Evolution and Morphogenesis of Simulated Modular Robots: A Comparison Between a Direct and Generative Encoding

Frank Veenstra, Andres Faina, Sebastian Risi and Kasper Stoy

IT University of Copenhagen, Denmark
{frve, anfv, sebr, ksty}@itu.dk

Abstract. Modular robots offer an important benefit in evolutionary robotics, which is to quickly evaluate evolved morphologies and control systems in reality. However, artificial evolution of simulated modular robotics is a difficult and time consuming task requiring significant computational power. While artificial evolution in virtual creatures has made use of powerful generative encodings, here we investigate how a generative encoding and direct encoding compare for the evolution of locomotion in modular robots when the number of robotic modules changes. Simulating less modules would decrease the size of the genome of a direct encoding while the size of the genome of the implemented generative encoding stays the same. We found that the generative encoding is significantly more efficient in creating robot phenotypes in the initial stages of evolution when simulating a maximum of 5, 10, and 20 modules. This not only confirms that generative encodings lead to decent performance more quickly, but also that when simulating just a few modules a generative encoding is more powerful than a direct encoding for creating robotic structures. Over longer evolutionary time, the difference between the encodings no longer becomes statistically significant. This leads us to speculate that a combined approach – starting with a generative encoding and later implementing a direct encoding – can lead to more efficient evolved designs.

Keywords: Modular Robots, Evolutionary Algorithms, Direct & Generative Encodings

1 Introduction

Evolutionary Robotics has covered a vast amount of research on the automated design of Robotic entities via artificial evolution [1, 2, 3]. To rapidly explore different robotic morphologies and control systems that can be physically assembled in the real world, robotic modules are useful as evolutionary building blocks. A robotic module being an independent unit that encapsulates part of its functionality [4]. This encapsulation is important for the (re)configuration of modular robot compositions. In contrast to static robotic entities, modular robots can be reconfigured enabling researchers to quickly explore different morphologies. However, it is difficult to design a representation of the genotype to

phenotype mapping of a modular robot and we can either evolve all parameters of every simulated module or reuse parts of the genome to construct and control a modular robot. The latter approach – a generative encoding – would require a smaller genome and could in turn evolve decent morphologies and control more quickly. In contrast, evolving all parameters of every robotic module enables us to fine tune behavioral parameters but also increases the search space.

In nature, most multicellular organisms develop from a *zygote* [5]. The zygote and its genome comprises the developmental representation of the organism [6]. The resulting developmental process allows for the reuse of genes which can give rise to recursive structures in the phenotype. Computational models representing an artificial organism's phenotype either use a direct or generative encoding (also indirect encoding). A direct encoding constitutes a one-to-one mapping of genotypic components into the phenotype meaning that the genes encode for every simulated module. In contrast, generative encodings – similar to the development of an organism from a zygote – reuse elements of the genome for constructing the phenotype. Generative encodings have a smaller genotypic state space due to this reuse of genes.

Since the morphological search space in modular robots is limited to the amount of connection sites available on each module, encodings that directly map the assembling process of modular building blocks have been implemented for the generation of robot morphologies [7, 8, 9]. Usually, these direct encodings implement an additional symmetry operator that increases the effectiveness of artificial evolution. Simple generative encodings [10, 11, 12, 2, 13, 14, 15, 16] have been shown to quickly lead to useful robot morphologies. It is, however, unclear whether designing platforms that evolve modular robot morphologies should rather use a direct or generative representation and if the generative encoding is still useful when just a few modules are being used. A generative encoding should no longer have an advantage if the genetic state space in both encodings is of similar size since the amount of mutable parameters are equal.

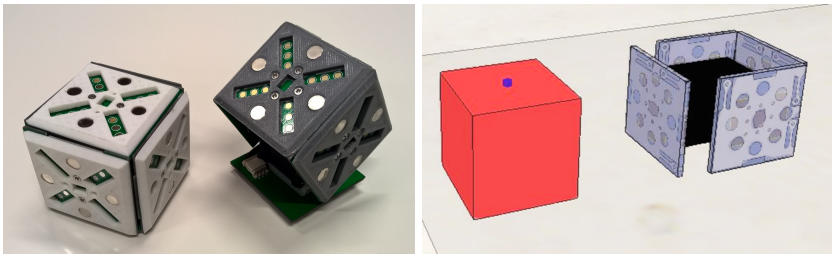
In this paper we investigate whether a generative encoding or a direct encoding is more useful for evolving modular robots for locomotion. Both encodings make use of evolutionary algorithms to optimize the simulated robots. The direct encoding utilizes the 'Evolutionary designer of heterogeneous modular robots' (Edhmor; see section 2.2)[8] system. The generative encoding is based on a parallel rewriting system called a Lindenmayer System (L-System) [17]. As mentioned earlier, the size of the search space of the direct encoding grows exponentially considering the amount of mutatable parameters (in our case the amount of modules) while the search space of the generative encoding stays roughly the same size. The ability of the direct encoding to mutate parameters of individual modules enables more local improvements. In contrast, since small mutations in the generative encoding can lead to drastic phenotypic changes, the generative encoding might be more prone to stagnate in local optima. Though the scope of this paper does not encompass transferability, the implemented encodings serve as a stepping stone towards evolving feasible modular robotic entities in reality.

2 Methodology

Many modular robotic systems make use of central pattern generators for controlling the modules [18, 19, 15]. These central pattern generators are derived from their natural equivalents seen in biology [20, 21]. The implementation of modifiable central pattern generators seems a logical step towards evolving modular robots, however, we think that this convolutes the search space of the evolutionary system unnecessarily for the aim of this paper. To still achieve a patterned output in our modular system, sinusoidal functions control each module individually in a decentralized manner. By fixing the morphological parameters of the simulated modules and limiting the control parameters of the modules to sinusoidal functions, we are able to analyze how the different encodings can be implemented for evolving robotic structures. For evolving simulated robot morphologies, two evolutionary platforms were used to evaluate the direct and the generative encoding. Both platforms employ the robotics simulator 'Virtual Robot Experimentation Platform' (V-REP; version 3.32) [22]. The next sections will discuss the common elements as well as the differences for each platform.

2.1 Common Elements for both Platforms

Both encodings simulate the exact same modules modeled in V-REP. A cube module and a servo module were designed for the platforms. The modules are based on earlier designs of physical modules (figure 1a; developed at the IT University of Copenhagen). In turn, the simulated modules (figure 1b) are modeled according to the physical properties of these modular units. The real modules can be attached to one another via magnets and the breaking force and torque parameters resulting from these connections is modeled in the simulated modules.



a b

Fig. 1. Illustration of the modules created in the real world (a) and in the simulator (b).

The modules contain male and female connection sites that enable the modules to connect together. The connections are modeled with a force sensor in

V-REP. If the force on a connection site exceeds 1.7 N m of torque or 80.0 N of force, the force sensor between the modules breaks leading to the fragmentation of the morphology. 10 consecutive threshold violations for the force sensor had to be registered before a connection could break.

The cube module (dimensions x,y,z is 55mm,55mm,55mm; weight is 100g) is used as an initial building block for the modular robot to which other modules are attached. This cube has five female conection sites (top, right, left, front, back). The servo module (dimensions x,y,z is 55mm,55mm,80mm; weight is 160g) has three female connection sites (top, right, left) and one male attachment site (bottom). The bottom male connection site of the servo module is thus able to connect to any of the female connection sites of other cube or servo modules.

The joint of the servo module implements a PI controller (P is 0.1 and I is 0.01) and could exert a maximum torque of 1.5 N m. A sinusoidal wave function controls the position of the joint in the servo module. The maximum amplitude of the sinusoidal wave function ranged from -90°and +90°degrees from its original position. The offset, phase and amplitude of the sinusoidal function are mutable parameters. When a new module is added to the simulation, only the male connection site of the new module can be connected to any female connection site of the robot. The new module has four different orientations in which it can attach to a new connection site (note that the amount of orientations is a bit different in the direct encoding: section 2.2). The servo modules implemented the default simulation material while the cube module used the "rest_stack_grasp_material" as material types simulated by V-REP.

The goal of the simulated robots was to move as far away from its initial position in a horizontal direction as possible within 20 seconds of simulation time. This distance is measured by the horizontal distance that the initial cube module has traveled. Before starting a simulation in V-REP, modules are joined together to form a robot morphology. The entire robot is then shifted upwards so that its lowest point is 0.1 millimeter above the simulated ground. To take into account the movement due to the robot simply falling over, the distance traveled in the first 2.5 seconds of the simulation is discarded. An additional cost function was added to compensate for modules that were disconnected due to the breaking of a connection site. The fitness value of each individual is directly correlated to the horizontal distance traveled multiplied by the amount of connections broken between the modules to the power 0.8 and can be derived from equation 1.

$$F = \sqrt{(p_e x - p_1 x)^2 + (p_e y - p_1 y)^2} * \eta^{0.8} \quad (1)$$

Where F represents the fitness value obtained by calculating the eventual position (p_e) minus the position after 2.5 seconds (p_1) traveled in both x and y directions. η represents the amount of broken module connections of the morphology after 20 seconds of simulation time.

A simulation environment consisted of a default floor and was simulated using the bullet dynamics engine (version 2.78). The dynamics settings were set to accurate (default) with a time-step of 50ms. Six experiments were done comparing the different encodings. Three of the experiments ran twelve evolutionary

runs whereby a maximum of 5, 10 or 20 servo modules and one cube module were allowed. These three experiments were done to see how the direct encoding performed. The other three experiments analyzed the efficiency of the generative encoding and was also composed of twelve evolutionary runs simulating a maximum of 5, 10 or 20 modules. The runs are limited to a fixed amount of evaluations. In the simulations that could simulate a maximum of five modules, 12,500 evaluations were done. The other runs were limited to 25,000 evaluations; more evaluations were performed in these runs since the search space is larger when increasing the amount of simulated modules. 25,000 evaluations were chosen as a trade-of between performance and computational requirements. Since a high end physics simulator is used, the computational requirements are considerable. The next sections will cover the direct and indirect encodings in more detail.

2.2 Direct Encoding

The 'Evolutionary designer of heterogeneous modular robots' (Edhmor)[23, 8] system is used as the direct encoding strategy to assemble and evaluate robot morphologies. The Edhmor system is organized as a tree representation, where nodes represent control parameters of a module and its type and edges represent how a module is attached to a parent module. The direct encoding is used together with a constructive algorithm. This algorithm starts building a random population of robots with just a few modules. Afterwards, different mutation phases are applied cyclically. The mutation phases of the algorithm are:

- *Add Module:* Add a module into a morphology.
- *Mutate morphology:* Change the orientations or the place where some modules are connected
- *Mutate control:* Change the control parameters of some modules
- *Prune robot:* Test all the morphologies generated by removing a module and its children.

In every phase, a mutation operator is applied several times to produce different random mutations of the same individual which are tested in the simulator. For example, when adding a new module to a robot, five different robots are generated and each of them have a new module placed in different positions and orientations. These phases revert to the previous robot if the mutation does not increase the fitness of the robot, except in the add module phase. This phase forces morphological evolution to take place which has been shown to be advantageous when evolving virtual creatures [24].

The evolutionary algorithm of Edhmor is furthermore generational, the 10% worst performing robots are removed from the population every cycle. Half of them are replaced by random robots with a low number of modules, the other half is generated by applying symmetry operators to the best robots. The symbolic representation and its phenotype are depicted in figure 2. A more detailed overview of the system can be found in [8].

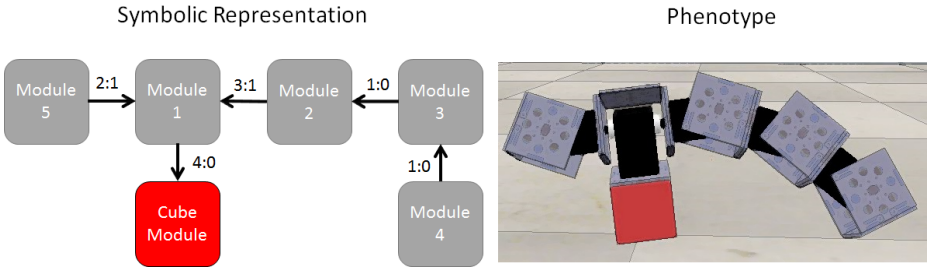


Fig. 2. Representation of the direct encoding. (left) Symbolic representation of the direct encoding: each rectangle represents a module and each arrow represents a connection between modules. There are two numbers for each connection, which indicate the face of the parent where the child node is attached and the orientation of the child node. (right) The symbolic representation of the direct encoding that encodes for a phenotype.

2.3 Generative Encoding

The implemented generative encoding is based on a context sensitive Lindenmayer-System (L-System) [17, 25] – a parallel rewriting system. In our case, the variables used in the L-System represent the modules employed to construct a robot (figure 3), similar to [26]. In the simulation environment, each variable represents a specific module *state* which encompasses all the parameters of the morphology, control and attachment rules inherent to a module state. The genome of the generative encoding is thus composed of a fixed amount of module states predefined before an individual is generated and evaluated in the simulation environment. The relevant genetic parameters of the module state are used to create a new module. These genetic parameters include the attachment rules for modules. The attachment rules of the cube module included the information of which module is connected to what connection site and in which orientation. The same attachment rules are possible in the servo modules but the servo modules only contain three attachment sites. The implemented attachment rules are in essence similar to the rewriting rules of a normal context sensitive L-System [27]. It is context sensitive since a module cannot be placed at an attachment site if another module is already occupying it. Furthermore, modules cannot be created if this causes a collision with other created modules.

The generative encoding was limited to using five different module states. The first state (the axiom) represents the cube module and the four other states represent the servo module. The four states that represent a servo module encode for the same module but can differ in their mutable parameters responsible for the sinusoidal function that controls the servo module. The internal sinusoidal function that controlled the PI controller of the modules could be mutated in the genome of the module states. This means that the robot can actually not have more than four distinct sinusoidal controllers. For illustrating the different object states, they are colored in the phenotype. The modules could either be

red, yellow, blue or pink depending on their state. Four iterations of the L-System were done to create the robot phenotypes starting with the cube module as the axiom.

All parameters of the module states were subject to evolution. There was a 15% chance of a morphological parameter to be mutated and a 5% chance of a control parameter to be mutated. A symmetry mutation operator enabled an object state to arise at the opposite site of a module where it originally was expressed. Though symmetry is an inherent trait to an L-System, the symmetry operator enhanced the probability of creating symmetrical phenotypes. Since the genome of an individual is represented by different module states, a crossover operator enabled different states to be exchanged between individuals. The crossover function had a 20% chance that a module state of an individual came from a different individual than its original parent.

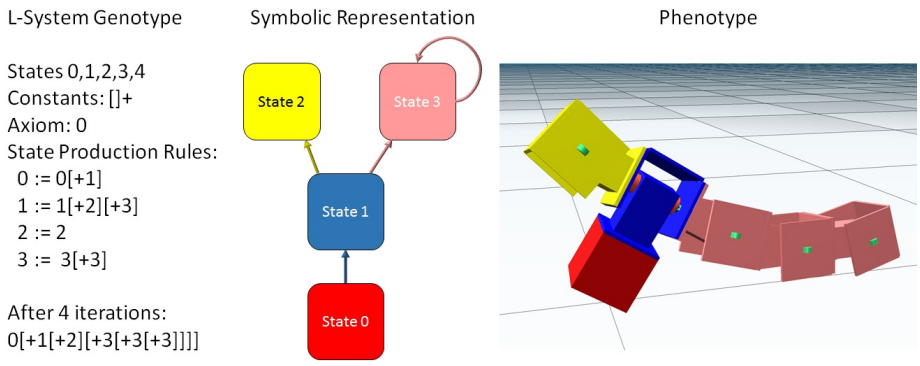


Fig. 3. Representation of the generative encoding. (left) The L-System parameters form the genotype of the morphology whereby the variables of L-Systems are replaced by module states. The '+' constant represents the placing of the next module at the specific attachment sites of a module. The symbolic representation of the genotype (middle) serves as a visualization on how the genotype constructs the phenotype (right).

3 Results

The results of the different evolutionary runs were divided in a performance analysis and a phenotype analysis. The performance analysis was done to get a clear insight in the efficiency of the encodings. Knowing a bit of what type of phenotypes resulted from the evolutionary runs gives us more insight in what prominent evolved characteristics were and how we can ultimately improve the simulator for the design of actual modular robots.

3.1 Performance analysis

As can be seen in figure 5, the average fitness values – as well as the averages of the maximum fitness values – of the evolutionary runs is quite different per encoding. The generative encoding seems to be able to quickly find decent behaviors that are rewarded with a high fitness value. A Mann-Whitney U test has been performed at specified intervals to check whether the encodings performed significantly different. The performance difference was measured using the average fitness values of the maximum fitness of each individual evolutionary run at a specified time. The test resulted in a significant difference between evolved populations after 6,250 evaluations (p-value: 0.000612) and 12,500 evaluations (p-value: 0.003674) when simulating a maximum of 5 modules. There was also a significant difference between the two encodings at 6,250 evaluations (p-value: 0.00328), not at 12,500 evaluation (p-value: 0.0124106) but again at 25,000 evaluations (p-value: 0.001617) when evolving a maximum of 10 modules. The runs of the simulation evolving a maximum of 20 modules was also statistically different at evaluation 6,250 (0.00332) but not at evaluation 12,500 (p-value: 0.177805) and also not at evaluation 25,000 (p-value: 0.209462). The maximum and average fitness values of the individual runs can be seen in figure 4.

3.2 Phenotypes

Different distinct phenotypic behaviors emerged after a certain amount of evolutionary time. The direct encoding evolved various kinds of strategies though the generative encoding had evolved more simple, distinct types of locomotion due to the similarity in behavior seen in several modules. Caterpillar like behavior could be seen evolved conglomerates that were composed of a single chain of modules7b. A single chain of modules could also result in a different type of rolling locomotion 6a, 6b. For some evolved robots there was no apparent logic to how they moved. Two robots tossed their weight around which resulted in complex rolling (figure 6c,7c and 7d) and one robot performed a crawling (figure 7e) behavior. The types of behavior should become more evident when consulting the supplied video [28].

The constructive strategy of the direct encoding has a tendency to add modules to the robot. This results in the best individual of all the different evolutionary runs to be composed of 5 modules when simulating a maximum of 5 modules. In the case that the maximum number of modules is 10, 8 out of 12 runs have reached the maximum number of modules and the average is 9 modules. When 20 modules are allowed, the average is 12.33 with a standard deviation of 2.87. In this experiment, the amount of modules are limited by the fact that the excess of torque breaks the connections between the modules, which are heavily penalized by the fitness function.

All the robots with a maximum of five modules developed similar morphologies, linear structures, with a rolling behavior. One of them is shown in figure 6a. With a limit of 10 modules, branches in the structure of the robots appear.

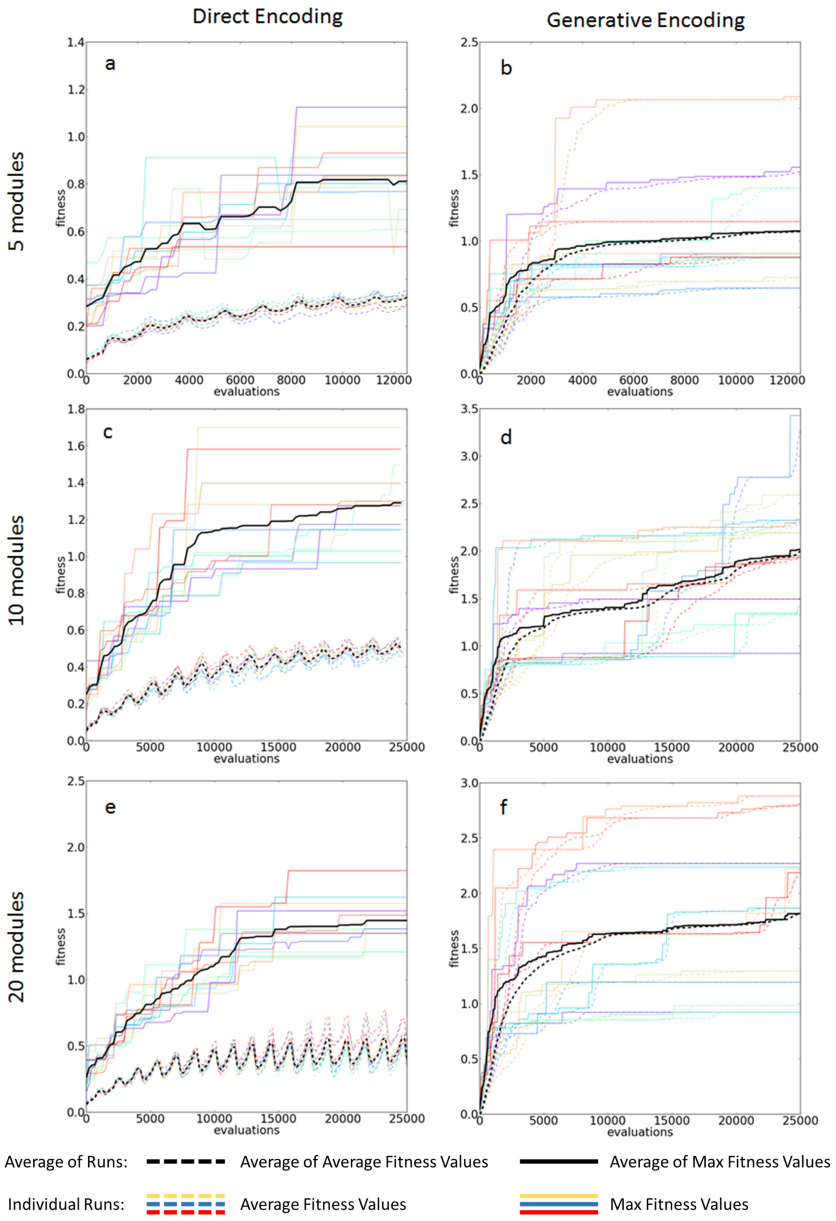


Fig. 4. The graphs represent the individual runs done for each experiment. The six graphs are represent the direct encoding simulating a maximum of 5 servo modules (a); generative encoding simulating a maximum of 5 modules (b); direct encoding simulating a maximum of 10 servo modules (c); generative encoding simulating a maximum of 10 modules (d); direct encoding simulating a maximum of 20 servo modules (e), generative encoding simulating a maximum of 20 modules (f). The bold black line represents the average maximum fitness values for all runs while the black dotted line represents the average of the average fitness values of all runs. The colored lines represent individual runs, where the solid line represents the maximum fitness value of the population and the dotted line the average fitness of the population.

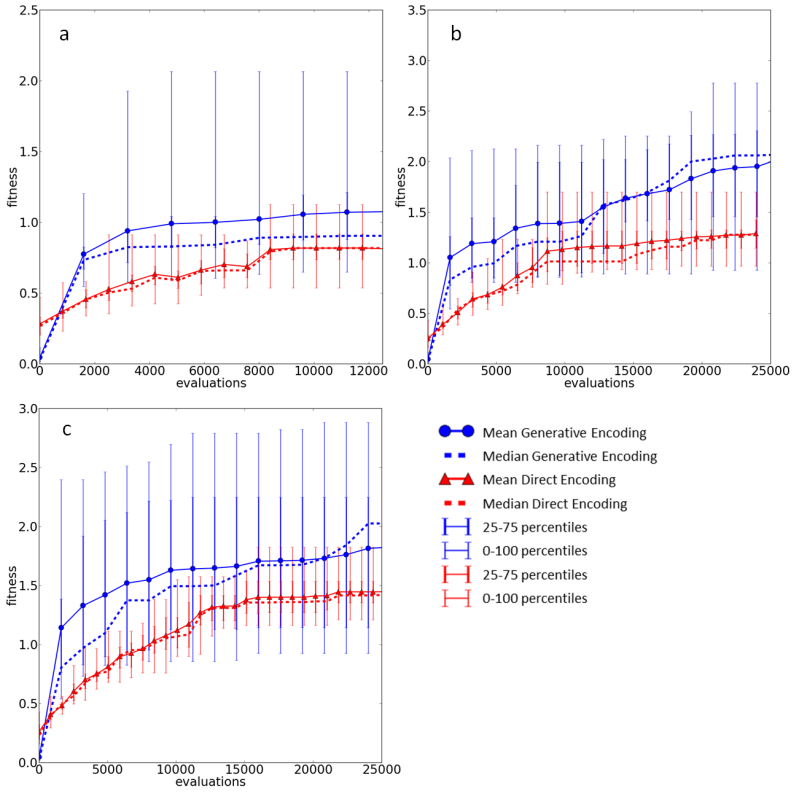


Fig. 5. The graphs display the average maximum fitness values of the different evolutionary runs when simulating a maximum of 5 modules (a), 10 modules (b) and 20 modules (c). The solid blue line marked with circles represents the average maximum fitness value of all the runs of the generative encoding. The red solid line marked with triangles represents the average maximum fitness values of the direct encoding. The dotted lines represent the median of the two types of encodings. The thick error bars depict the 25-75 percentiles and the thin error bars depict the 0-100 percentiles.

Despite the fact that the rolling behavior is still predominant, a crawling behavior can be found in some individuals (figure 6b). When increasing the maximum number of modules to 20, some unspecified conglomerates of modules are found but most of the behaviors roll or crawl as in figure 6c.

In the generative encoding there was a recurrence of simple friction based phenotype (figure 7a) when simulating a maximum of 5 modules. This friction based phenotype seemed to exploit friction parameters of the simulator. Exactly half of the evolutionary runs that allowed for a maximum of 5 servo modules stagnated in a local optima with this kind of phenotype. Moreover, the fitness values of these individuals were quite low while other simple more effective morpholo-

gies, such as the phenotype shown in 7b, were possible to evolve. Considering the amount of modules of the resulting phenotypes, the amount of modules present in all robots was considerably smaller in the generative encoding compared to the direct encoding. The average amount of modules in the best evolved individuals of all runs was 3.5, 6.83 and 8.615 for the runs allowing 5, 10 and 20 modules max respectively. 5 out of 12 runs, when simulating a maximum of 20 modules, led to the evolution of morphologies composed of more than 10 modules (as seen in figure 7); 10 out of twelve led to the use of more than 5 modules. Seven out of 12 runs simulating a maximum of 10 modules led to the evolution of using more than 5 modules.

In the generative encoding not all genes of module states are represented in the evolved phenotypes. On the contrary, it seems that the evolutionary algorithm actively selects against the use of more modules. Out of all the evolutionary runs, the runs with a maximum of 5 modules only evolved phenotypes with on average 1.33 expressed servo module states. The runs simulating a maximum of 10 modules had an average of 2.66 expressed servo module states and the runs of simulating a maximum of 20 modules had on average 2.23 expressed servo module states. The phenotypes seen in figure 7d and 7e are examples of large phenotypes using only two types of servo module states.

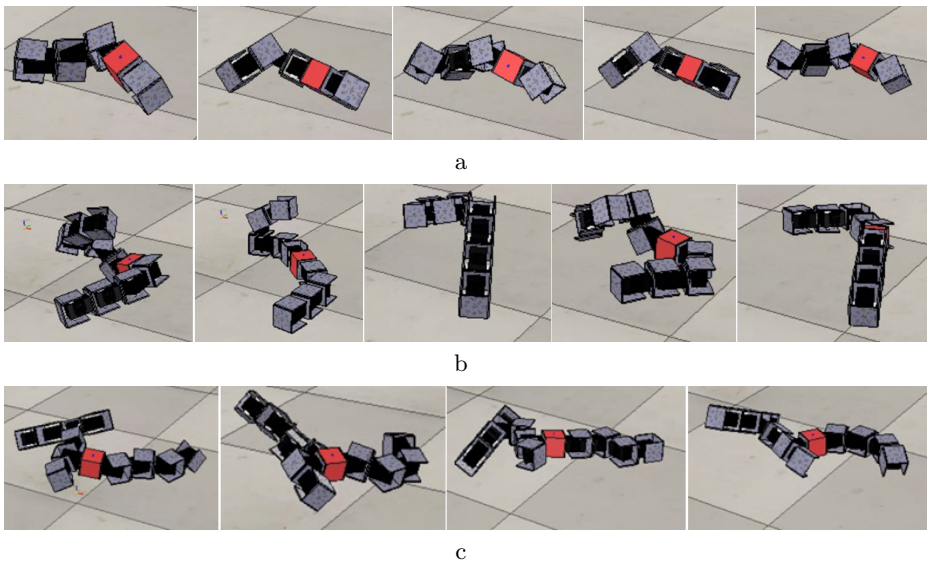


Fig. 6. Phenotypes obtained through the direct encoding. Resulting phenotypes simulated with a maximum of 5 modules (a), 10 modules (b) and 20 modules (c). For a more detailed visualization of the phenotypes, see [28]

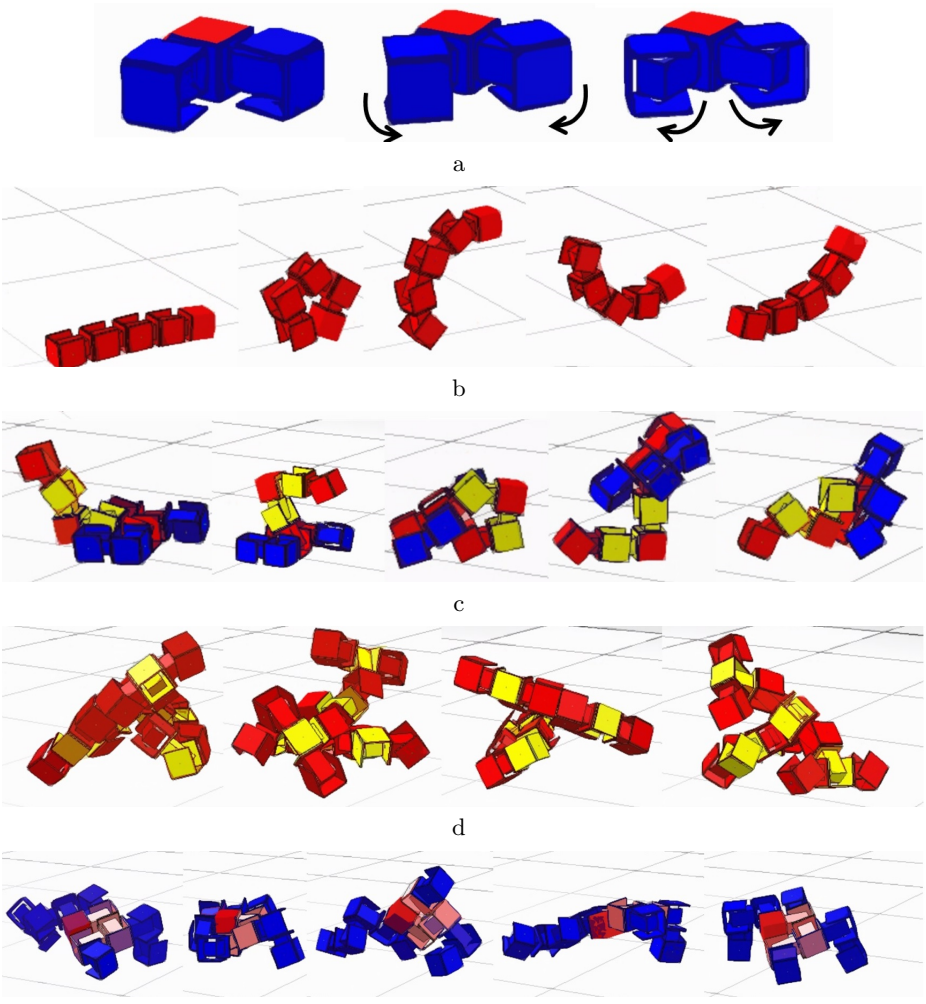


Fig. 7. Various phenotypes acquired through the generative encoding. Resulting phenotypes simulated with a maximum of five modules (a), ten modules (b) and 20 modules (c;d). For a more detailed visualization of the phenotypes, see [28]

4 Discussion

As can be derived from the graphs (figures 4 and 5) there is a difference in performance between the generative and direct encoding. The most striking difference in performance can be seen in the initial phase of the generative encoding where it outperforms the direct encoding. Over time, the direct encoding was able to catch up with the generative encoding and the performance differences were no longer statistically significant. The generative encoding still had an advantage in the long run when only a maximum of 5 modules could be simulated. This result was counter intuitive since we expected the generative encoding to perform better in the long run when more modules could be simulated.

A smaller portion of the genome of the generative encoding can lead to modular robots containing more modules than the direct encoding. The amount of servo module states used in the generative encoding was 2.23 on average in the final evaluations of the evolutionary runs of the generative encoding. Since 4 servo module states could be stored in the genome it is noteworthy to see that not all genetic information is expressed in the phenotype of the generative encoding. This result illustrates the usefulness of reusing the genome for creating modular robot morphologies. Being able to evolve robots with just a few genotypic parameters is furthermore an advantage and might lead to discovering abstract recursive mechanisms that are useful for the specified objective.

All evolutionary runs of the direct encodings led to phenotypes that utilized more modules compared to the generative encoding. This is due to a strong pressure in the direct encoding for adding new modules to the existing morphology. The mutations in the generative encoding can lead to destructive genotypes more quickly potentially posing a limiting factor to the amount of modules simulated for the individuals. Although the generative encoding outperformed the direct encoding in our comparison, the generative encoding was still prone to premature convergence. This premature convergence was not seen in the direct encoding due to other evolutionary parameters that were used in the encoding. An improved version of the evolutionary algorithm could implement methods to increase diversity and evolvability as done in speciation [29] – implemented in Neuroevolution of Augmenting Topologies (NEAT) [30] –, novelty search [31] or Age Layered Population Structure (ALPS) [32, 33]. Regarding the L-System, an alternative generative encoding, such as a Compositional Pattern Producing Network ([34]), can be a relevant alternative generative encoding for evolving modular robots (as applied in [16]).

Albeit out of the scope of this paper, the presented data is of limited use for robotic applications since we do not know how well the evolved behaviors transfer to reality. However, we expect that a hybrid approach of the two encodings would be a useful strategy to cope with the reality gap. The generative encoding can be used to evolve the global morphology and control of the robot while the direct encoding would tweak morphological and control parameters online or in a feedback loop with the simulator. This would be beneficial since the generative encoding cannot locally change parameters specific to individual modules. Nonetheless, it might also be better to evolve phenotypes using the

generative encoding and have an online learning system – such as a form of local decentralized learning [35] – adjust the control of the modules accordingly.

The presented semi-homogeneous modular robot system presents a promising step in the direction of evolving feasible modular robots. Increasing the heterogeneity in the system would give us additional insight in how we should model or modules in the future to produce even better robots. One could think of applying additional structural modules that have a variable stiffness. Since many organisms exploit various biomechanical attributes – be it elasticity, friction, strength – adding this type of module can enable evolution to come up with morphological solutions [36] and reduces the need for every part of the robot morphology to be actuated. Additionally, sensory modules can be implemented to extent the functionality of the system giving the robot inputs to its control system. The products of evolution of these potentially evolved hetegeneous modular robots can become experimental platforms that can be consulted before designing and building a non-modular equivalent.

5 Conclusion

Much work in evolutionary robotics is devoted to brain-body optimization strategies though few studies take into account the transferability of the evolved morphologies and control systems. We try to decrease this gap and enable researchers to have a fast way of evolving and evaluating robots in simulation and reality. Our robotic platform that simulates conglomerates of modules showed that a generative encoding, despite, having less optimization freedom, is more effective for evolving locomotion in simulated robots. The reuse of genes in the generative encoding seems to work well for the evolution of robot morphologies and control. This is a great advantage when constructing a robot out of many modules since many of them can be assigned with the same control parameters. We conceive that the generative encoding is able to evolve more abstract and simple robots and suspect that a hybrid system would be ideal for experimenting with the reality gap of the evolved robots. This hybrid system can initially use a generative encoding in simulation followed up by a direct encoding that locally optimizes parameters in a real robot.

Acknowledgement

This project was in part funded by Project 'flora robotica' which has received funding from the European Unions Horizon 2020 research and innovation program under the FET grant agreement, no. 640959. Computation/simulation for the work described in this paper was supported by the DeIC National HPC Centre, SDU. Special thanks to Rodrigo Moreno Garca (Universidad Nacional de Colombia) and Ceyue Liu (China University of Mining & Technology) that helped shape the design and implementation of the robotic Modules.

References

1. Lipson, H., Pollack, J.B.: Automatic design and manufacture of robotic lifeforms. Nature **406**(6799) (2000) 974–978
2. Hornby, G.S., Lipson, H., Pollack, J.B.: Generative representations for the automated design of modular physical robots. IEEE Transactions on Robotics and Automation **19**(4) (2003) 703–719
3. Eiben, A.E., Bredeche, N., Hoogendoorn, M., Stradner, J., Timmis, J., Tyrrell, a.M., Winfield, A.F.T.: The Triangle of Life: Evolving Robots in Real-time and Real-space. Advances in Artificial Life, ECAL 2013 (2013) 1056–1063
4. Stoy, K.: The deformatron robot: A Biologically inspired homogeneous modular robot. Proceedings - IEEE International Conference on Robotics and Automation (May) (2006) 2527–2531
5. Reece, J.B., Urry, L.A., Cain, M.L., Wasserman, S.A., Minorsky, P.V., Jackson, R.B.: Campbell Biology. (2010)
6. Floreano, D., Mattiussi, C.: Bio-Inspired Artificial Intelligence. (2008)
7. Marbach, D., a.J. Ijspeert, Ijspeert, a., a.J. Ijspeert: Online optimization of modular robot locomotion. IEEE International Conference Mechatronics and Automation, 2005 **1**(July) (2005) 248–253
8. Faíña, A., Bellas, F., López-Peña, F., Duro, R.J.: EDHMoR: Evolutionary designer of heterogeneous modular robots. Engineering Applications of Artificial Intelligence **26**(10) (2013) 2408–2423
9. Guettas, C., Cherif, F., Breton, T., Duthen, Y.: Cooperative co-evolution of configuration and control for modular robots. 2014 International Conference on Multimedia Computing and Systems (ICMCS) (October 2015) (2014) 26–31
10. Sims, K.: Evolving virtual creatures. In: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques. Number July (1994) 15–22
11. Sims, K.: Evolving 3D Morphology and Behavior by Competition. Artificial Life **1**(4) (1994) 353–372
12. Hornby, G., Pollack, J.: The advantages of generative grammatical encodings for physical design. Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546) **1** (2001) 600–607
13. Auerbach, J.E., Bongard, J.C.: Evolving Complete Robots with CPPN-NEAT: The Utility of Recurrent Connections. Gecco-2011: Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (2011) 1475–1482
14. Cheney, N., MacCurdy, R., Clune, J., Lipson, H.: Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Generative Encoding. Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation - GECCO '13 (2013) 167
15. Bonardi, S., Vespignani, M., Moeckel, R., Kieboom, J.v.d., Pouya, S., Sproewitz, A., Ijspeert, A.J.: Automatic generation of reduced CPG control networks for locomotion of arbitrary modular robot structures. In: Proceedings of Robotics: Science and Systems. (2014)
16. Auerbach, J.E., Heitz, G., Kornatowski, P.M., Floreano, D.: Rapid Evolution of Robot Gaits. In: GECCO15. (2015) 743–744
17. Lindenmayer, A.: Mathematical models for cellular interactions in development. I. Filaments with one-sided inputs. Journal of theoretical biology **18**(3) (1968) 280–299
18. Kamimura, A., Kurokawa, H., Yoshida, E., Murata, S., Tomita, K., Kokaji, S.: Automatic locomotion design and experiments for a modular robotic system. IEEE/ASME Transactions on Mechatronics **10**(3) (2005) 314–325

19. Sproewitz, A., Moeckel, R., Maye, J., Ijspeert, A.J.: Learning to Move in Modular Robots using Central Pattern Generators and Online Optimization. The International Journal of Robotics Research **27**(3-4) (2008) 423–443
20. Still, S., Hepp, K., Douglas, R.J.: Neuromorphic walking gait control. IEEE Transactions on Neural Networks **17**(2) (2006) 496–508
21. Ijspeert, A.J.: Central pattern generators for locomotion control in animals and robots: A review. Neural Networks **21**(4) (2008) 642–653
22. Rohmer, E., Singh, S.P.N., Freese, M.: V-REP: A versatile and scalable robot simulation framework. IEEE International Conference on Intelligent Robots and Systems (2013) 1321–1326
23. Faiña, A., Orjales, F., Bellas, F., Duro, R.: First Steps towards a Heterogeneous Modular Robotic Architecture for for Intelligent Industrial Operation, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011) (2011)
24. Cheney, N., Bongard, J., Sunspiral, V., Lipson, H.: On the Difficulty of Co-Optimizing Morphology and Control in Evolved Virtual Creatures. In: Proceedings of the Artificial Life Conference 2016 (ALIFE XV). (2016) 226–234
25. Lindenmayer, A., Jürgensen, H.: Grammars of Development: Discrete-State Models for Growth, Differentiation, and Gene Expression in Modular Organisms. In Rozenberg, G., Salomaa, A., eds.: Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology. Springer Berlin Heidelberg, Berlin (1992) 3–21
26. Veenstra, F., Faina, A., Stoy, K., Risi, S.: Generating Artificial Plant Morphologies for Function and Aesthetics through Evolving L-Systems. In: Proceedings of the Artificial Life Conference 2016, MIT Press (2016) 692–699
27. Prusinkiewicz, P.A.L., Lindenmayer, A.: The algorithmic beauty of plants. Plant Science **122**(1) (1997) 109–110
28. Veenstra, F., Faina, A., Risi, S., Stoy, K.: Video: Evolving Modular Robots Using Direct and Generative Encodings. In: https://www.youtube.com/watch?v=HCDftic1AdA. (2017)
29. Cook, O.F.: Factors of Species-Formation. Science **23**(587) (1906) 506–507
30. Stanley, K.O., Miikkulainen, R.: Efficient Evolution of Neural Network Topologies. Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on Evolutionary Computation (2002) 1757–1762
31. Lehman, J., Stanley, K.O.: Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. Artificial Life XI (2008) 329–336
32. Hornby, G.S.: ALPS: The Age-Layered Population Structure for Reducing the Problem of Premature Convergence. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation. (2006) 815–822
33. Hornby, G.S.: The age-layered population structure(ALPS) evolutionary algorithm. Proceedings of the 9th annual conference on Genetic and evolutionary computation (2009)
34. Stanley, K.O.: Compositional pattern producing networks: A novel abstraction of development. Genetic Programming and Evolvable Machines **8**(2) (2007) 131–162
35. Christensen, D.J., Schultz, U.P., Stoy, K.: A distributed and morphology-independent strategy for adaptive locomotion in self-reconfigurable modular robots. Robotics and Autonomous Systems **61**(9) (2013) 1021–1035
36. Pfeifer, R., Iida, F.: Morphological computation: Connecting body, brain and environment. Japanese Scientific Monthly **58**(2) (2005) 48–54