# How Different Encodings Affect Performance and Diversification when Evolving the Morphology and Control of 2D Virtual Creatures

Frank Veenstra[1] and Kyrre Glette[1,2]
[1] Department of Informatics, University of Oslo, Norway
[2] RITMO, University of Oslo, Norway
frankvee@ifi.uio.no

## Abstract

A challenge in evolutionary robotics is the in parallel adaptation of morphologies and controllers. Here, we considered encoding methods for morphogenesis of 2D virtual creatures that can be created from directed trees. Using an evolutionary algorithm, we optimized locomotion in these virtual creatures and compared a direct encoding, an L-System, and two types of encodings that produce neural networks—a Compositional Pattern Producing Network (CPPN) and a Cellular Encoding (CE). We evaluated these encodings based on performance and diversification, and we introduced an OpenAI gym environment as a computationally inexpensive benchmark for exploring morphological evolution. The direct encoding and L-System generated more fit solutions compared to the network strategies. Considering morphological diversity, the direct encoding finds solutions more locally in the morphological search space, the L-System made larger jumps across this search space, and both network approaches also make larger jumps though find fewer solutions in this space. With these results we show how encodings exhibit different characteristics as developmental approaches. Since the genotype-phenotype mapping plays a major role in evolutionary robotics, further modifications using more complex tasks and environments can lead to a better understanding of morphogenesis and thereby improve how morphologies and controllers of robots are evolved.

## Introduction

A major challenge in Evolutionary Robotics (ER) is concerned with optimizing both the morphology and control of robots. For virtual creatures, it is still surprising much work barely surpasses that of what Sims (Sims, 1994) accomplished two and half decades ago (Kriegman, 2019). Many of the challenges in ER can be associated with the design of the algorithms that optimize and construct the robots (Cheney et al., 2016). Mostly, recent advances have been made by altering the Evolutionary Algorithms (EAs) to either promote diversity (Lehman and Stanley, 2011), or find gradients in the search space to exploit (Hansen et al., 2003). Evolving populations in nature convey a blind process of adaptation while still needing to adhere to principles of possessing *evolvability*. Wagner and Altenberg (1996) stated that a critical factor in shaping this evolvability in biological organisms is the *representation problem*, or *genotype-phenotype*

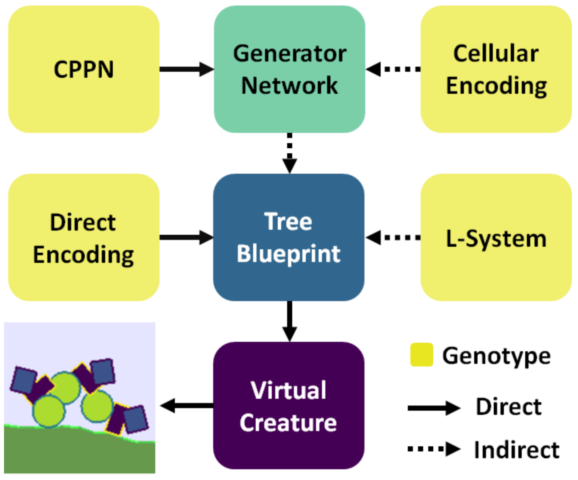


Figure 1: **Illustration of how encodings generate virtual creatures**. The solid lines represent a direct encoding step whereas the dotted lines represent an indirect encoding step. The encodings contain the evolvable parameters (genes) and the blueprint entails how to construct a robot in Box2D.

*mapping* (Nolfi and Floreano, 2000). In contrast to the optimization strategies that operate on the phylogenetic time scale (Pfeifer and Bongard, 2006) of evolving populations, we address the workings of the genotype-phenotype mapping by looking at various encoding strategies as developmental abstractions that can be used for optimizing both the control and morphology of 2D virtual creatures.

Genotype-phenotype mappings can be created through direct encodings (one-to-one mappings), and various types of indirect (generative) encodings. For the creation of virtual creatures, indirect encodings have been implemented in the form of rewriting systems (Sims, 1994; Hornby et al., 2003; Veenstra et al., 2017), morphogen-based (Turing, 1952) / cell chemistry approaches (Stanley and Miikkulainen, 2003; Bongard and Pfeifer, 2003), and neural networks (Stanley, 2007; Auerbach and Bongard, 2010; Cheney et al., 2014; Ha, 2019), to name a few. Even though these types of encodings can alleviate some design requirements for EAs, challenges such as prematurely converging morphologies

(Joachimczak et al., 2016; Cheney et al., 2018) are still prevalent. Even when comparing encodings when evolving virtual creatures, significant differences in diversification aren't always detectable (Miras et al., 2018). Design challenges for creating real-world robots from evolved components (Lund et al., 1997; Lipson and Pollack, 2000; Faíña et al., 2013), could greatly benefit from designs that allow for the reuse and recursion of specific components, as is usually done in modular robots (Stoy et al., 2010; Auerbach et al., 2014).

Robot and virtual creature components are commonly organized in lattices or articulated chains. In voxel-based soft robots (VSR), a lattice of similarly sized volumetric components is constructed where individual components directly affect neighboring components via joints (Hiller and Lipson, 2014). The big advantage of this method is that the entire robot morphology can be defined in cartesian coordinates. When using articulated chains however, representing components in cartesian space is challenging when sizes and orientations of components vary. The hierarchical component dependency in the chain type approach does allow for morphologies to be generated from directed trees. Since many robot simulators have a similar hierarchical component dependency, we evaluate four types of encodings that can generate directed trees, which serve as blueprints for the morphologies of virtual creatures (Figure 1).

Ha (2019) experimented with adjusting the morphology of a virtual creature as its design, especially in reinforcement learning, is rarely considered for optimization for the task at hand. In addition, we would therefore like to emphasize that it becomes even more challenging when the body of a robot changes its number of inputs and outputs during the optimization process. Withdrawing from using advanced robotics simulators, here, we focused on a minimal testbed with low computational requirements using the Box2D physics engine (Catto, 2019). This has been implemented by creating a new environment for OpenAI gym (Brockman et al., 2016), which aims to expose benchmark problems to a common interface. In this environment, we created a simple scenario for investigating the genotype-phenotype mappings for evolving directed tree-based blueprints for virtual creatures. We believe that this can be a unifying, easy-to-use testbed for experimenting with morphological evolution. As the bipedal walker testbed is widely used for optimization methods (Klimov, 2019), we employ a similar environment using a fixed random seed for creating terrain that becomes progressively rougher. While utilizing the Distributed Evolutionary Algorithms in Python (DEAP) (Fortin et al., 2012) for the EAs, we added a simple *generator network* architecture that allowed robots to be constructed from neural networks. We use the term generator network to define neural networks that are queried for the construction of the blueprints for the virtual creatures.

In our experiments we evaluated a direct encoding, a parametric Lindenmayer-System (L-System) (Lindenmayer and Jürgensen, 1992), and two neural network implementations as our generator networks. The first neural network is a Compositional Pattern Producing Network (CPPN), a type of neural network that has been frequently used for morphogenesis in evolving virtual creatures robotics (Auerbach and Bongard, 2011; Cheney et al., 2014). The second is a network created using a Cellular Encoding (CE) (Gruau et al., 1994). With these implementations we hope to align a few concepts for the generation of virtual creatures and modular ER; using encoding strategies that will be compatible with various optimization methods and simulation environments. Apart from evaluating the performance of these encodings, we are also interested in how they exhibit search space traversal.

Our contribution is twofold. Firstly, we implemented various encoding strategies and evaluated them based on performance and diversity when optimized using a standard EA. Secondly, the testbed introduced in this paper will allow users to test optimization strategies for morphological evolution. Additional neural network strategies that might be useful for morphogenesis can be included by using them as an encoding defined by the generator network. With the implementation of the testbed as an OpenAI gym environment, we hope that the machine learning community will be able to experiment with various optimization methods for evolving both morphology and control systems that are useful for virtual creatures and robotics.

## Methods

Our experiments were conducted using the OpenAI gym framework where we introduce a new gym environment for morphological evolution in modular robotics[1]. In this framework, we focus on morphological evolution of robots that are created through directed trees. Different encoding strategies can be used to construct these directed trees. These trees that serve as blueprints can be interpreted into a virtual creature. The trees are built by initially creating an axiom node (root of the tree), and iteratively expanding the tree structure by adding new nodes. Every node in this tree encapsulates information about a module that can be created in the simulation environment. The modules that were used could have up to three other modules attached to them. A module consists of one specific component, which can have various shapes and control systems.

In addition to measuring the performance of the virtual creatures, we measured a simplified tree edit distance (TED) for each individual in a population compared to all other individuals in the population. Here we only counted the number of vertex insertions or deletions between two trees without counting substitutions. This measure gave us an indica-

[1]source code of the platform can be found at `https://github.com/FrankVeenstra/gym_rem2D`

tion of the existing morphological diversity within a population at any given time. The tree blueprints of the elites were also plotted to see how populations traversed the morphological solution space.

The original *bipedalwalkerv*2 environment (Klimov, 2019) rewards agents based on how far they moved within a certain amount of time. Since our approach can create many individuals that through random mutation might perform badly, or even move in the opposite direction, we added a feature in the environment that stops evaluating an individual if it is does not move forward quickly enough. We modeled this feature by adding a vertical line that slowly moves forward, ending the evaluation when it crosses the root module of an individual being evaluated. We modified the environment to become progressively more challenging. This is implemented through increasing a noise factor that generates the environment. For the experiments we fixed the random seed for procedurally generating an environment that was deemed suitable (Figure 2). This environment does not contain any large downward slopes and ends with a large hill. The task for the virtual creatures, that are created near the flagpole, is to move to the right as far as possible.

## Modules

In our experiments, modules are basic components of the robot that encapsulate part of the functionality of the entire robot (Stoy et al., 2010). However, for simplicity, and ensuring not to introduce many convoluting factors, we limited ourselves to using simple primitive shapes as our modules. In addition, we excluded any collisions between modules in our implementation. Each module contains three connection sites where other modules can attach (top, left and right). The top connection site attaches a new child module directly where the relative y vector of the parent module crosses the simulated edge of the parent module. The left and right connection sites attach a new module at a specified angle relative to the parent module's coordinates and orientation—left being defined as the negative angle, right as a positive angle.

For controlling our robots, we implemented an open-loop controller where each module contained a simple sine wave function:

$$y(t) = A\sin(\omega t + \varphi) + D \tag{1}$$

where $A$ is the amplitude, $\omega$ is the frequency, $\varphi$ represents the phase and $D$ is the joint angle offset. In addition to the equation, a maximum angle was defined as $\pi/2$—the negative and positive values determined the outer bounds of $y$. The controller mutation operator changes the amplitude, frequency, phase and offset of the sinusoidal wave function using a gaussian distribution random number generator. All controller values as stored in the genotype were normalized between -1.0 and 1.0.

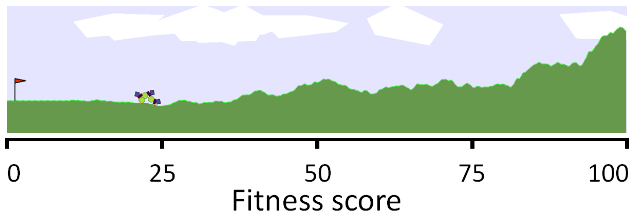


Figure 2: **Box2D environment used for experiments.** Note that the flag is not scaled.

## Evolutionary Algorithm

The implemented EA came from the DEAP (Fortin et al., 2012) framework. This EA consisted of a $(\lambda, \mu)$ (Eiben et al., 2003) strategy—we used tournament selection (tournament size 4) as our offspring selection mechanism, and a generational replacement operator (Floreano and Mattiussi, 2008) where the offspring population replaces the parent population completely. It is known that this generational replacement operator can lead to noisy evolutionary progressions, though we used this type of operator since we noticed a dramatic decreased morphological convergence when compared to more elitist approaches (Veenstra et al., 2019). The mutation operators for our encodings was divided between morphological mutations and controller mutations. The mutation rates determined the probability with which respective parameters were mutated. A fixed sigma value was implemented for adjusting the parameters of the genotype.

For finding suitable parameters for evolving our virtual creatures, we ran a mutation rate sweep of 256 evolutionary runs for each encoding with different mutation parameters. We used a combination of 8 different parameters for the controller mutation rate, and the morphological mutation rate. The evolutionary runs of the sweep were simulated for 500 generations, using a population size of 100 (50,000 evaluations). We ran 4 duplicates with different random seeds for each unique combination of parameters. From this sweep, we chose parameters that we set for 20 duplicates of longer evolutionary runs for each encoding (100,000 evaluations per duplicate; population size of 100; each duplicate again using a different random seed). The parameters adjusted for the experiments can be found in Table 1.

## Creating robots from blueprints

All encodings used for our experiments resulted in a common blueprint format, a directed tree. This blueprint was interpreted by the Box2D environment into both the morphology and controller of the virtual creature. The different encodings that create these blueprints each handled mutations differently. Mutations in the direct encoding directly affect the directed tree. The mutations in the L-System only affect the rewrite rules. The mutation operators in the generator networks affected how the network topology, activation

Table 1: **Experimental settings**. The values within parenthesis were only used for the parameter sweep.

| | Shared parameters | | | |
|---|---|---|---|---|
| Evaluations | 100,000 | | (50,000) | |
| Duplicates | 20 | | (4) | |
| Population size | 100 | | | |
| Mutation Sigma | 0.2 | | | |
| | Direct | L-system | CPPN | CE |
| Mutation rate | 0.32 | 0.16 | 0.02 | 0.08 |
| Morph. mut. rate | 0.16 | 0.04 | 0.02 | 0.08 |

functions, and weights were altered. For each encoding, we supplied a list containing eight modules that served as template modules. For the indirect encodings, this list was mutated, which altered the controller and morphology of each module in the L-System, and just the morphology in the generator networks. We limited the depth of the directed tree to 8 nodes, and set the maximum number of modules that could be created to 20 for one robot.

Each node in the tree contained information about (1) the module type, (2) the morphological parameters associated with the module type, and (3) the controller. Any modules that happened to be located below the terrain when the tree was intepreted, were removed from the virtual creature. Hence, it is not guaranteed that nodes from the blueprint were expressed in the phenotype. The following sections illustrate how each encoding is used to create the blueprints.

**Direct Encoding** The direct encoding manipulates the directed tree that forms the blueprint of the robot. The mutation operators used for the direct encoding were: (1) *remove node* (and sub-nodes), (2) *add node*, and (3) *mutate controller*. When mutating a direct encoding genome, we looped through each node and evaluate whether these mutation operators should be called for individual nodes. To ensure that we add, on average, the same number of modules in a mutation step, the probability of adding a module was divided by the total number of nodes already present in the tree. The *add node* mutation operator is called with a probability for each available connection site on a node. Since the *remove node* mutation removes all subnodes as well, the probability of this mutation happening was set half the probability of an add node mutation.

**L-System** The L-System implementation is based on a parametric L-System (Lindenmayer and Jürgensen, 1992), where the grammar rewrite rules act directly on symbols defined in our alphabet that reference the parameters stored in nodes. This approach is similar to a CE that has been used as a rewrite system for directed graphs (Gruau, 1993). In our case, since we used two different module types, we added four symbols for each module type to which we associated specific rewrite rules. This means that eight rules in total

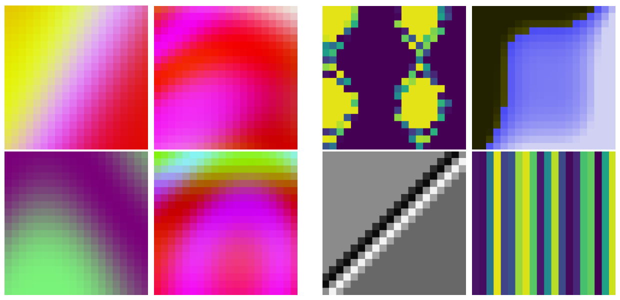

(a) CPPN (b) Cellular Encoding

Figure 3: **Typical RGB output values after randomly initializing the CPPN and the CE.** For both networks, the x and y coordinates were used as inputs to the network.

were employed for generating the entire robot's morphology and controller. We only allowed the tree to expand by adding new leaf nodes based on their associated rules—nodes that were created did not change. Each symbol also contains a reference to a specific controller. Therefore, every similar symbol expressed in the tree contains a deep copy of the same controller. The controller mutation operator changed the controllers stored in the symbols. The morphological mutations changed the rewrite rules directly associated with each symbol.

**Neuroevolution** The generator networks were created through a direct and indirect representation. The direct representation utilizes a CPPN (Stanley, 2007). The CPPN has originally been created to work in conjunction with Neuroevolution through Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002), which contains an evolutionary algorithm. However, to isolate the results of the encoding itself, we decided to use DEAP's standardized evolutionary algorithms instead.

The indirectly encoded generator network was created through a CE (Gruau et al., 1994). To illustrate how each encoding results in different expression patterns, we illustrate how these networks can generate 2D images by having the x and y coordinates of a screen as its inputs, and RGB values as its outputs (Figure 3b). The figure shows that the CPPN creates smoother gradients when compared to the CE. To clarify, the CE and CPPN encodings both create compositional pattern producing networks (the generator network) but we would like to emphasize that with CE and CPPN we refer to the encoding, not the network.

For creating virtual creatures, we queried the networks iteratively. For a query, we gave each network three inputs of a single available connection site. The three inputs were, (1) the current depth of the connection site normalized between values of -1.0 to 1.0, (2) the parent module index from the module list (the 8 template modules) normalized between -1.0 and 1.0, and (3) the value of the angle of the connection

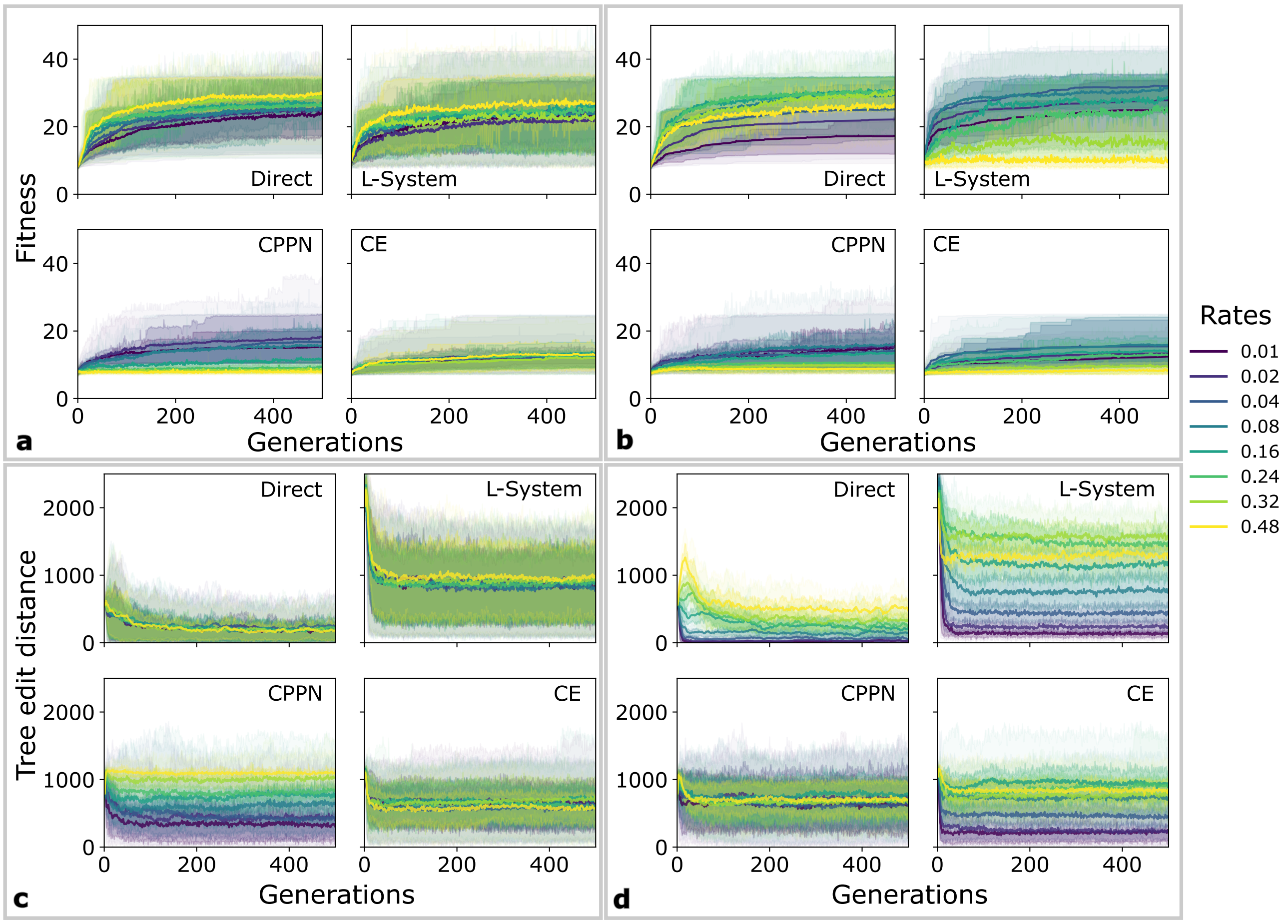


Figure 4: **Results of the mutation rate sweep.** The accumulative maximum fitness values are presented for each encoding. The top two figures show the fitness progression influenced by the mutation rate (a) and the morphological mutation rate (b) for each encoding. The bottom two figures display how the TED changes during the evolutionary run as influenced by the mutation rate (c) and the morphological mutation rate (d). The legend indicates the values of either the controller mutation rate (left) or the morphological mutation rate (right).

site (left = -1.0, top = 0.0, and right = 1.0). The network in turn produced 6 outputs determining: (1) whether a module should be attached or not, (2) which module should be connected, and (3) four outputs determining the parameters of the controller of the module (amplitude, phase, frequency and offset). As mentioned, the morphology of the modules is changed in the supplied template list separately and the generator networks' second output determined which module from this template to use.

**CPPN** The CPPN was based on the implementation from *neat-python* (McIntyre et al., 2019). We altered the CPPN through simply calling the mutation operator build in the library. We set the values of the $conn\_add\_prob$, $conn\_delete\_prob$, $node\_add\_prob$, $node\_delete\_prob$, and $bias\_replace\_rate$ equal to the morphological mutation rate parameter that we used since these attributes alter the topology of the network. The controller mutation rate altered $activation\_mutate\_rate$, $weight\_replace\_rate$, $weight\_mutate\_rate$ and $bias\_mutate\_rate$ in a similar manner. The sigma value we use was set to alter $weight\_mutate\_power$, $bias\_mutate\_power$ and the $response\_mutate\_power$. To the best of our knowledge, this was a decent mapping from our mutation rate parameters to mutation settings of the CPPN.

**Cellular Encoding** The cellular encoding used to create the generator network was based on Gruau et al. (1994). In this encoding, a network was generated by initially having a single cell (or node) connected to an input and output node. This cell is then divided, according to division rules, into more cells through either sequential or parallel division. The rules we implemented for generating this network were (1) change the node type, (2) divide sequentially, and (3) divide in parallel. The weights and activation functions were attached to the rules determining how the division happened.

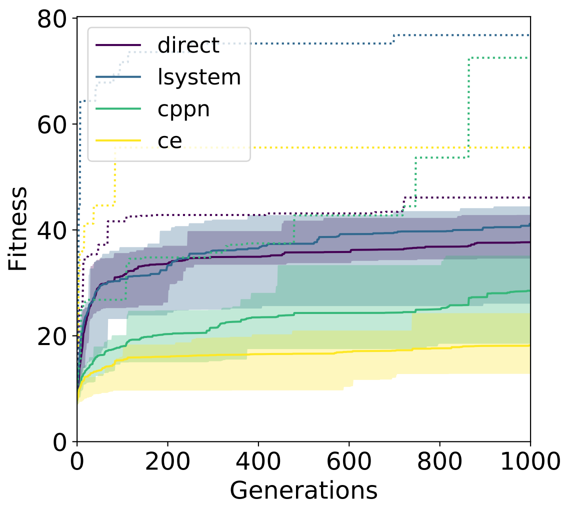


Figure 5: **Evolutionary progression of all encodings.** The shaded area represents the 25th to 75th percentiles. The dotted line shows the maximum fitness of the best run.

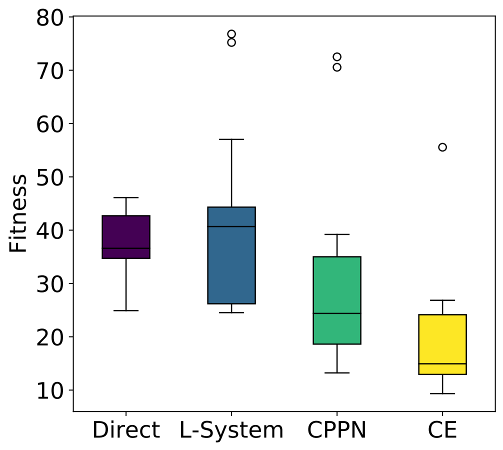


Figure 6: **Boxplot of the accumulated best fitness for each encoding.**

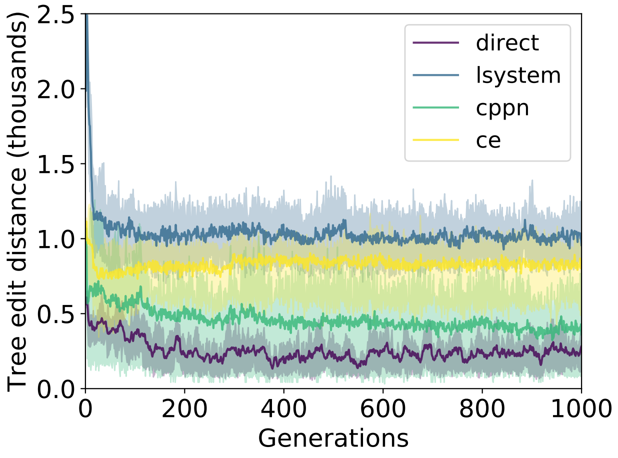


Figure 7: **TED measured over generations**. For each encoding, the lines show the cumulative TED. The shaded area represents the 95% confidence interval.

We limited the number of node that could be produced by the CE to 50. This limitation was set as more nodes led to a computational bottleneck. We limited the number of rewrite rules the cellular encoding could use to 10. Each node object contained information about the type of node, and the weights of the node's output edges.

To generate the network, we iterated 10 times over the nodes present in the network to determine how each node should be rewritten. This process was terminated early if the maximum number of allowable nodes was reached. A node could also contain any activation function that were also available to the CPPN.

## Results

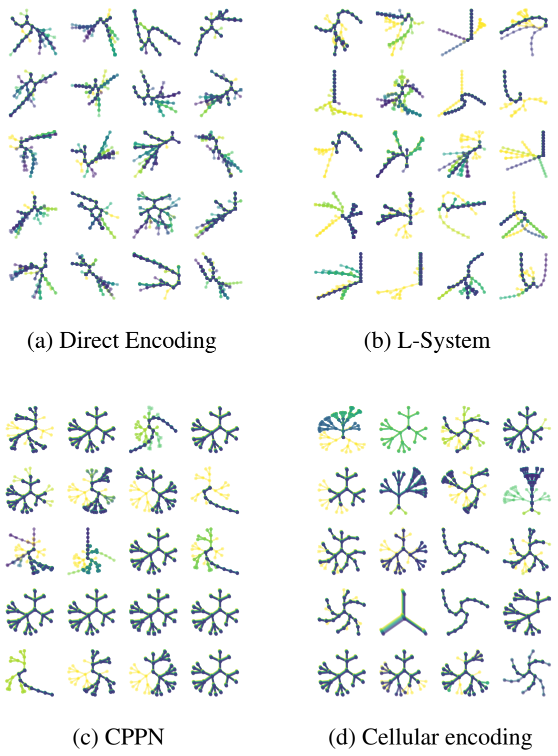
**Mutation rate sweep** The performance of each encoding differs greatly with regards to the parameters we set. As can be seen from Figure 4, the morphological mutation rate parameter has a bigger effect on the L-System than on the Direct encoding. The diversity measured by the TEDs within a population was also greatly affected by the mutation parameters we set (Figure 4). Since this distance metric only measures morphological parameters of the tree blueprint, we can see that the controller mutation rate did not affect the TEDs in the direct encoding, L-System or CE, whereas it did affect the CPPN. Similarly, the morphological mutation rate parameter has the opposite effect—the CPPN was not affected whereas the other encodings were. Based on these results, we chose the parameters for the experimental runs that can be seen in Table 1.

**Performance analysis** The performance of all runs for each encoding is presented in Figure 5, and the difference in performance between the accumulated fitness values is shown in Figure 6. For comparing any significant differences, we performed six two-sided Mann-Whitney U tests. There was no significant difference between the direct encoding and the L-System (p-value $> 0.2$), but there was between the direct encoding and the CPPN (p-value $< 0.001$), and the direct encoding and the CE (p-value $< 0.001$). Similarly, the L-System was significantly different from the CPPN (p-value $< 0.001$) and the CE (p-value $< 0.001$) as well. Furthermore, the CPPN and cellular encoding also showed a significant difference in performance (p-value $<$ 0.002). We also observed that the performance of the direct encoding and CPPN is less spread out across runs whereas the L-System and the cellular encoding produced more outliers (e.g. see the dotted lines in Figure 5).

**Diversity analysis** The diversity based on the TED measured within a population over time is shown in Figure 7.

Here, the L-System seems to maintain the highest TED in the population followed by the CE, CPPN and direct encoding respectively indicating that the indirect encodings keep a larger morphological diversity within the population. Figure 8 displays how the blueprints of the elites of each run change over time. From these images it can be qualitatively seen that elites differ most from one another in the direct encoding and L-System, whereas elites seem to converge quicker in the network generator approaches.

The tree structures depicted in Figure 8 were also plotted for all individuals taken at intervals of 20 generations. Since each node in the representation has a specific positional coordinate when plotted, we summed up the x and y coordinates of each position of each node to get a new abstract positional coordinate (note that different trees can map to the same coordinates). Through rasterizing these values, a map of elites, similar to visualizations shown in MAP-Elites implementations Mouret and Clune (2015), can be constructed. Figure 9 depicts the cumulative fitness score for each individual of each encoding in every morphological solution space that was found. From this figure one can see that the direct encoding spreads out locally, individuals in an area having roughly the same fitness. The L-System seems to jump across the morphological space where high fit solutions are rarely found in clusters. The generator networks jump across the search space in a similar arbitrary manner, though not many unique morphological solutions were found.


(a) Direct Encoding (b) L-System

(c) CPPN (d) Cellular encoding

Figure 8: **Blueprints of elites of plotted across generations.** The blueprints are shown across generations (from light yellow [generation 0] to dark blue [generation 1000]).

## Discussion

In this article we looked at how different methods can be used as developmental abstraction for the genotype-phenotype mapping when evolving the morphology and control of 2D virtual creatures. Performance wise, we saw that the direct encoding and L-System outperformed the implemented generator network strategies based on fitness. In addition, our results indicate that the encoding strategies greatly affect how the solution space is traversed. Mainly, the direct encoding, traverses the morphological solution space more locally (Figure 9) whereas the indirect encodings make larger jumps across this search space.

Other types of neural networks, or different inputs and output definitions, can likely improve the performance of the generator networks. However, the purpose of this implementation is also to start considering a unified approach for morphogenetic strategies that result in directed trees. Different encoding types can hereby contribute to the exploration vs exploitation trade-off. Users familiar with neural networks can incorporate any type of neural network for optimizing these directed trees, which potentially lead to interesting designs and aid us in understanding artificial developmental processes.

A challenge of the generator network approaches for creating robot morphologies is concerned with tuning the many adjustable parameters. Although we have tuned some of these parameters through the parameter sweep, it is unknown whether the decreased performance is related to the plethora of parameters or perhaps the mapping of the input and outputs of the networks. In contrast, the L-System only contains a few rewrite rules and has the least parameters to optimize, making it easy to implement. The limitation of the L-System is that controllers cannot be fine-tuned if they are of the same type. This limitation does create a few interesting points of potential improvement for the L-System. The encoding could be improved by, for example, allowing for the dynamic splitting of rewrite rules over time. One rule can be split up into two sub-rules that can subsequently enable fine-tuning of individual modules. Since we used open-loop controllers defined by sine wave generators, an additional improvement for all encodings could be a sensory feedback loop that adjusts the controllers dynamically or online (Sproewitz et al., 2008; Moreno and Gomez, 2010; Nordmoen et al., 2019).

Since all virtual creatures created for this project went through an intermediate directed tree blueprint, this blueprint can also be transformed back into a direct encoding. This adds the possibility of incremental evolution and hybrid encodings. One could for example start evolving a virtual creature using an indirect encoding and at some

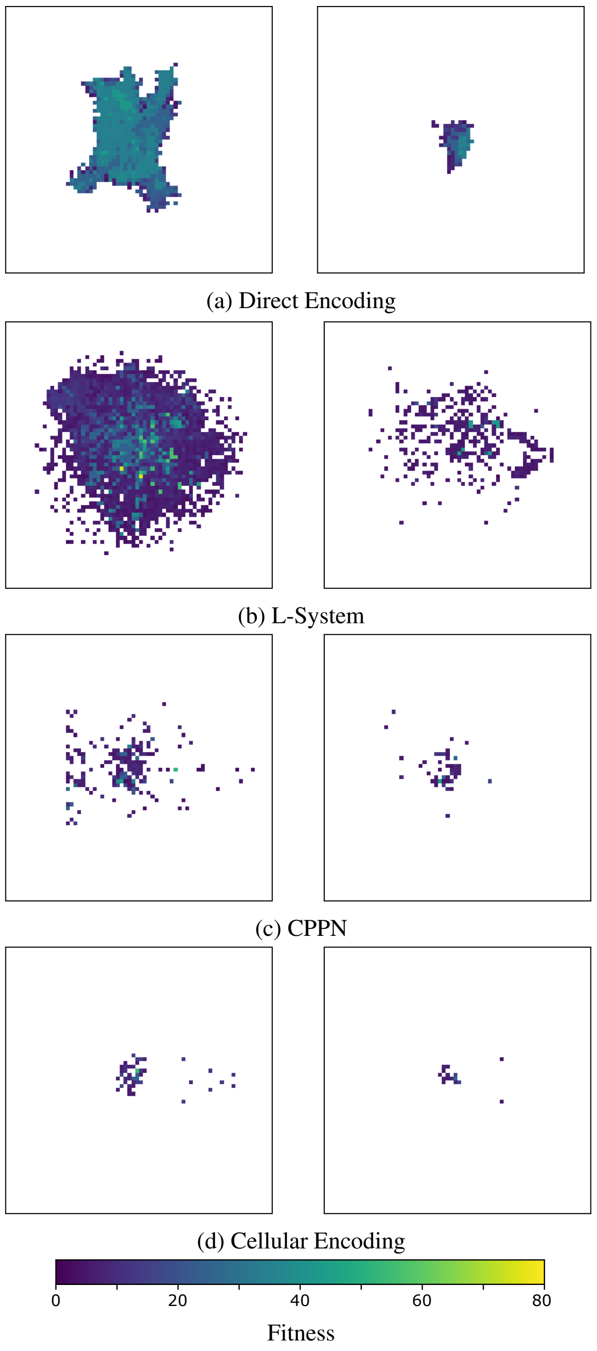


Figure 9: **Heat maps showing how each encoding explores the search space.** Figures on the left show the cumulative exploration for each encoding. Figures on the right show the exploration of a single run.

point, during optimization, switch from indirect encoding to direct encoding. This could allow for individuals that jumped across the search space to be fine-tuned, which is difficult to implement using the original indirect encodings. Since there is a big difference in how the encodings traverse the search space, it would be interesting to see what effect diversification mechanism for EAs would have on the performance of the encodings presented.

We are unsure whether the performance of various strategies in this 2D platform will translate well to 3D problem spaces, and we would like to investigate this in the future by deploying the same methods using 3D testbeds such as presented in James et al. (2019); Coumans and Bai (2017). Another interesting method could hereby be to simulate various environments similar to Wang et al. (2019), where these environments could include 3D simulators. Through quick explorative experiments done using the 2D testbed, more complex 3D environment can be used when indicative hyperparameters are set. Using directed trees also enables the creation of more realistic morphology changing simulated robots (Alattas et al., 2018).

With the presented testbed and encoding results, we hope to have given some insights in how morphological evolution for virtual creatures is shaped by encoding strategies. A potentially unifying approach for using these strategies in the OpenAI gym environment can ease the process of evolving both the morphology and control of simple virtual creatures. In addition, it can enable the exploration of genotype-phenotype mechanisms for increasing performance in artificial system, and potentially gain insight in developmental strategies seen in natural systems.

## Conclusion

Different types of encodings can be utilized to construct virtual creatures. Considering the effectiveness of each the encodings we used, the L-System and direct encoding performed best. We did see that diversification across evolutionary runs differed for each encoding. Namely, the direct encoding tended to explore local areas of the morphological search space in contrast to bigger jumps that were made using the indirect approaches. The testbed introduced in this paper is an OpenAI gym environment for the generation of 2D virtual creatures. This testbed is a computationally inexpensive benchmark for experimenting with morphological evolution of virtual creatures. This method can be useful to get a better understanding in what genotype-phenotype mapping to use for evolving simple virtual creatures. Ultimately, if this understanding can translate to more complex tasks and environments, it can aid in enhance design principles for creating other simulated and physical robots.

## Acknowledgements

This work was supported by the Research Council of Norway through its Centres of Excellence scheme, project number 262762. Computational resources were supplied by UNINETT Sigma2. Special thanks to Jørgen Halvorsen Nordmoen for his valuable contributions to the experiments conducted for this paper.

## References

Alattas, R., Patel, S., and Sobh, T. (2018). Evolutionary modular robotics: Survey and analysis. *Journal of Intelligent & Robotic Systems*.

Auerbach, J., Aydin, D., Maesani, A., Kornatowski, P., Cieslewski, T., Heitz, G., Fernando, P., Loshchilov, I., Daler, L., and Floreano, D. (2014). Robogen: Robot generation through artificial evolution. pages 136–137.

Auerbach, J. E. and Bongard, J. C. (2010). Evolving cppns to grow three-dimensional physical structures. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 627–634.

Auerbach, J. E. and Bongard, J. C. (2011). Evolving complete robots with cppn-neat: the utility of recurrent connections. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1475–1482.

Bongard, J. C. and Pfeifer, R. (2003). Evolving complete agents using artificial ontogeny. In Hara, F. and Pfeifer, R., editors, *Morpho-functional Machines: The New Species*, pages 237–258, Tokyo. Springer Japan.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *CoRR*, abs/1606.01540.

Catto, E. (2019). Box2d, `https://box2d.org/`.

Cheney, N., Bongard, J., SunSpiral, V., and Lipson, H. (2018). Scalable co-optimization of morphology and control in embodied machines. *Journal of The Royal Society Interface*, 15(143):20170937.

Cheney, N., MacCurdy, R., Clune, J., and Lipson, H. (2014). Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. *ACM SIGEVOlution*, 7(1):11–23.

Cheney, N., Sunspiral, V., Bongard, J., and Lipson, H. (2016). On the difficulty of co-optimizing morphology and control in evolved virtual creatures. In *Artificial Life Conference Proceedings 13*, pages 226–233. MIT Press.

Coumans, E. and Bai, Y. (2017). Pybullet, a python module for physics simulation in robotics, games and machine learning.

Eiben, A. E., Smith, J. E., et al. (2003). *Introduction to evolutionary computing*, volume 53. Springer.

Faíña, A., Bellas, F., López-Peña, F., and Duro, R. J. (2013). Edhmor: Evolutionary designer of heterogeneous modular robots. *Engineering Applications of Artificial Intelligence*, 26(10):2408 – 2423.

Floreano, D. and Mattiussi, C. (2008). *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175.

Gruau, F. (1993). Cellular encoding as a graph grammar. *IEE Colloquium on Grammatical Inference: Theory, Applications and Alternatives*, pages 17/1–1710.

Gruau, F. et al. (1994). Neural network synthesis using cellular encoding and the genetic algorithm.

Ha, D. (2019). Reinforcement learning for improving agent design. *Artificial Life*, 25(4):352–365. PMID: 31697584.

Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18.

Hiller, J. and Lipson, H. (2014). Dynamic simulation of soft multimaterial 3d-printed objects. *Soft robotics*, 1(1):88–101.

Hornby, G. S., Lipson, H., and Pollack, J. B. (2003). Generative representations for the automated design of modular physical robots. *IEEE transactions on Robotics and Automation*, 19(4):703–719.

James, S., Freese, M., and Davison, A. J. (2019). Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*.

Joachimczak, M., Suzuki, R., and Arita, T. (2016). Artificial metamorphosis: Evolutionary design of transforming, soft-bodied robots. *Artificial Life*, 22(3):271–298. PMID: 27139940.

Klimov, O. (2019). bipedal walker, `https://github.com/openai/gym/blob/master/gym/envs/box2d/bipedal_walker.py`.

Kriegman, S. (2019). Why virtual creatures matter. *Nature Machine Intelligence*, 1(10):492–492.

Lehman, J. and Stanley, K. O. (2011). Novelty search and the problem with objectives. In *Genetic programming theory and practice IX*, pages 37–56. Springer.

Lindenmayer, A. and Jürgensen, H. (1992). Grammars of Development: Discrete-State Models for Growth, Differentiation, and Gene Expression in Modular Organisms. In Rozenberg, G. and Salomaa, A., editors, *Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*, chapter 1, pages 3–21. Springer Berlin Heidelberg, Berlin.

Lipson, H. and Pollack, J. B. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974.

Lund, H. H., Hallam, J., and Lee, W.-P. (1997). Evolving robot morphology. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 197–202. IEEE.

McIntyre, A., Kallada, M., Miguel, C. G., and da Silva, C. F. (2019). neat-python, `https://github.com/CodeReclaimers/neat-python`.

Miras, K., Haasdijk, E., Glette, K., and Eiben, A. E. (2018). Search space analysis of evolvable robot morphologies. In *Applications of Evolutionary Computation - 21st International Conference, EvoApplications 2018*, volume 10784 of *Lecture Notes in Computer Science*, pages 703–718. Springer.

Moreno, R. and Gomez, J. (2010). A hybrid control strategy for a chain type modular robot. *Modular Robots: The State of the Art*, page 111.

Mouret, J.-B. and Clune, J. (2015). Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology,Intelligence,and Technology*. MIT Press, Cambridge, MA, USA.

Nordmoen, J., Nygaard, T. F., Ellefsen, K. O., and Glette, K. (2019). Evolved embodied phase coordination enables robust quadruped robot locomotion. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 133–141. ACM.

Pfeifer, R. and Bongard, J. C. (2006). *How the Body Shapes the Way We Think: A New View of Intelligence (Bradford Books)*. The MIT Press.

Sims, K. (1994). Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, page 15–22, New York, NY, USA. Association for Computing Machinery.

Sproewitz, A., Moeckel, R., Maye, J., and Ijspeert, A. J. (2008). Learning to move in modular robots using central pattern generators and online optimization. *The International Journal of Robotics Research*, 27(3-4):423–443.

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Stanley, K. O. and Miikkulainen, R. (2003). A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130.

Stoy, K., Brandt, D., and Christensen, D. (2010). *Self-Reconfigurable Robots: An Introduction*.

Turing, A. (1952). The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society B*, 237:37–72.

Veenstra, F., Faina, A., Risi, S., and Stoy, K. (2017). Evolution and morphogenesis of simulated modular robots: a comparison between a direct and generative encoding. In *European Conference on the Applications of Evolutionary Computation*, pages 870–885. Springer.

Veenstra, F., Hart, E., Buchanan, E., Li, W., Carlo, M. D., and Eiben, A. E. (2019). Comparing encodings for performance and phenotypic exploration in evolving modular robots. In *GECCO '19*.

Wagner, G. P. and Altenberg, L. (1996). Perspective: Complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. (2019). Paired open-ended trailblazer (POET): endlessly generating increasingly complex and diverse learning environments and their solutions. *CoRR*, abs/1901.01753.