Comparison of Different Genotype Encodings for Simulated 3D Agents

Maciej Komosinski   Adam Rotaru-Varga

Institute of Computing Science
Poznan University of Technology
Piotrowo 2, 60-965 Poznan, Poland
maciej.komosinski@cs.put.poznan.pl

Abstract

This paper analyzes the effect of different genetic encodings used for evolving 3D agents with physical morphologies. The complex phenotypes used in such systems often require non-trivial encodings. Different encodings used in Framsticks – a system for evolving 3D agents – are presented. These include a low-level direct mapping and two higher-level encodings: a recurrent and a developmental one. Quantitative results are presented from three simple optimization tasks (active height, passive height, and locomotion speed). The low-level encoding produced solutions of lower fitness than the two higher-level encodings under similar conditions. Results from recurrent and developmental encodings had similar fitness values but displayed qualitative differences. Desirable advantages and some drawbacks of more complex encodings are established.

Keywords

virtual creature evolution, morphological evolution, 3D agents, genotype encoding, developmental encoding

1 Introduction

A survey of the field [21] indicates that there are a number of recent studies of the evolution of simulated creatures equipped with realistic physical behavior [7, 15, 18]. Most of these works can be traced back to the influential work of Karl Sims [20]. When comparing such systems with other evolutionary systems, we can note that the use of a physical simulation layer implements a complex genotype–fitness relationship. Physical interactions between body parts, the coupling between control and physical body, and interactions during body development can all add a level of indirection between the genotype and fitness. The complexity of the genotype–fitness relationship offers a potential for rich evolutionary dynamics.

The most important element of the genotype-to-fitness relationship is the genotype-to-phenotype mapping, or genotype encoding. There is no obvious simple way to encode a complex phenotype – which consists of a variable-size, structured body and a matching control system – into a simpler genotype. Moreover, an evolutionary algorithm can perform poorly when using a certain genotype encoding, and better when using others, for reasons not yet immediately obvious. The employed genotype encoding can have a significant effect on the performance of the evolution. This fact

The final version of this paper appeared in *Artificial Life Journal* 7(4):395–418, 2001. http://dx.doi.org/10.1162/106454601317297022 has been recently recognized by researchers who directed efforts into developing more sophisticated encodings [2, 3, 10]. The aforementioned evolutionary systems lack a common base for experimentation[1], they use different physical engines, evolutionary algorithms, and various approaches for genotype encodings. Such differences render a comparison aimed solely at the effect of genetic encodings difficult.

In the present work we use a single system, Framsticks, as the context of our analysis of various genotype encodings. Framsticks is a realistic, three-dimensional simulation of agents and their interactions [12, 15, 16]. We present the three encodings currently implemented in the system, which include a simple low-level encoding and two higher-level ones: a direct recurrent and an indirect developmental. The low-level encoding is the simplest, and is considered to be a special case, while the other two are more complex, having been designed to be more evolvable. We compare the performance of the encodings in there optimization tasks (passive and active height, velocity), in experiments which differ only in the encoding used.

The organization of the article is as follows: Sect. 2. contains a general discussion on the effect of different encodings. Sect. 3. and Sect. 4. provide an overview of the Framsticks system and its three different encodings, followed by experimental results in Sect. 5. Conclusions and direction for further work are included in Sect. 6.

2 The Role of the Genetic Encoding

Most evolutionary simulation systems distinguish between the concepts of *genotype* and *phenotype*, and employ a *mapping* between the two (this mapping is the trivial identity mapping in simple GA cases). As an evolutionary system allows for a more complex phenotype space, a more complex genotype-to-phenotype mapping (or *encoding*) is called for to allow genotypes to concisely describe complex phenotypes. Given a phenotype space, an encoding does not automatically follow: it is possible to construct differrent genotype spaces which map into the phenotype space, and even for one particular genotype space, it is possible to devise numerous mappings from it to the phenotype space. Does the selection of a particular encoding have a significant effect on the outcome of the evolutionary search? While it is hard to devise an 'ideal' encoding, it is certain that some encodings perform better than others. The issue of genetic encodings has been first addressed in the context of GAs, with the binary vs. Gray coding being one specific example [17]. These studies established that the genetic encoding can have a noticeable effect on the evolutionary search.

The task of evolution of physical agents is difficult for several reasons. Single points in the phenotype space are complete creatures, with a structured morphology and control (each creature has a 'body' and 'brain'). There is a variable amount of information required to describe such an organism. The dimensionality of the search space is not fixed, and the space does not lend itself to a straightforward neighborhood definition. Some features of the phenotypes change continuously (e.g., length of a body part), while others change discretely (e.g., number of body parts). Furthermore, the evaluation of a single phenotype involves a lengthy physical simulation, which can amplify small changes in the phenotype and lead to large changes in fitness; the imposed fitness landscape is multi-peaked; and evaluations contain non-deterministic components. In order to deal with such a large and complex search space, adequate genetic encodings are called for. The large number of CPU cycles required for fitness evaluations poses an additional technical difficulty.

Let us consider the implications of a chosen genetic encoding for a phenotype space (which we consider as given, determined by the chosen simulation rules). Typically an encoding maps only to a *subset* of the phenotype space. Valid phenotypes exist which cannot be expressed by the encoding. An example in the Framsticks system is morphologies containing cycles: although

[1]Recently some researchers expressed ideas about using a common physics platform [21].

they are valid phenotypes, the default encoding (***recur***)[2] cannot express them. The existence of phenotypes which are impossible to encode means that entire portions of the search space are sealed off from the search. This, nonetheless, might benefit the search, if the pruned space is smoother or denser in high-fitness points than the entire space.

More importantly, the encoding defines the *topology* of the phenotype space. Genetic operators (which are encoding-specific) determine which genotypes – and the corresponding phenotypes – are neighbors. The topology is encoding-specific: suppose phenotypes F_1 and F_2 are encoded by neighboring genotypes G_1 and G_2 under encoding S, and encoded by Γ_1 and Γ_2 under encoding Σ. Even if G_1 and G_2 are neighboring (there is a mutation in S which turns G_1 into G_2), Γ_1 and Γ_2 can be distant in the genotype space in Σ. For example, the ***recur*** genotypes '`XXXX`' and '`XLXXX`' are separated by one point mutation (insert '`L`'), but the corresponding ***simul*** genotypes are separated by three mutations (one for each of the last three sticks). An encoding – together with its associated genetic operators – determines which phenotypes are neighboring, and also influences which phenotypes have a higher probability of being visited.

The phenotype space topology imposed by the encoding determines which parts of the space are easily accessible by the search. Although the evolutionary search operates on phenotypes, the genotype encoding indirectly influences the outcome of the search. The bias imposed by an encoding becomes evident in the case of a random search: two random searches with different genotype encoding yield different results, despite the identical phenotype space. A phenotype space has an inherent topology: phenotypes of similar fitness are 'close' to each other. The genotype encoding imposes another topology, by defining which phenotypes are close genetically (measured as the number of mutations separating the corresponding genotypes). Under a good encoding, these two topologies are more highly correlated [9]. Unfortunately, this correlation is impossible to directly measure, because the vast complexity of the phenotype space and its fitness landscape. Our judgments of encodings are thus subjective, based on various theoretical and experimental observations.

Different encodings have different characteristics, and some are better suited for some types of evolutionary search. It is hard to objectively establish these qualities of the encodings, and it is doubtful that a single best one exists. This perspective was our motivation to expand the Framsticks system to support several encodings at the same time.

3 The Framsticks World

The following sections contain an overview of the Framsticks system and its capabilities. The Framsticks system simulates a three-dimensional world populated by agents. Agents are composed of an articulated morphology ('body') and attached control system ('brain'), which are described in turn. For a more detailed presentation, consult [1, 12, 15, 16].

3.1 Body

The bodies of the agents are composed of a set of interconnected simple elements called *sticks*; a stick consists of two material endpoints connected through a flexible rod. Sticks have various physical and biological properties (mass, stamina, assimilation, etc.). Articulations exist between sticks where they share an endpoint; the articulations are unrestricted in all three degrees of freedom (bending in two planes plus twisting).

A wide range of physical interactions between sticks are simulated: static and dynamic friction, damping, action and reaction forces, gravity, buoyancy (uplift pressure under water), and energy losses from deformations; some of these forces are shown in Fig. 1. The finite element method is

[2]The different encodings are presented in Sect. 4.

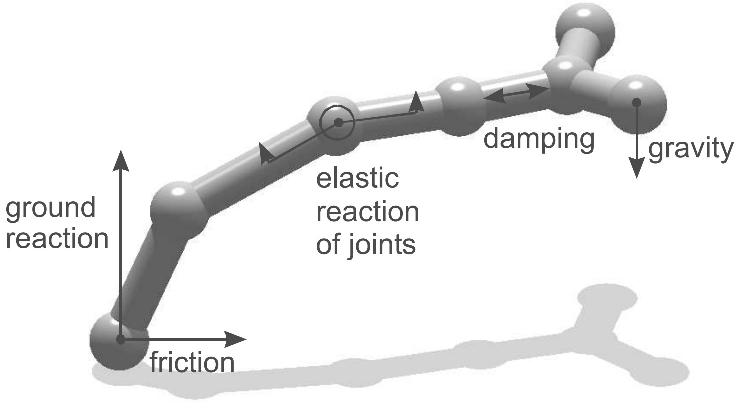


Figure 1: Various kinds of forces considered in the physical simulator of the Framsticks system.

used for step-by-step simulation. Collisions are simulated between stick of different agents, but not between the sticks of a single agent (for efficiency reasons).

3.2 Brain

A Framsticks agent is also equipped with a control system, which is implemented by a network of artificial *neurons*. Some neurons are specialized into *sensors* and *effectors*, for interfacing with the mechanical body.

Generic neurons are simple processing units, similar to the ones used in standard artificial neural networks. Every neuron has a variable number of weighted connections from other neurons, and several parameters which influence its function. A neuron output value is updated periodically. The updating rule is based on the standard sigmoid function:

$$o = \frac{2}{1 + e^{-i \cdot \beta}} - 1 \tag{1}$$

In this equation o is the activation value (output), i is the sum of weighted inputs, and β is the steepness parameter of the transfer function.

There are some extensions employed in the Framsticks neuron model, which allow for a wider variety of neurons. Neurons possess an internal state s, which is updated with a certain inertia: s follows i with a speed of change v, which depends on the difference between i and s:

$$o_t = \frac{2}{1 + e^{-s_t \cdot \beta}} - 1 \tag{2}$$

$$s_t = s_{t-1} + v_t \tag{3}$$

$$v_t = v_{t-1} \cdot \lambda + \mu \cdot (i_t - s_{t-1}) \tag{4}$$

The additional tunable parameters are λ and μ. For $\lambda = 0$ and $\mu = 1$ we get (1). The parameters β, λ and μ can all be under genetic control.

The rules for computing neuron activation values are deterministic, however, initial activations are set stochastically to small values. The reason for employing randomness is to discredit neural

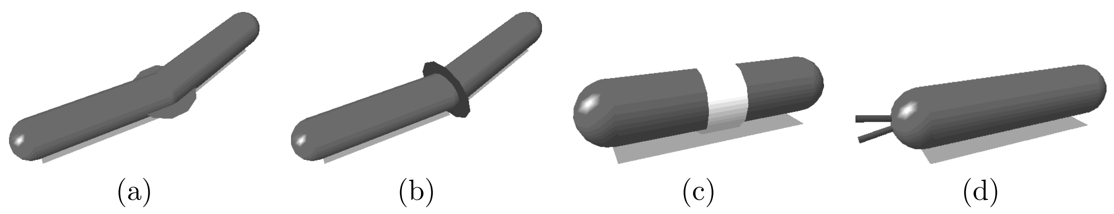


Figure 2: Visualisation of effectors: (a) bending and (b) rotating, and receptors: (c) equilibrium and (d) touch. Effectors are drawn on the first of the two endpoints they influence.

networks that rely extensively on specific initial patterns, and encourage those which rely more on the information present in the environment. The latter ones tend to produce more robust solutions.

The special neurons include effectors (muscles) and various sensors. Effectors are *muscles* that can exert modulated forces at articulations they are attached to. Muscles exists in two varieties: *bending* and *rotating*.

Sensors, also attached to sticks, include orientation and touch sensors.[3] An *orientation sensor* (denoted '`G`', also called a gyroscope) measures the orientation of the stick relative to the horizontal plane. A *touch* sensor (denoted '`T`') reacts to a contact force at the end of the stick, and can detect contact with the ground (or lack of it). Both orientation and touch sensors are useful for building controls for locomotion and other behaviors. See Fig. 2. for an illustration of the Framsticks effectors and sensors. Additionally, see the Appendix for more detail on the capabilities of the system.

4 Genetic Encodings in Framsticks

4.1 Support for multiple encodings

There are multiple encodings supported by the Framsticks system, each with its own representation and operators. The system manipulates and transforms genotype strings in various representations, and ultimately decodes them into the internal representation used by the simulator.

Any creature can be completely described using a low-level representation, by listing all of its components and attributes. This representation can be treated as a special genotype encoding – special because it is a direct one-to-one mapping – which we call ***simul***. Other higher-level encodings convert their representation into the corresponding ***simul*** version (possibly through another intermediary representation), as illustrated in Fig. 3. The reverse mapping of higher-level encodings is difficult to compute, which is also true for biological phenotype encodings. As a consequence, in the general case it is not possible to convert a lower-level representation into a higher-level one (or a higher-level one into another higher-level one). Nonetheless, an approximate transformation is possible from ***devel*** genotypes to ***recur*** genotypes, but not the other way around.

Each encoding has its associated genetic operators (mutation, crossover, and optional repair), and a decoding procedure which translates a genotype into a ***simul*** (or another 'lower') representation. A new encoding can be added relatively easily, by implementing these components, without the need to work with internal representations. The Framsticks system is accompanied by the Software Development Kit (SDK) to simplify this process [16].

In this article we describe three encodings: the direct low-level, direct recurrent, and indirect developmental. There are other encodings which are currently under development (similarity-based, metabolism-based, etc.). The *direct low-level encoding* (denoted ***simul***, from 'simulator', elsewhere

[3]There is also a smell sensor, not used in the present work.

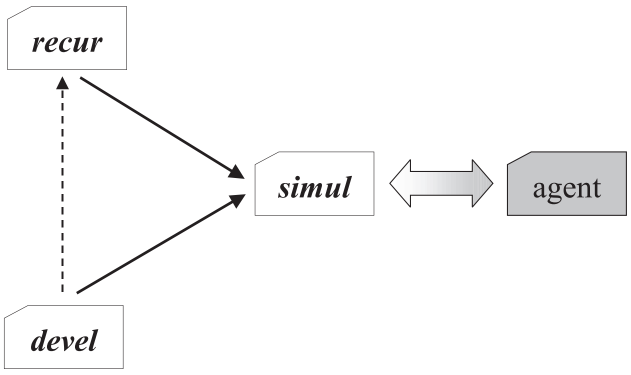


Figure 3: The architecture for multiple encodings. Solid arrows indicate decoding of one representation into another. Dashed arrow indicates an approximate transformation (with potential loss of information).

also referred to as 'f0') is a universal encoding which directly describes any valid phenotype. The *direct recurrent encoding* (***recur*** from 'recurrent', 'f1'), the original encoding in Framsticks, is significantly more compact than ***simul***, but still preserves a one-to-one mapping between genotypic and phenotypic parts. It uses higher-level rules to achieve compactness and transparency. The *indirect developmental encoding* (***devel*** from 'developmental', 'f4') is similar to ***recur***, but describes the process of creation of a creature, rather than its final form. Consequently, it supports some features that are results of interactions during the developmental process: most importantly, modularity.

4.2 Direct low-level

The direct low-level, or ***simul***, encoding describes agents exactly as they are represented in the simulator. This encoding is more of a direct representation than a proper encoding, but it is possible to use it as such. It does not use any higher-level features to make the genotype more compact or flexible, and because of this, it is expected that this encoding is not very well suited for evolution. Its useful characteristics are that it has a minimal decoding cost and that it is universal: every possible agent can be described using this encoding. These properties make it possible to use the ***simul*** encoding as an intermediary representation during the translation from other higher-level encodings.

A ***simul*** genotype consists of a list of descriptions of all the objects the agent is composed of: *parts*, *joints*, *neurons*, and *neuron items* (connections, sensors, effectors). Every description specifies all the attributes of the object explicitly (except those which are equal to their default value). A sample genotype is provided in Fig. 4. All objects are implicitly numbered by their position in the list, and these absolute order numbers are used for later reference (e.g., each neuron has a reference to the part it is attached to). The absolute reference numbers are global properties, and are affected by a change in the number or order of the objects. Generally the ***simul*** encoding does not impose any restriction on the phenotypes[4], and it even allows morphologies with cycles (such as the one illustrated in Fig. 5.a). This is not true of the other two encodings. Further details of this format can be found in [13].

In order to use ***simul*** as a true encoding, both genetic operators are implemented. Point

[4]There is one small restriction present, however: the orientation of parts is always zero. The orientation of a part influences both how receptors ('`T`') and muscles work. Touch sensors always 'look' in one direction, and muscles work along the same axis. This is not a serious restriction, as muscles can still produce various movements. In other representations, the orientation of a part is determined by a 'growing axis'.

```
p:D=0
p:1, m=2, vol=2, D=0
p:2, m=3, vol=3, D=3
p:2.50017, -0.000170005, -0.865927, D=5
p:2.50017, 0.000170005, 0.865927, 3, vol=3, D=7
p:3.50017, 0.000170005, 0.865927, D=9
p:2.00051, 0.000340067, 1.73215, D=11
j:0, 1, dx=1, D=0
j:1, 2, 1.5706, dx=1, D=3
j:2, 3, rz=-1.047, 1, D=5
j:2, 4, rz=1.047, 1, D=7
j:4, 5, rz=-1.047, 1, D=9
j:4, 6, rz=1.047, 1, D=11
```

Figure 4: A sample ***simul*** genotype. This genotype corresponds to the phenotype shown in Fig. 7.b – ***recur*** genotype '`XRRX(X,X(X,X))`'. Lines starting with a 'p:' represent material endpoints, while lines starting with a 'j:' represent rods joining two endpoints. The first two numbers after 'j:' are the references for the two endpoints.

mutation of a genotype is straightforward: it either changes one attribute of one object, or (less frequently) removes an existing object or adds a new one. In either case, mutation affects exactly one element of the agent. The ***simul*** encoding does not offer a straightforward method for crossover. Thus we based crossover on phenotypic geometry: both morphologies are cut in two parts using a plane randomly positioned in space, and the two halves from each of the agents are grafted together. Neurons follow the part they are attached to, and broken links are reconnected to recover lost functionality. Such an example is shown in Fig. 5. In a special case, when two identical parents are crossed over, the resulting offspring is identical to the parents.

4.3 Direct recurrent

The direct recurrent, or ***recur***, encoding was the first one employed in the Framsticks system. This higher-level encoding was designed so that genotypes are compact and robust in face of genetic operators. Being easily understood and manipulated by humans was also a consideration.

The details of the ***recur*** encoding had been covered in [12, 14, 15], so only a brief overview is given here. In a ***recur*** genotype, the component sticks of a morphology are described using a string as follows: each stick is represented by a letter '`X`', and two consecutive '`X`'s represent two sticks joined together. If there is a stick joined to several other sticks, they are represented using the structure '`X(X ...   , X ...   , X ...   )`'. This is sufficient to represent any cycle-free stick topology, see Fig. 6. for some examples.

Various attributes of sticks are specified using modifier letters. Modifiers change the value of a certain property in a relative manner, starting with an implicit default value. Lowercase letters denote modifiers which decrease, and capital letters denote modifiers which increase the value of an attribute by a fixed factor. Repeated letters achieve a compound effect, for example, '`lX`' represents a short stick, while '`LLLX`' a very long one. Furthermore, modifiers are 'fuzzy' and do not only affect the first following stick, but later ones as well, with a decreasing weight. For example, the lengths of the sticks in the genotype '`XLLXXX`' are 1.00, 2.00, 1.51, and 1.255 respectively. There are modifiers for basic attributes (present also in the ***simul*** encoding), and some ***recur***-specific ones,

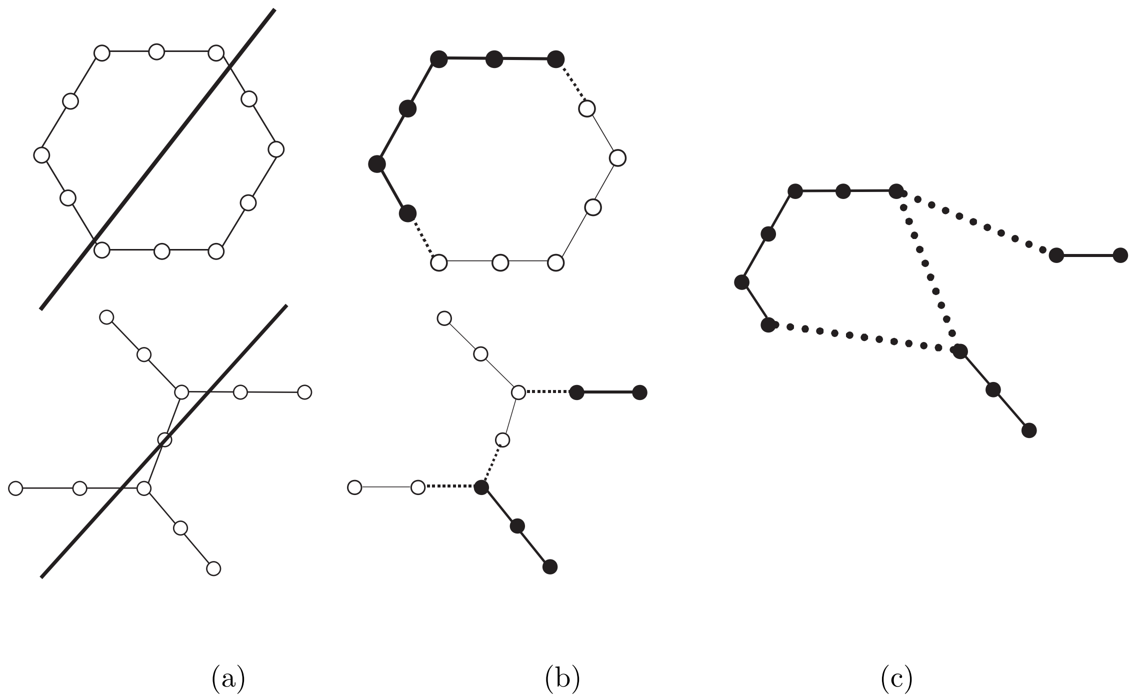


Figure 5: A ***simul*** crossover example. (a) The two parent structures; shown with their cutting planes. (b) The separated parents; dashed lines are joints which are broken, and the two halves shaded in black are used in the child. (c) The child structure; dotted lines are newly created joints.

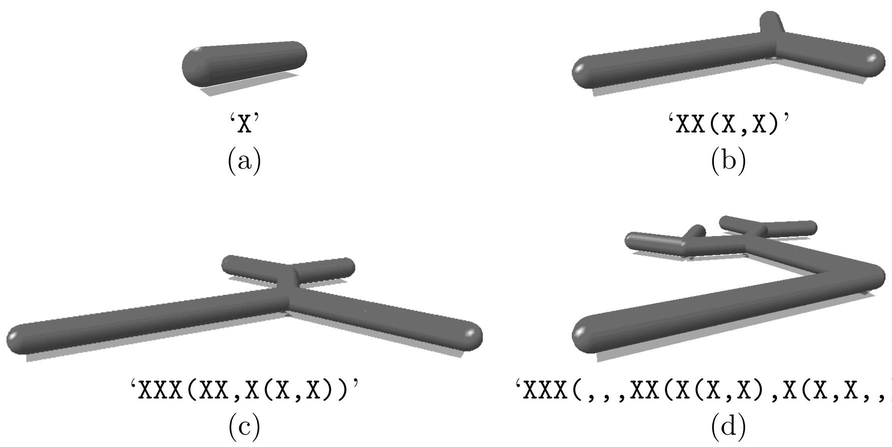


Figure 6: Examples of genotypes describing bodies of increasing complexity. (a) Single stick, (b) two joined sticks branching into single sticks, (c) recurrent branching, (d) recurrent branching with some branches missing.

Symbol	Meaning
R	Rotation of branching plane by 45°
Q	Skew of branching plane
C	Curvedness
L	Length
F	Friction
M	Muscle strength

Table 1: Modifier symbols in ***recur***. '`RRR`' rotates the current branching plane by $3 \cdot 45° = 135°$; '`rr`' by $-90°$, etc.

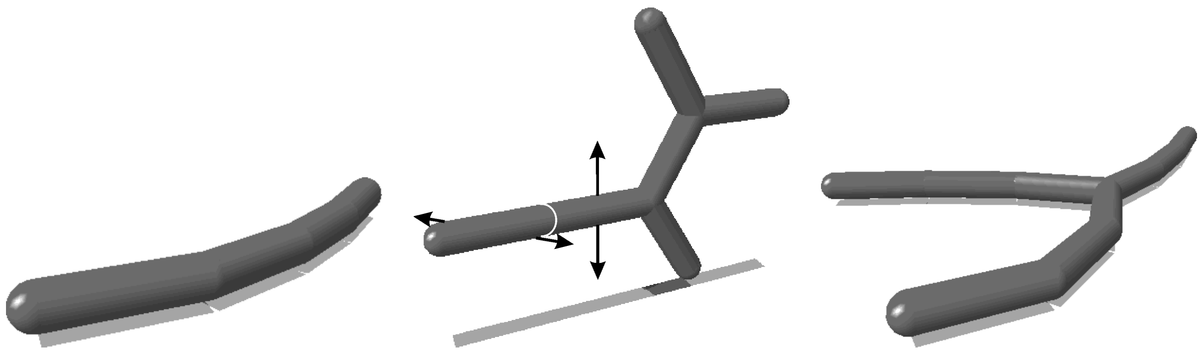


Figure 7: Examples of ***recur*** modifiers. (a) Shortening and curvedness, (b) rotation of branching plane by 90°, (c) tendency for curving.

such as the curving angle between two sticks. Modifiers and their meanings are listed in Table 1. Related examples of genotype-phenotype pairs are presented in Fig. 7.

Neurons are represented by the symbols '`[ ... ]`', inserted after the stick they are attached to. There are various parameters specified for a neuron: type ('`|`' for bending muscle, '`@`' for rotating muscle, etc.), a comma-separated list of connections in the format '`<input>:<weight>`', where input is either an integer, in which case it denotes another neuron using relative numbering (-1 means the previous, +2 the second next one, etc.), or a sensor ('`T`', '`G`', etc.). Fig. 8. presents three examples with explanations: '`X[0:5]`' is a stick with a neuron which has a single recurrent link from itself, '`X[@1:3]X[G:-2]`' is two sticks, both with one neuron, the second connected to a gyroscope sensor, and the first connected to the second, and driving a bending muscle.

The ***recur*** encoding has specialized genetic operators. Mutation either adds a new modifier, a new stick or neuron, or deletes an existing modifier, stick, or neuron, or changes the parameters of an existing neuron. Crossover operates on the genotype strings in a straightforward way: it swaps randomly isolated substrings among two strings. However, cut points are restricted to logical positions (i.e., the multiple characters describing a single neuron are never separated).

The ***recur*** encoding has several properties resulting from its design features. Important properties are:

– Linkage between body parts is done implicitly, rather than using explicit references to parts

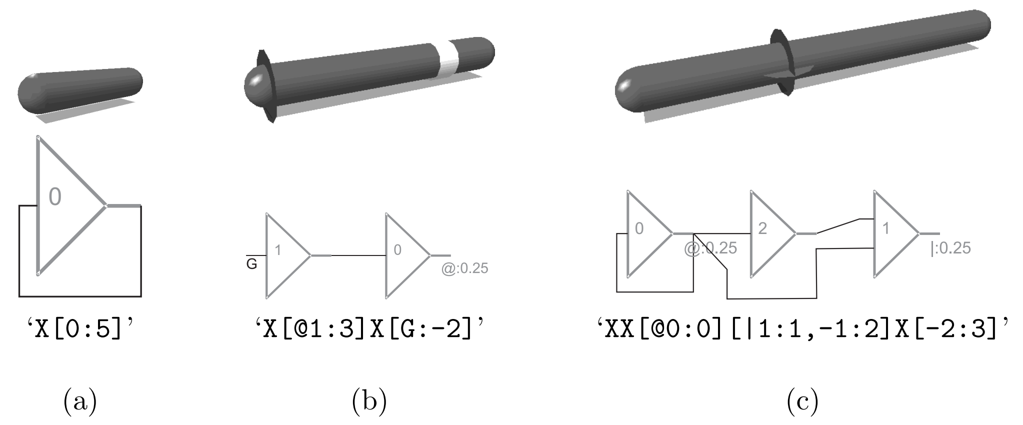


Figure 8: Examples of describing linkage between body and control. (a) Single stick and neuron, not connected functionally, (b) two sticks: one with a rotating muscle, the other with a G receptor, (c) example of a more complex NN and two kinds of neurons for the same joint. The body shape, the neural network, and the genotype is shown in each case. Note that the neural networks are shown with inputs arranged to the left and outputs to the right, which results in a different ordering of the neurons than in the genotypes.

(eg., ‘`XX`’ implicitly creates a link between the two sticks).

– Linkage between neurons is done using relative rather than absolute numbering. Relative numbering is susceptible to disruptions only if the added (removed) neuron is between the two ends of the link, but resistant otherwise. The latter cases are more frequent, and links specified by absolute numbering would be broken in such cases.

– Attribute changes propagate along the body structure.

– Implicit default values and automatic constraints are used extensively (e.g., if there is a link from a sensor, there must be a sensor).

– Control elements (sensors, effectors) are described near the elements controlled.

Now we can list the characteristics of the ***recur*** encoding:

1. Non disruptive. Small changes to the genotype generally cause small changes to the phenotype. The changes can propagate to multiple sites, but in a continuous and structured manner.
2. Crossover-friendly. Substrings retain at least part of their meaning when isolated and inserted into another context. Disrupted references are fixed by automatic constraints and implicit rules.
3. Human-friendly. The relationship between a genotype and its phenotype is relatively easy to understand by a human user, so genotypes can be be analyzed and modified by hand.
4. Complete. The genotype specifies every aspect of the phenotype completely (albeit not all directly).
5. Minimal redundancy. There are no genotype parts with no influence over the genotype.

In summary, the direct recurrent encoding uses higher-level constraints, and is potentially more effective in an evolutionary search than the direct low-level encoding. However, this conjecture needs to be tested empirically.

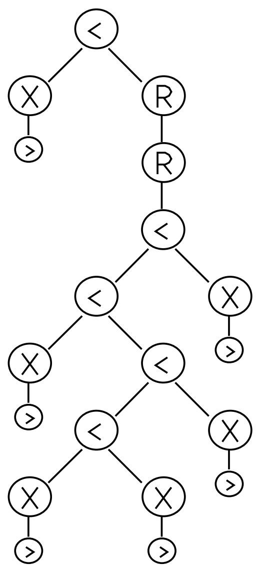


Figure 9: A sample ***devel*** genotype represented as a tree. The genotype is '`<X>RR<<X><<X>X>X>X`', equivalent to the ***recur*** genotype presented in Fig. 7.b.

4.4 Indirect developmental

The indirect developmental encoding describes an agent by specifying its developmental process rather than its final form. A preliminary description of this encoding has been included in [14]. The encoding models a set of interacting phenotypic parts which execute actions specified by the genotype. This encoding is inspired by the cellular developmental encoding of Gruau [7], but is extended to include the body as well. Developmental encodings have been applied to evolution of neural networks [11, 23, 8, 4, 19], and more recently to both neural networks and morphologies [5, 6, 2, 3, 10], and have been found to be superior to direct encodings, by producing more structured and modular phenotypes. Extra complexity and unnecessarily large phenotypes are reported as disadvantages.

Using the ***devel*** encoding, creatures are built by passing through a developmental phase. A developing creature consists of a set of interconnected *cells*, which can be *undifferentiated* or *differentiated* (sticks or neurons). Cells execute genetic codes which alter their properties, or create new cells through division. After a division the newly created cells execute different codes (they *differentiate*). At this point the genetic codes fork, which is why the entire genotype is organized as a tree. Development of a creature starts out as a single undifferentiated cell. As new cells are created, they follow their instruction in parallel. Undifferentiated cells can mature into sticks ('`X`') or neurons ('`N`'). Development halts when all cells mature [7, 19].

Devel is similar to ***recur*** insofar as it uses similar genetic symbols ('`X`', '`[ ... ]`', etc.). In ***recur*** the genotype is traversed, and the codes are interpreted by an external builder process, which creates the parts of the creature. In ***devel*** the codes are interpreted by the developing parts themselves. ***Recur*** codes consist of ones that generate new parts, and modifiers that change properties of parts that are to be created. ***Devel*** also has analogous modifiers, but these affect existing parts.

A ***devel*** genetic code tree is represented as a string, generated by traversing the tree pre-order (for each node, first the node is described, then its subtrees). A division '`<`' is followed by the codes executed by the parent cell, until the corresponding stop code '`>`', then by the codes executed by the daughter cell. If either of the codes is itself a tree, the same rule is applied recursively. This

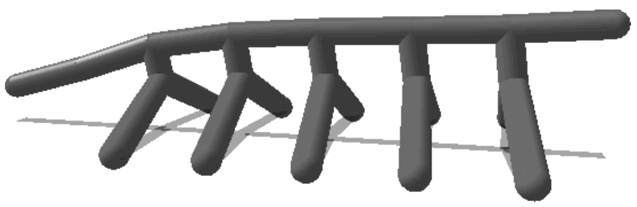


Figure 10: Example of the repetition operator in genotype '`rr<X>#5<,<X>RR<<llX>LX>LX>>X`'.

way the '`<`' and '`>`' codes also act like nested parentheses. An example is presented in Fig. 9: the genotype '`<X>RR<<X><<X>X>X>X`' is shown as parsed into a tree structure. The resulting genotype (Fig. 7.b) consists of six sticks, easily seen from the fact that the tree contains six terminal nodes (and five branching nodes).

The ***devel*** encoding supports repetition of the same code portion more than once. This is implemented by the repetition code '`#`', which has two subtrees (like '`<`'): the first is the code which is repeated, and the second is the rest, executed after the repetitions. The '`#`' code also specifies the number of repetitions. The repeated subtree can contain an arbitrary number of codes, including divisions. In this case only daughter cells will continue the repetition, not both of them.

The repeated genotype portion can consist of a single node or a subtree of arbitrary size. If the repeated subtree includes only a modifier, the effect of the modifier will be enhanced, but no new parts will be created. If it includes a division ('`<`'), multiple copies of the element will be created. The simplest case is the repetition of a single stick or neuron. The repeated portion can contain multiple division codes, and even nested repetition codes. For example, the genotype presented in Fig. 10. features a repeated body segment consisting of three sticks.

Phenotype modularity produced through developmental repetitions can be contrasted with modularity produced through phylogenic gene duplication. During evolution a genotype portion encoding body substructures can be duplicated (especially by crossover). However, even if the resulting duplicated phenotypic structures are identical, they are no longer encoded by the same genetic codes, and thus will evolve independently. In the case of ***devel*** encoding, repeated identical body parts are encoded by shared genetic codes. This opens up the possibility of mutations which affect all copies simultaneously [19]. Modularity is argued to be a useful property when evolving complex entities [7, 2]. However, the modularity in ***devel*** encoding is somewhat limited in that modules are identical, and cannot diverge or differentiate.

The ***devel*** encoding is accompanied by specialized genetic operators: mutation affects a single node in the genotype tree (it is similar to the ***recur*** mutation). Crossover isolates and swaps subtrees from the genotype trees, which is the standard method used in genetic programming.

The ***devel*** encoding is similar to the ***recur*** encoding in that it uses analogous genetic codes, and every ***devel*** genotype can be translated into an approximate ***recur*** genotype. Differences include support for modularity and the mechanism for propagating attribute values from one element to another. In ***devel***, when a cell divides, the new cell inherits the attributes of the old cell. The effect is similar to how modifier effects are propagated in ***recur***. But the propagation-by-division is more flexible: for example, dividing neurons can duplicate their existing connections (see Fig. 11. for an example). This provides one way to compress information needed to describe a neural network.

In conclusion, the ***devel*** encoding is similar to the ***recur*** encoding, but uses some features of a developmental encoding. It supports segmentation and modular body layouts, features which are not supported by ***recur***. However, ***devel*** is not a full-fledged developmental encoding, because its support for modularity is limited: repeated modules are always identical. (This is also true of the encoding used in [10], but not of the one used in [3].)

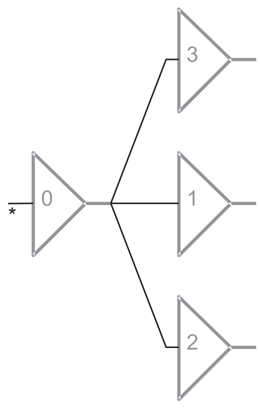


Figure 11: Example of multiplication of neural links when a cell is divided. The ***devel*** genotype is '`<X>N<[*:0]>[-1:10]<><>`' (the corresponding ***recur*** is '`X[*:0][-1:10][-2:10][-3:10]`').

Characteristic	***Simul***	***Recur***	***Devel***
Genotype complexity	Medium	Medium	High
Interpretation complexity	Low	Medium	Medium
Body constraints	None	High	High
Brain constraints	None	Low	Low
Modularity	None	None	High
Compression	None	None	Variable
Redundancy	None	Low	Low

Table 2: Characteristics of the presented genetic encodings.

4.5 Characteristics of the Three Encodings

In this section we attempt to summarize the characteristics of the three encodings, and chart the differences in Table 2. We include the following characteristics in the comparison. Genotype complexity refers to the internal structural complexity of genotypes, and the degree of dependencies between genes[5]. The ***simul*** and ***recur*** encodings have medium complexity, while the ***devel*** encoding has somewhat higher complexity, due to the repetition structures.

Interpretation complexity refers to the algorithmic complexity of the process of interpretation of a genotype. ***Simul*** direct encoding requires very low overhead, while the two higher level encodings require extra memory. Body and brain constraints are present if the encoding inherently limits which phenotypes are expressable. ***Simul*** has no such limitation, while both higher level encodings impose at least one major constraint (no cycles allowed) and some minor ones (some attribute values can be set with limited precision, etc.). Modularity refers to the encoding's ability to produce identical or similar phenotypic units encoded by shared genes. Only ***devel*** qualifies in this category. Compression refers to encodings which can encode a number of phenotypic parts in a lesser number of genotypic parts; this property is related to modularity. Finally, an encoding has redundancy if it permits genotypes with unexpressed portions. This can happen under both ***recur*** and ***devel***, but not extensively.

[5]We use the word 'gene' loosely, referring to the smallest element of a genotype (a letter in the genotype string), or a set of adjacent elements (a substring of the genotype string).

Setting	Value
Population size	200
Cloning probability	20%
Crossing-over probability	16%
Point-mutation probability	64%
Initial distance from the ground	0.1

Table 3: GA settings used in the experiments.

The three encodings differ in several characteristics, which makes it impossible to analyze the effect of one particular feature in isolation. Constructing encodings which differ in no more than one feature would be possible only if these encodings have much in common (e.g., a common meta-encoding). This contradicts our purpose to test fundamentally different approaches to encode organisms. In different encodings operators are inherently different in most cases.

5 Empirical Comparison

In order to analyze the different encodings, we compared their performance against the same tasks. We kept the fitness function and other settings constant in all the runs, and changed only the genetic encoding used. We present results in the case of three optimization tasks: passive height maximization, active height maximization, and locomotion velocity.

The first two tasks required maximization of the average height of the agent, as measured by the geometric center of body parts. In order to limit evolution to static morphologies, in the first task we turned off the simulation of neural networks. This limitation was removed for the second task, which opened up the possibility of movement to enhance fitness. The motion of a creature may increase the height on average, but morphology is still the dominant factor. The height maximization tasks are relatively simple compared to the full potential of the Framsticks system, but enables us to analyze results easily, just by looking at static images.

The third task was maximization of locomotion speed on land. This task required a coordinated set of body parts (limbs), and control structures (effectors, neurons, and usually sensors). Various results for locomotion using Framsticks have been reported previously in [15, 14].

5.1 System parameters

The evolutionary algorithm used was a steady-state GA, with the most important parameters listed in Table 3. Genotypes were selected using tournament selection with size 2. When a genotype was selected for reproduction, it was modified in 80% of the cases. If it was modified, it was mutated (80% probability) or crossed over. If a genotype was not modified, it was cloned, which served to lengthen the average lifetime of genotypes, to get fitness values sampled more than once. This was needed because of the nondeterministic nature of the simulation (random initialization of neuron states). The fitness of a genotype was defined as the average of the fitness values of the multiple individuals sharing the genotype.

These settings were a result of a tedious experimentation and adjustment process. For every combination of genetic encoding and task we executed 10 final runs, for a total of $3 \times 3 \times 10 = 90$ runs.

Setting	Height (passive and active)	Velocity
Performance measurement interval	50	500
Simulation time after stabilization	1000	5000
Number of evaluations in a single run	40,000	60,000

Table 4: Parameters of fitness formula and the number of evaluations in a single evolutionary run.

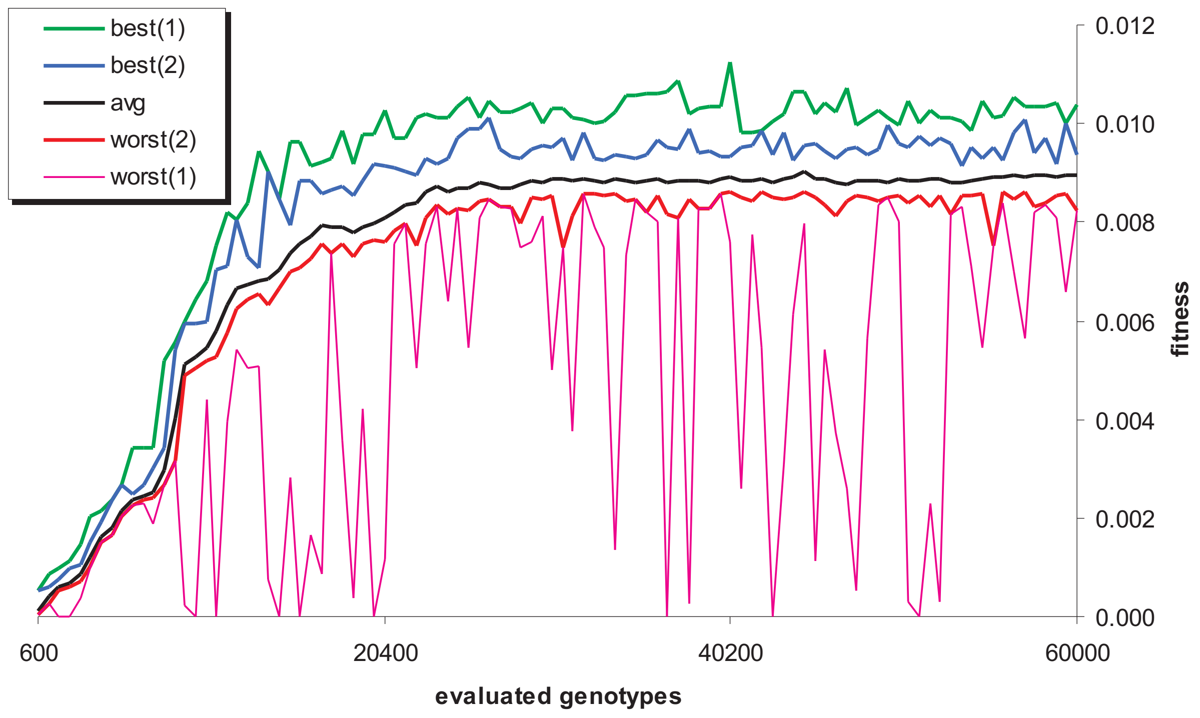


Figure 12: Best, average, and worst fitness values during an evolutionary run.

Further settings are detailed in Table 4. All values concerning time are in simulation steps, and values concerning distance are in simulation units. For comparison, the default stick length is 1.0. Every evaluation was started with a stabilization period, during which no performance measurements were taken, and neurons were kept inactive. This was done in order to prevent the use of the potential energy resulting from the initial placement and position of the creatures.

5.2 Results – Quantitative analysis

In a quantitative analysis the notion of best individual is important, but complicated by the fact that fitness evaluations are non-deterministic. A fitness can depend on how many times a genotype had been evaluated. The more evaluations, the more stable and reliable is a fitness estimate.

We reproduce a typical fitness profile from a velocity-oriented experiment. Fig. 12. plots various fitness values against time. The middle line is the mean fitness of the population. The lines above (best(2)) and below (worst(2)) are the best and worst fitness values of genotypes evaluated *at least two times*. Lines best(1) and worst(1) are the best and worst fitness values of all genotypes

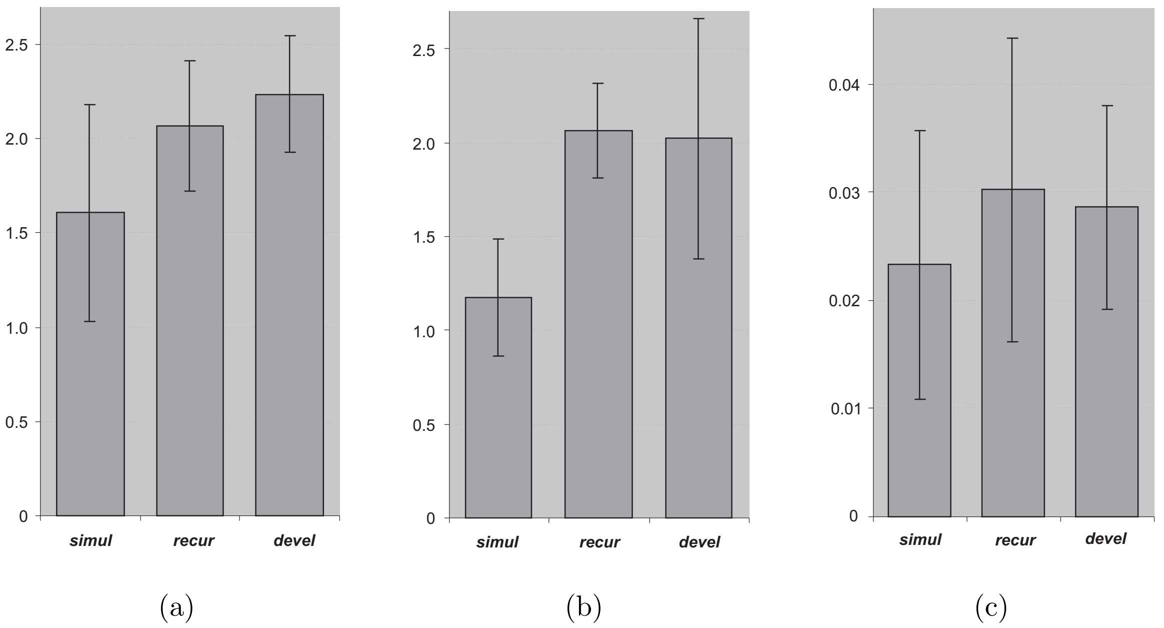


Figure 13: Best fitness values found in the three tasks: (a) passive average height, (b) active average height, (c) average velocity. Mean values are shown with standard deviations.

(including those evaluated only once). As it can be seen, best(1) and worst(1) vary widely. Therefore we decided to define the 'best' genotype as the genotype with highest mean fitness, evaluated at least two times (best(2)). Thus 'best', 'highest', etc., are used with such meaning in the forthcoming discussion.

Charts in Fig. 13 summarize the fitness results for the three tasks and three genetic encodings. The bars show averages of best individuals taken from the 10 runs; standard deviations are also shown. In all three tasks, the worst was the ***simul*** encoding. ***Recur*** and ***devel*** were comparable. Statistically, the difference was important between ***devel*** and ***simul*** in task (a), and between ***recur*** and ***simul***, and ***devel*** and ***simul*** in task (b). In task (c), no differences were statistically significant[6].

It may seem interesting that active height maximization did not yield better results than the passive one. One explanation is that static constructs alone were sufficient to produce solutions very close to physical limits. The fitness result values, taken together with attempts to manually construct agents (described in more detail below), indicate that the value of 2.50 for average height of the center is very hard to exceed. This is due to some physical limitations: the maximum stick length is 2.00, and the high elasticity of sticks and joints makes them unable to bear large weights. The disadvantage of a moving design (e.g., jumping) is instability, and apparently, in this regime the disadvantage was stronger than the benefit. Motion might be better, however, in increasing *maximal* height rather than average height (as in the case of a big leap followed by collapse).

[6]Tests were conducted with the assumption of normal distribution of results and after testing the hypothesis of equal variances. The significance level was 0.01.

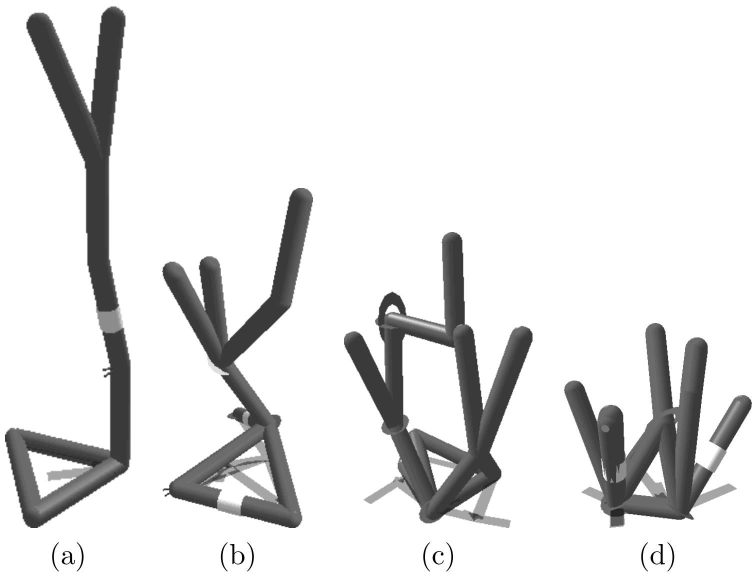


Figure 14: Representative best agents in passive height maximization task, ***simul*** encoding.

5.3 Results – Qualitative analysis

5.3.1 Height, passive agents

In this task three kinds of construction were typical, with variable intensity of branching: from antenna-like creatures through tree- and bush-like creatures (Fig. 14). Typical ***simul*** solutions had a triangular base (90% of agents), which allowed for high stability and stiffness. This shows the importance of cycles in the morphology. Using cycles it is possible to convert some of the compressing forces into pulling forces, which are handled better by sticks. If cycles were not available, we would have expected even lower maximum fitness values.

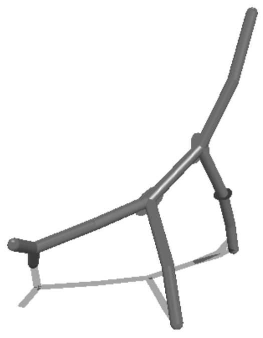
Using the ***simul*** encoding, one of the observed kinds of solutions was a number of sticks erected from a base. Such a structure allowed for a high position of the geometric center with high stability and stiffness (Fig. 14.d). With ***recur***, in 50% of the agents base points were joined not at ground level, but above (Fig. 15.a, b, c). One of the exceptions was the bush-like agent with 4353 parts (Fig. 15.d). In the case of the ***devel*** encoding, the influence of modularity was observed in structures resembling a spiral, a chain, or a segmented backbone (Fig. 16).

Although evolved creatures were stable during their normal evaluation period, in most cases they were knocked out of balance with minimal effort. Some minimal motion could usually be observed even in the case of passive agents, due to elastic forces and non-equilibrium initial conditions. In some cases agents flipped over spontaneously when simulated for times exceeding the length used during the evolution. This shows the extreme degree of adaptation to the given environment and peculiarities of the fitness evaluation.

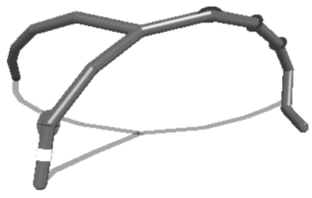
5.3.2 Height, active agents

This task is similar to the passive height task, but with the possibility of using neural networks to generate movement. Many resulting structures were similar to ones from the previous task. In ***simul*** and ***recur***, about 40% of agents appeared to be moving purposefully; the rest were moving in a way that did not deteriorate their fitness or did not move at all (Fig. 17.a, b). A purposeful movement was usually stretching and straightening (Fig. 17.c), with an orientation sensor as the signal source. Among ***devel*** agents no purposeful movement was observed, but some interesting

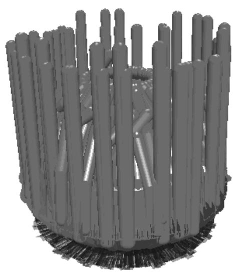
(a)



(b)



(c)



(d)

Figure 15: Representative best agents in passive height maximization task, ***recur*** encoding.

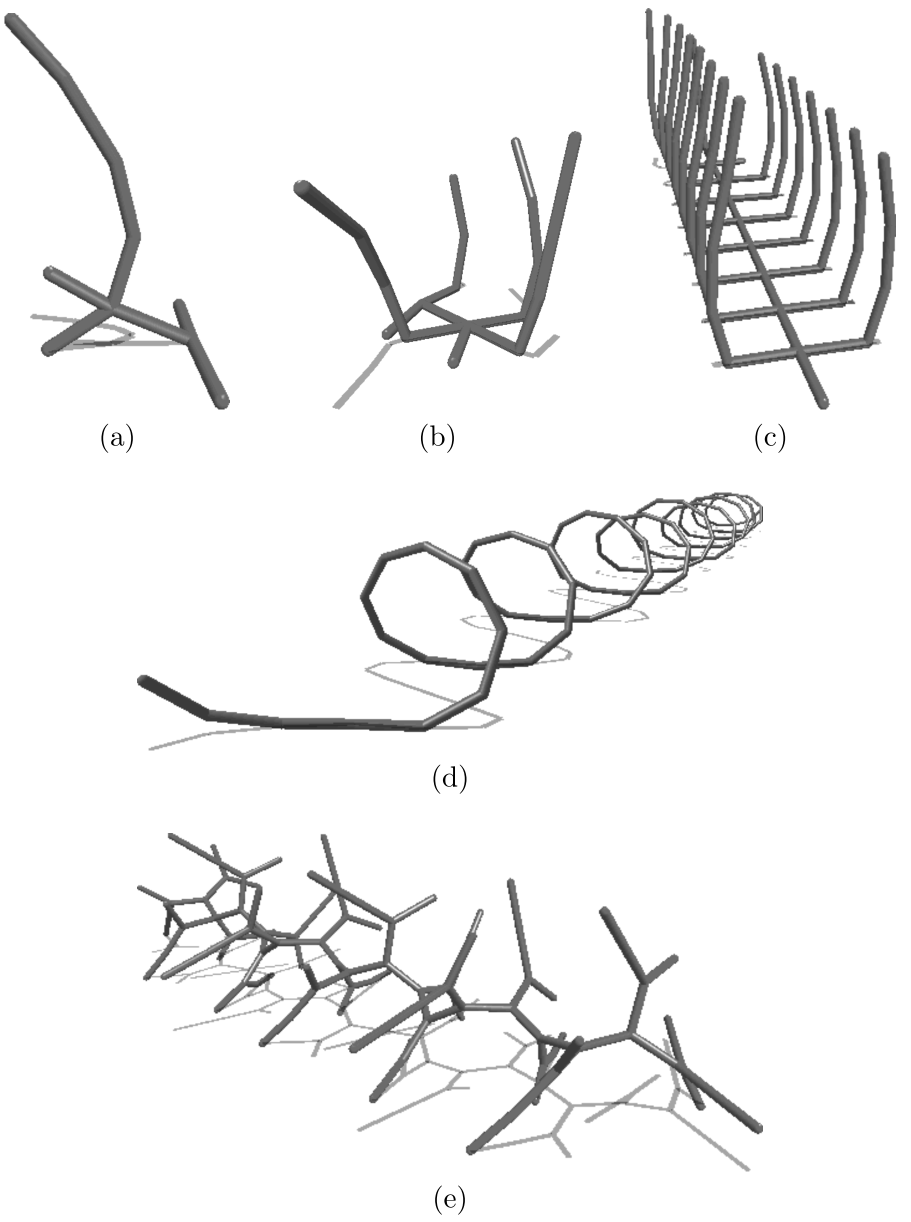


Figure 16: Representative best agents in passive height maximization task, ***devel*** encoding.

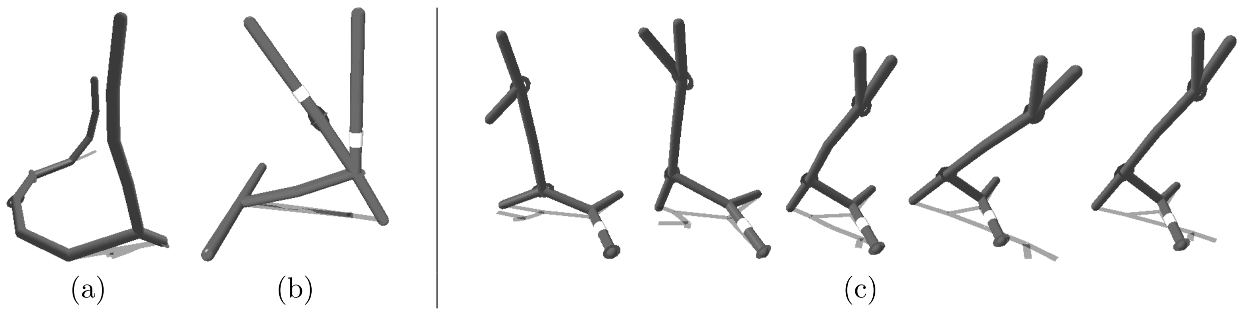


Figure 17: Representative best agents in active height maximization task. ***Recur*** encoding.

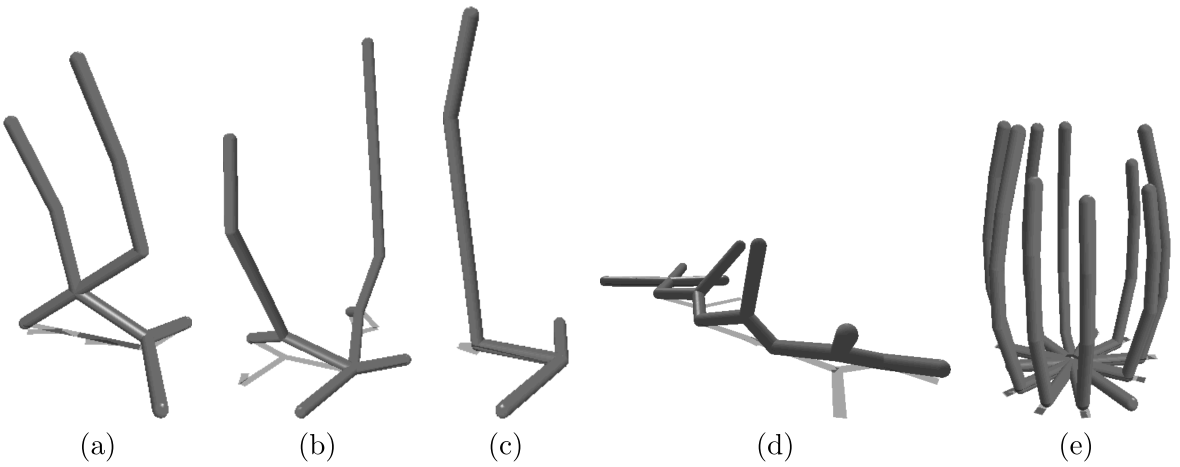


Figure 18: Representative best agents in active height maximization task. ***Devel*** encoding.

constructions emerged (Fig. 18).[7]

5.3.3 Velocity

In this task, evolved agents had small bodies (small weight), and used neural networks with effectors and receptors. This task is not so severely imitated by simple physical constraints as the height tasks, and there was a higher variety among the observed strategies.

Among agents evolved for velocity the most frequently encountered structure consisted of a few sticks, branched or bent (Fig. 20.b). The last stick was moved by a bending muscle, and used as a limb for pushing back. The branching on the other end stabilized the direction of the locomotion (if an agent fell down after a jump, it turned over and got into the same orientation, due to the stabilizing limbs). Consecutive small jumps resulted in locomotion in one direction (Fig. 19.a).

Another popular solution was a construction with two pushing limbs, with a body perpendicular to the direction of the locomotion (Fig. 20.a). Usually only one limb had a muscle, and the other served as stabilization, moving passively, and helping sustain the direction of movement.

[7]Another peculiarity of the height maximization task was noted while trying to design solutions by hand. In a simple active setup, a vertical beam is balanced by a muscle at its base. The muscle needs to switch from pulling to pushing as the beam passes the vertical direction. However, an orientation receptor is most sensitive to the *horizontal* direction, so it cannot be placed directly on the beam. In order to use an orientation sensor, it should reside on a horizontal stick, or large neuron biases need to be used. These solutions are less probable, as they require the co-occurance of several changes.

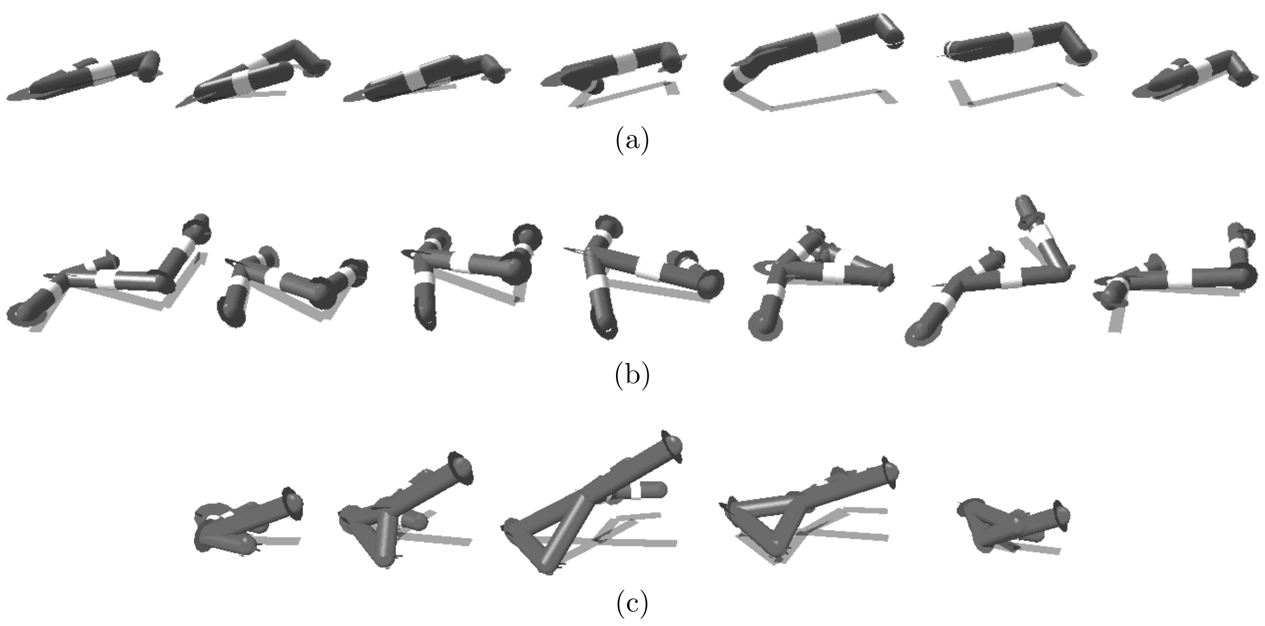


Figure 19: Representative best agents in velocity maximization task. ***Simul*** encoding.

Apart from the common designs, many different strategies could be observed, some illustrated here. The agent shown in Fig. 19.b. had three limbs (two pushing back, one pulling), the one in Fig. 19.c. used a triangular structure for pushing, and the one in Fig. 20.c. had a symmetrical body with two pushing limbs.

Since solutions had small bodies, there were no evident general differences between morphologies in the case of the three encodings. There was no room for segment repetitions and large scale modularity. Subjectively, the movement of ***recur*** and ***devel*** solutions appeared as somewhat biologically more plausible than ***simul*** movements. Neural networks, receptors, and effectors were very precisely tuned to the morphology, showing a tight coupling between morphology and control.

5.4 Comparison to human-designed agents

Designing agents by hand is a very complex process, in professional applications it requires planning and extensive knowledge about how the control system, receptors, and effectors work, as well as knowledge about the simulator. Designing neural networks for control by hand is especially difficult and tedious. For this reason, human-built agents usually have lower fitness than agents produced by evolution. However, human creations are often interesting qualitatively. Human designs have such properties as explicit purpose, elegance, simplicity (minimum of means), and often symmetry and modularity. These features are opposed to evolutionary results, which are characterized by hidden purpose, complexity, implicit and very strong interdependencies between parts, as well as redundancy and randomness.

The difficult process of designing neural networks can be circumvented by a hybrid solution: bodies can be hand-constructed, and control structures evolved for it. It is possible to turn off evolution of body parts (using ***simul*** or ***recur***). This approach can yield interesting creatures [1, 12, 14, 16], often resembling creatures found in nature.

We tried to design agents by hand for the presented tasks, without much success. The locomotion task is simply too complex for a new good solution to be designed by hand in a reasonable amount of time. We expected to be more successful in the simpler task of active height, but we managed to produce higher scores in only a single, extreme case. A ***simul*** structure in the form of the

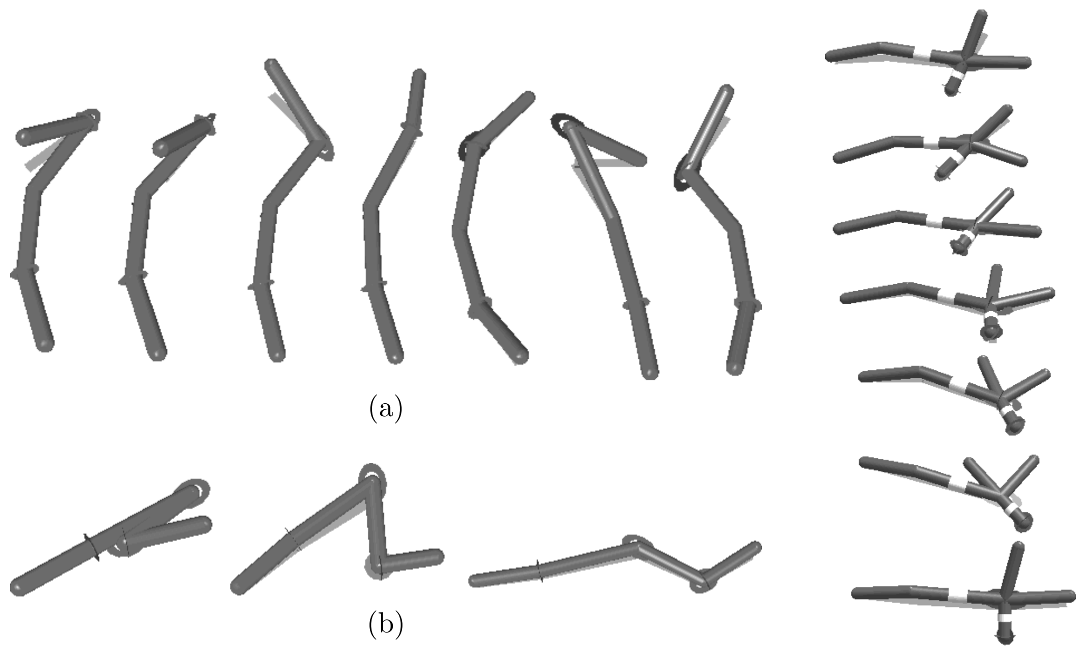


Figure 20: Representative best agents in velocity maximization task. (a, b) ***Recur*** encoding, (c) ***devel*** encoding.

edges of two cubes, one on top of the other, with carefully designed stick lengths and coordinates, obtained a fitness (2.51) above the best found in evolution. This solution is special because it had a clear overall design, and relied on some coordinates of different endpoints being precisely the same ('exact coordinates'), which rarely happens during evolution. Moreover, this structure employs cycles, so comparison to results from ***recur*** and ***devel*** is unfair, since cycles are not available in those encodings. The fact that a single hand-crafted design scored higher does not diminish the results of the evolution but rather strengthens them, because of the peculiar circumstances required.

6 Conclusions

Three different encodings were tested in the presented experiments, and the solutions produced are considered to be successful for the given tasks in all three cases. However, there were some important differences in the degree of success. The ***simul*** encoding performed worse than the two higher-level encodings. The most important differences between these encodings are that ***simul*** has a minimal bias and is unrestrictive, while the higher-level encodings (***recur*** and ***devel***) restrict the search space, and introduce a strong bias towards structured phenotypes. Also, the genetic operators of the higher-level encodings are generally less disruptive than the operators working on lower-level genotypes. Note that the issues of genetic operators and the imposed structure of an encoding are strongly related. Based on our results, we conclude that a more structured genotype encoding, with genetic operators working on a higher level, is beneficial in the evolution of 3D agents. The presence of a bias towards structured phenotypes can overcome the apparent limitation that entire regions of the search space are not accessible by the search.

Comparing ***devel*** to ***recur***, we found that the two performed similarly, although ***devel*** is somewhat more complex than ***recur***. In this case the extra complexity of the encoding (support for modularity) did not result in better solutions. Modularity in evolved agents was not very frequent (in the case of velocity task, very rare), which we attribute to the fact that the selected tasks were not very complex.

Another conclusion supported by results presented in this article is that despite similar fitness values, bias introduced by the encoding may be useful in some applications (engineering and robotics, for example). Different structures yield different dynamics, energy consumption, durability etc., although these criteria were not considered in the optimizations reported here. The significant influence of a chosen encoding can be clearly seen in the agents we obtained: those with low-level direct encoding ***simul*** displayed neither order nor structure. The two encodings restricting morphology to a tree produced more clear constructions, and for developmental encoding segmentation and modularity could be observed.

Each higher-level encoding introduces a specific bias into solutions. Introducing a new, sophisticated genotype encoding may be worthwhile even if it does not lead to higher fitness solutions, because it may facilitate the emergence of new evolutionary 'ideas'. Evolutionary search biased towards different areas of the search space usually results in qualitatively different solutions.

On a general note, it was found that the apparent redundancy in evolved individuals was often an illusion, as any minor change proved deleterious. This shows the strength of implicit relations between parts within the evolved agents. It was usually impossible to construct solutions by hand which exceeded or matched the fitness of evolved individuals. These findings indicate that solutions found by evolution were highly optimal and difficult to improve upon.

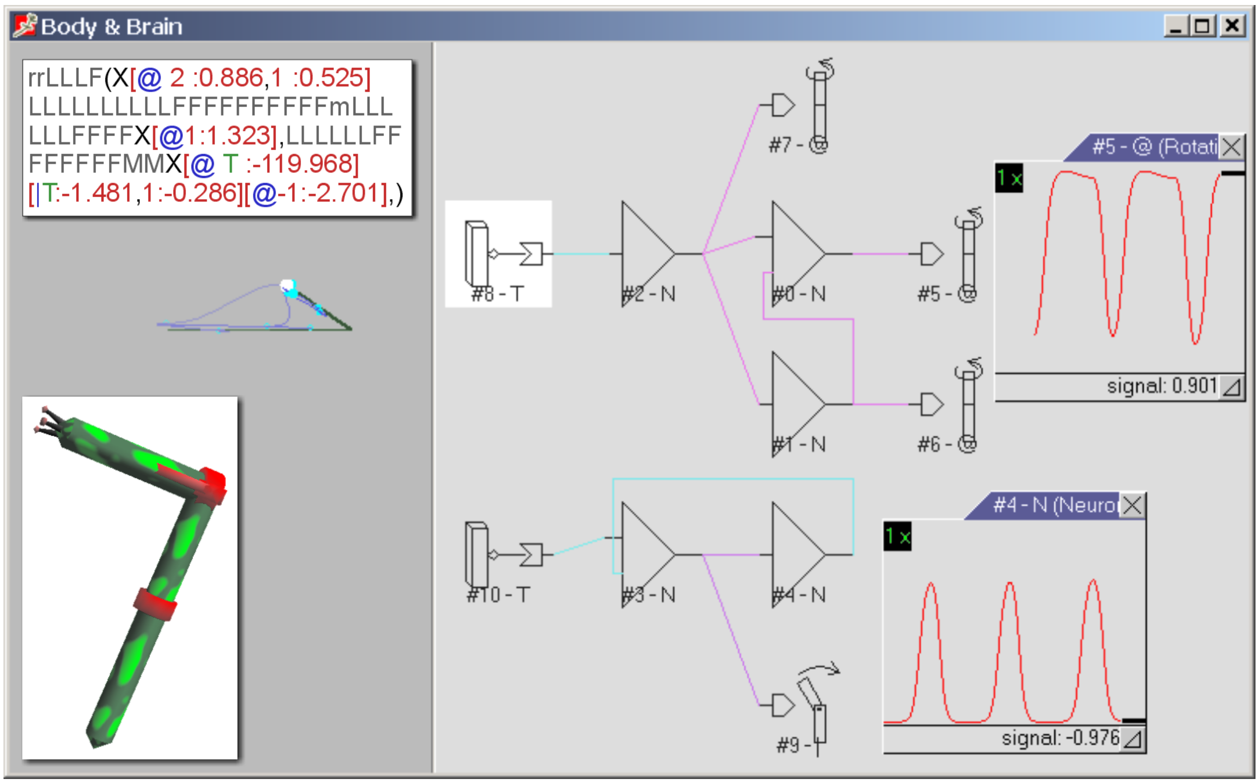
Future work

Work is currently under way on the Framsticks system to implement a universal environment for various experiments. Many experiments could be performed in addition to the ones presented here, of which we mention only a few. Estimates for the 'ruggedness' of an imposed fitness landscape allows testing of the conjecture about the 'smoothness' of various encodings, without the bias of the evolutionary algorithm [22]. Control experiments using ***devel*** with support for repetition turned off can be used to isolate the effect of this particular feature.

Similar experiments should be conducted with different, more complex tasks, and specifically, with tasks that require multiple behaviors (e.g., locomotion and foraging). Numerous ideas exist regarding new encodings, including a similarity-based encoding, and an encoding based on simple models of cell metabolism, which allow for a gradual way of modeling interactions between body parts during development. Finally, work is under way on applications of a phenotype similarity measure, which allows for speciation and automatic taxonomy analysis [13].

Appendix

We include here a screenshot from the Framsticks simulator (Fig. 21), showing a detailed view of an agent. The picture shows the body and brain of a ***recur*** creature from a speed-oriented experiment. The left pane shows a schematic view of the body (neural links are shown in blue, the body is shown in black). The inset in the upper-left corner is the corresponding genotype. The inset in the lower-left corner is a 'solid' rendering of the same creature. The right pane shows the neural network of the creature. Receptors and effectors are shown as small icons, two touch sensors on the left, and four muscles on the right (three rotating and one bending). Two smaller windows graph the current output signal of two chosen neurons. The color of neural links reflects the signal value. Note that one touch sensor is selected (white square); its location is shown in body (white small filled circle).



```
rrLLLF(X[@ 2 :0.886,1 :0.525]
LLLLLLLLLLFFFFFFFFFFmLLL
LLLFFFFX[@1:1.323],LLLLLLFF
FFFFFFMMX[@ T :-119.968]
[|T:-1.481,1:-0.286][@-1:-2.701],)
```

Figure 21: Screenshot of a creature inspection window from the Framsticks application. See text for details.

References

[1] A. Adamatzky, M. Komosinski, and S. Ulatowski. Software review: Framsticks. *Kybernetes: The International Journal of Systems and Cybernetics*, 29(9/10):1344–1351, 2000.

[2] J. C. Bongard and C. Paul. Investigating morphological symmetry and locomotive efficiency using virtual embodied evolution. In J.-A. Meyer and et al., editors, *From Animals to Animats: The Sixth International Conference on the Simulation of Adaptive Behaviour*, 2000.

[3] J. C. Bongard and Pfeifer R. Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In L. Spector and et al., editors, *Proceedings of The Genetic and Evolutionary Computation Conference, GECCO-2001*, pages 829–836. Morgan Kaufmann, 2001.

[4] R. Calabretta, S. Nolfi, D. Parisi, and G. P. Wagner. A case study of the evolution of modularity: Towards a bridge between evolutionary biology, artificial life, neuro- and cognitive science. In C. Adami, R. Belew, H. Kitano, and C. Taylor, editors, *Artificial Life VI: The 6th International Conference on the Simulation and Synthesis of Living Systems*, pages 275–284. MIT Press, 1998.

[5] F. Dellaert and Randall D. Beer. A developmental model for the evolution of complete autonomous agents. In P. Maes, M. J. Mataric, J.-A. Meyer, J. B. Pollack, and S. W. Wilson, editors, *From Animals to Animats. Proceedings of the 4th International Conference on Simulation of Adaptive Behavior, SAB 1996*, pages 393–401. MIT Press, Cambridge, Mass., 1996.

[6] P. Eggenberger. Evolving morphologies of simulated 3d organisms based on differential gene expression. In P. Husbands and I. Harvey, editors, *Proceedings of the 4th European Conference on Artificial Life (ECAL97)*. MIT Press, 1997.

[7] F. Gruau. Modular genetic neural networks for 6-legged locomotion. In J.-M. Alliot and et al., editors, *Artificial Evolution European Conference*, pages 201–219, 1996.

[8] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. R. Riolo, editors, *Proceedings of the First Annual Conference on Genetic Programming 1996*, pages 81–89. MIT Press, Cambridge, Mass., 1996.

[9] W. Hordijk. Population flow on fitness landscapes. Master's thesis, 1994.

[10] G. S. Hornby and J. B. Pollack. Body-brain co-evolution using l-systems as a generative encoding. In L. Spector and et al., editors, *Proceedings of The Genetic and Evolutionary Computation Conference, GECCO-2001*. Morgan Kaufmann, 2001.

[11] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476, 1990.

[12] M. Komosinski. The world of framsticks: Simulation, evolution, interaction. In *Proceedings of 2nd International Conference on Virtual Worlds (VW2000)*, volume LNAI 1834, pages 214–224. Springer Verlag, 2000.

[13] M. Komosinski, G. Koczyk, and M. Kubiak. On estimating similarity of artificial and real organisms. *Theory in Biosciences*, 120, 2001.

[14] M. Komosinski and A. Rotaru-Varga. From directed to open-ended evolution in a complex simulation model. In M. Bedau, McCaskill, Packard, and Rasmussen, editors, *Proceedings of 7th International Conference on Artificial Life (ALIFE7)*, pages 293–299. Morgan Kaufmann, 2000.

[15] M. Komosinski and S. Ulatowski. Framsticks: Towards a simulation of a nature-like world, creatures and evolution. In *Proceedings of 5th European Conference on Artificial Life (ECAL99)*, volume LNAI 1674, pages 261–265. Springer Verlag, 1999.

[16] Maciej Komosinski and Szymon Ulatowski. Framsticks web site. `http://www.framsticks.com`.

[17] Mathias and D. Whitley. Transforming the search space with gray coding. In *IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 1994.

[18] J. B. Pollack, Lipson H., Ficici S. G., P. Funes, G. S. Hornby, and Watson R. A. Evolutionary techniques in physical robotics. In J. Miller, editor, *Evolvable Systems: from Biology to Hardware; Proceedings of the Third International Conference (ICES 2000)*, volume LNCS 1801. Springer Verlag, 2000.

[19] A. Rotaru-Varga. Modularity in evolved artificial neural networks. In D. Floreano, J.-D. Nicoud, and F. Mondada, editors, *Proceedings of 5th European Conference on Artificial Life (ECAL99)*, pages 256–260. Springer Verlag, 1999.

[20] K. Sims. Evolving 3d morphology and behaviour by competition. In R. Brooks and et al., editors, *Proceedings of the Artificial Life IV Conference*, pages 28–39. MIT Press, 1994.

[21] T. Taylor and C. Massey. Recent developments in the evolution of morphologies and controllers for physically simulated creatures. *Artificial Life*, 7(1):77–88, 2001.

[22] V. K. Vassilev. An information measure of landscapes. In T. Bäck, editor, *Proceedings of 7th International Conference on Genetic Algorithms (ICGA97)*, pages 49–56. Morgan Kauffman, 1997.

[23] D. L. Whitley, F. Gruau, and L. Pyeatt. Cellular encoding applied to neurocontrol. In L. Eshelmann, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*. Morgan Kauffman, 1995.