is the *Bad Walker*. Two examples of morphologies from this space, including the initial morphology m_0, are shown in fig. 3a. The initial morphology is not suited for walking. When the middle foot is removed, however, the robot is able to move forward. With each mutation, the evolutionary agent can either move one foot to one of four pre-defined locations on the trunk (including both ends of the trunk, as shown in the right morphology m^* on fig. 3a), or remove it. Each mutation results in a no-op when applied to the root node. To construct the pre-defined fitness, we encourage morphologies that are closer to m^*. To this end, we increase the fitness if the long foot is removed and decrease it if one of the short feet is removed. Further, we add a certain value of the fitness depending on how close the short feet are to the corresponding ends of the trunk.

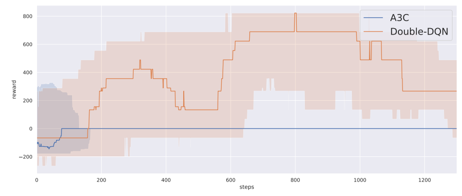
The second morphology space is larger and called the *Constructor*. Some possible morphologies from this space are presented in fig. 3b. We allow for *static horizontal* and *dynamic vertical* edges to be added. Each dynamic vertical edge has a controllable joint, and are slightly inclined to facilitate forward motion. There are 6 mutations that an agent can apply to each node: 4 correspond to adding a horizontal edge (2 edge directions $\times$ 2 possible attachment points) and 2 correspond to adding a vertical edge (corresponding to 2 attachment points). For this morphology space, we construct a pre-defined fitness that does not actually correspond to any "reasonably walking" morphologies. Instead, the fitness is defined such that optimal strategy is to always add vertical edges.

$$f(m) = 100 \times (\text{number of vertical edges in } m - \text{number of horizontal edges in } m). \tag{6}$$

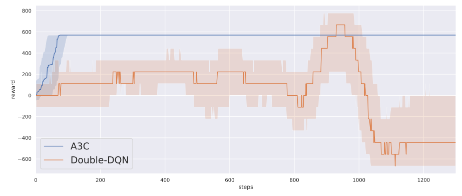
4.2 Experiments

Due to the novelty of our Evolutionary MDP approach, there are no existing benchmark environments that exist in the literature and we must build this infrastructure to conduct experiments. Our implementation builds on top of the open-source code of TAME by Hejna et.al [3]. Conveniently, their codebase also contains a baseline implementation of NGE which we use for NGE fitness evaluation. The hyperparameters for all experiments are given in Appendix A.

Evaluating different fitness functions. Our work implements two fitness evaluation strategies based on NGE and TAME. Both require evaluating the performance of the morphology within a MuJoco [17] task environment. The task is 2D forward motion for NGE fitness, whereas TAME-based fitness uses a *task-agnostic* 2D motion environment.



(a) Bad Walker morphology space



(b) Constructor morphology space

Figure 4: Experiments using pre-defined fitness. Plots of a single run are smoothed using median filtering. Confidence intervals correspond to .25 and .75 quantiles in the moving window of rewards.

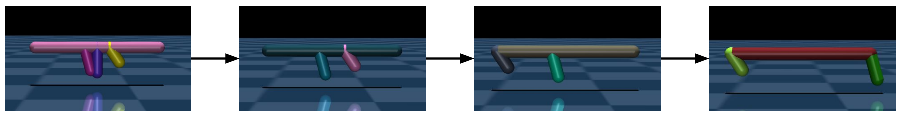


Figure 5: The optimal evolutionary trajectory found by the trained Double-DQN agent.

In NGE fitness, the control policy is trained for a fixed number of steps. We used either 3000 or 5000 steps in our experiments. Thus, using policy inheritance in the unroll, training a reasonably performing controller takes 2-3 steps.

After implementing the TAME fitness evaluation, a major issue became apparent. During the TAME morphology optimization algorithm, the action-prediction classifier q_ϕ is trained repeatedly on the generated morphologies. Surprisingly, the control flow is such that the classifier is never validated on morphologies that it has not seen before during training. Thus it overfits and fails to generalize to unseen morphologies, making it useless for our task. Though an overfitted classifier can still be useful in determining the "predictability" of actions as high variance actions are unlikely to be well predicted by an overfitted classifer. This would help to explain the positive results of the TAME algorithm presented in Hejna et.al [3]. As such, we omit presentation of experimental results using the TAME fitness function.

Training on pre-defined fitness. All RL implementations were ran on both morphology spaces using their respective pre-defined fitnesses. The results are shown in fig. 4. Since discrete morphology spaces with pre-defined rewards result only in a small number of possible fitness values, median-filtered plots only have several distinct levels. Surprisingly, Double-DQN is able to learn a reasonable mutation policy for the Bad Walker but fails for the constructor, and vice versa for A3C.

We believe that the failure of A3C on bad walker can be explained by the dynamics of the actor-critic. In this morphology, a significant share of actions return zero reward. If now the actor correctly predicts it as zero, the resulting policy gradient estimator will also become zero. This will lead to a critical point, from which gradient ascent cannot get out of.

Bad Walker with NGE fitness. We want the evolutionary agent to learn the optimal mutations such that the Bad Walker can walk forwards. The results presented focus on Double-DQN as A3C failed in this space likely for the same reason it failed with the pre-defined fitness. Two different sizes of the Q network with hidden layer dimension 20 and 100 were tried. The resulting training curves are shown in fig. 6. While, both networks were able to solve the task, the larger network was seen to make more stable progress throughout training. Surprisingly, the optimal morphology discovered by the agent is different and better than m^*, which we expected to be the optimal morphology. The difference being that the legs are swapped, leading to a more stable motion! A roll-out of the trained Double-DQN agent's policy is presented in fig. 5, showing that the agent reaches the optimal morphology in the shortest possible number of steps and then stays there by performing no-op mutations for the rest of the rollout. We also provide a video of these morphologies in this roll-out trying to walk forward using their RL trained controllers [3]. From the video, it is clear that the reward is delayed even when

[3]`https://youtu.be/BksKxVRwkpE`

the optimal morphology is reached due to the use of policy inheritance in the NGE fitness. Once the optimal morphology is reached, it still takes two no-op mutations until the inherited control policy is able to move forward at a reasonable speed.

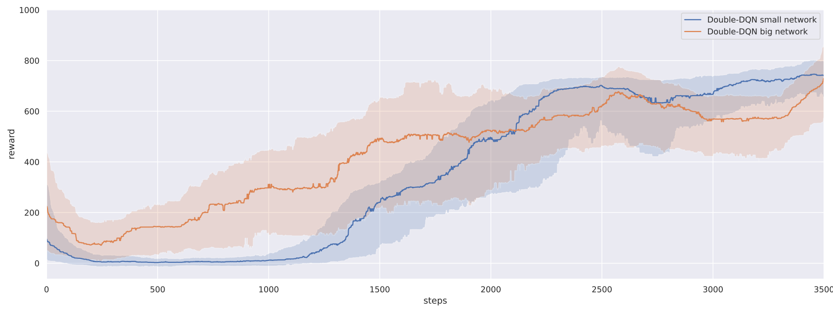


Figure 6: Experiments Double-DQN agent in the Bad Walker morphology space with NGE fitness. Plots of a single run are smoothed using median filtering. Confidence intervals correspond to .25 and .75 quantiles in the moving window of rewards.

Constructor with NGE fitness. The Constructor morphology space is much larger, and therefore takes longer for an agent to sufficiently explore. As A3C was parallelized over 20 cores, it was able to learn a good policy from a large number of steps in a reasonable amount of time. We omit presenting the results for Double-DQN which was not able to achieve any significant improvement of the reward in a reasonable amount of time due to the lack of parallelization. The reward and entropy of the A3C policy over training is presented in fig. 8. During an initial period until around $20'000$ steps, the agent stays in the exploration mode (entropy is high, close to maximal). After that, the entropy begins to decrease and the agent is more exploitative, corresponding to an increase in reward. The policy found by the A3C agent is presented in fig. 7. We provide a video of the morphologies in this roll-out trying to walk forward using their RL trained controllers [4]. The video shows that the agent can produce a simple bipedal robot from a non-moving "stick" morphology. Because the agent adds several legs to the same place, the morphology does not always move well, and the forward motion may be unstable.

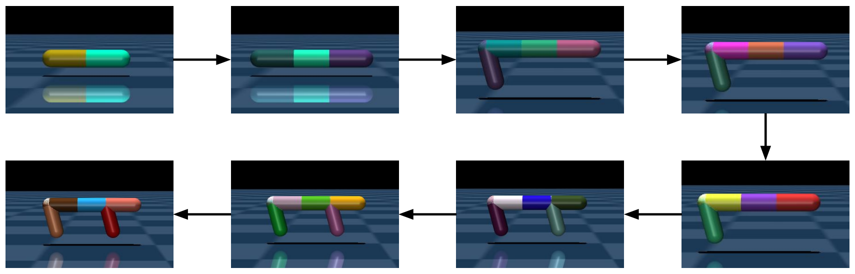


Figure 7: An unroll in the Constructor space using the trained A3C agent. The chosen actions are those with maximal probability from the learned policy. Note that in the Constructor space there is no no-op mutation. Whenever a morphology seems to stay the same, a new leg is added to one of the two places for legs.

5 Discussion and Future Work

This work has demonstrated that evolutionary search over morphology space can be formulated as an MDP which can be be solved by the application of Deep RL algorithms. It must be acknowledged that given the time constraints we were not able to investigate many aspects of the problem in further details. Some clear examples being, does Double Q-learning help? How much does policy inheritance effect training? Have we found the best hyperparameters? What about other RL algorithms (e.g. Rainbow, PPO [18, 19])?.

[4]`https://youtu.be/h6nEH1_pLbE`

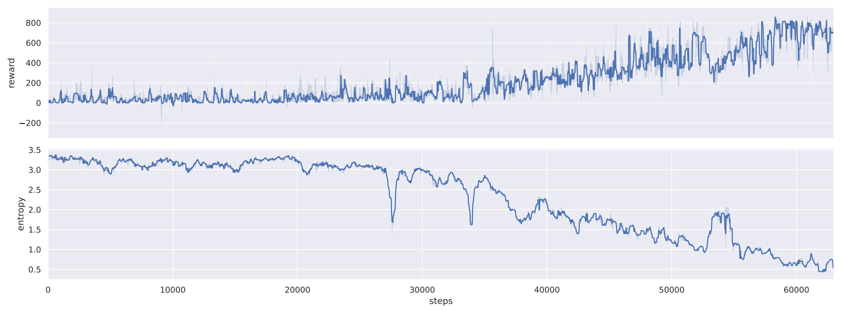


Figure 8: Experiments with A3C agent in the Constructor morphology space with NGE fitness.

The limiting factor to large-scale experiments is the time-consuming fitness evaluation. This is one of the factors that differentiate the evolutionary environment from most RL benchmark environments which generally step quickly, use deterministic reward functions, and assume constant action spaces. The evolutionary environment does not fit any of these descriptions and may be useful as a new difficult benchmark to test RL algorithms. Despite this, RL algorithms designed and tested on standard benchmark environments are shown to work for the evolutionary environment which exceeded our initial expectations given the stochastic and possibly delayed reward signals. The caveat here was that we evaluated the RL algorithms on our hand-crafted structured morphology spaces. The open question still remains to be seen if for very large (or even unbounded) morphology spaces with large action space, RL agents are able to learn to find useful morphologies.

We do not compare the performances of the morphologies found by our method against other morphology optimization algorithms such as NGE or TAME. The standard benchmark for comparing morphology optimization algorithms is to run them for a fixed number of generations and compare the best morphology found in the respective genealogies and the total wall time for the searches . However, our RL-based approach does not fit to this benchmark. As our method is purely data-driven, it has a slow training process, but the trained policy can be executed quickly. This leaves the open question of how applicable is our RL-based method as a morphology optimization technique?

For completeness we include a short discussion on failed experiments in Appendix B.

Given the discussion, we present multiple promising directions for future work.

More Extensive Experiments and Hyperparameter Search. The task remains to produce a more extensive set of result by benchmarking a larger selection of Deep RL algorithms and extensive hyperparameter search to disambiguate algorithmic failures from poor choices of hyperparameters.

Better Exploration. In this work, we employed the common ϵ-greedy exploration policy. Given how expensive a single step in the evolutionary MDP is, more intelligent exploration methods which take informed exploratory actions should be investigated.

Scaling Up. The question of interest is can our approach learn useful mutations for morphology spaces much larger than those presented in this work? While we implemented asynchronous training with A3C, we were still limited by the computing resources available. Another promising direction may be to first massively parallelize collection of evolutionary roll-out data and then training an agent on this data using offline/batch RL [20].

Handling Parametric/Continuous Morphological Changes. Our current implementation does not handle continuous morphology changes. Continuous actions cannot be handled by DQN based algorithms. This further motivates experiments with policy optimization algorithms which can handle both discrete and continuous actions.

Imposing Additional Structure on the Morphology Space. A promising direction is to take inspiration from *RoboGrammar* [6] by defining a grammar/ruleset by which morphologies are constructed which explicitly enforces structure on the morphological space. This gives strong priors for the function class of the optimal control input (e.g. periodic inputs for walking), allowing for good initial guesses which can be refined by a model predictive controller much faster than RL training.

References

[1] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22, 1994.

[2] Tingwu Wang, Yuhao Zhou, Sanja Fidler, and Jimmy Ba. Neural graph evolution: Towards efficient automatic robot design. *arXiv preprint arXiv:1906.05370*, 2019.

[3] Donald J Hejna III, Pieter Abbeel, and Lerrel Pinto. Task-agnostic morphology evolution. *arXiv preprint arXiv:2102.13100*, 2021.

[4] Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. Embodied intelligence via learning and evolution. *arXiv preprint arXiv:2102.02202*, 2021.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[6] Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

[7] Charles Schaff, David Yunis, Ayan Chakrabarti, and Matthew R Walter. Jointly learning to construct and control agents using deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9798–9805. IEEE, 2019.

[8] Lucy Jackson, Celyn Walters, Steve Eckersley, Pete Senior, and Simon Hadfield. Orchid: Optimisation of robotic control and hardware in design using reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2021)*. University of Surrey.

[9] Xinlei Pan, Animesh Garg, Animashree Anandkumar, and Yuke Zhu. Emergent hand morphology and control from optimizing robust grasps of diverse objects. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7540–7547. IEEE, 2021.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[11] Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control, 2020.

[12] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. *GitHub repository*, 2019.

[13] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[14] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[15] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[16] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[17] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[18] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[20] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[21] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[22] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

A Hyperparameters of the experiments

A.1 NGE Parameters

These are the parameters of the trained NGE control policies that were used for NGE-based fitness evaluation. Apart from the number of training steps, these parameters were kept fixed throughout all experiments.

Parameter	**Value**
policy network arch	$[64, 64, 64]$
inherit policy?	True
unrolls for fitness evaluation	9
PPO parameters	
number of epochs	8
number of steps	1000
entropy coefficient	0.001
learning rate	0.0003
batch size	128

Table 1: NGE hyperparameters

A.2 Double-DQN on bad walker

Parameter	**small Q network**	**big Q network**
Network architecture		
graph encoder arch	$[20, 20]$	$[100, 100]$
MLP arch	$[20]$	$[100]$
ϵ-greedy exploration		
starting ϵ	1.0	
final ϵ	0.1	
annealing strategy	linear	
number of annealing steps	20000	
Replay Buffer		
buffer size	10000	
batch size	32	
Other parameters		
γ	0.99	
Steps per episode	10	
NGE training steps per morphology	3360	
reward downscaling factor	100	
steps before resetting the target net	150	
optimizer	Adam	
learning rate	0.0003	
loss	smooth L_1	

Table 2: Hyperparameters of Q learning with NGE fitness on the bad walker space.

For the pre-defined bad walker fitness, we used the small Q network with the same parameters.

A.3 A3C on constructor

The parameters are presented in Table 3.

Parameter	Value
Value/Policy network	
graph encoder arch	$[64, 64, 64]$
node embedding dimension	64
value MLP arch	$[128]$
policy MLP arch	$[128]$
global aggregation strategy	mean
shared value and policy encoding	True
Other parameters	
number of processes	20
steps per episode	9
entropy coefficient	0.02
value loss coefficient	0.5
reward downscaling factor	100
γ	0.95
use GAE	True
λ for GAE	0.95
optimizer	Adam
learning rate	0.0003

Table 3: Hyperparameters of A3C with NGE fitness on the constructor space

B Brief Discussion of Failed Experiments

We implemented an asynchronous version of Deep Q Networks following [15] to speed up training via parallelization. Unfortunately, we were not able to solve deadlocking issues which appeared to occur when running the NGE fitness RL training within the separate worker cores.

For handling the non-deterministic fitness evaluation and resulting reward, we thought to apply work in the area of distributional RL [21] which attempts to estimate the full distribution of returns as opposed to just the expected return. The distributional RL formulation naturally handles stochastic reward functions.We implemented Quantile-Regression Deep Q Networks (QR-DQN) [22], a distributional RL algorithm in hopes of better handling the stochastic reward signal. However, we were unable to obtain successful results with QR-DQN within the time constraints.