Evolution of Generative Design Systems for Modular Physical Robots

Gregory S. Hornby Hod Lipson Jordan B. Pollack

DEMO Lab. Computer Science Dept. Brandeis University Waltham, MA 02454

Abstract

Recent research has demonstrated the ability for automatic design of the morphology and control of real physical robots using techniques inspired by biological evolution. The main criticism of the evolutionary design approach, however, is that it is doubtful whether it will reach the high complexities necessary for practical engineering. Here we claim that for automatic design systems to scale in complexity the designs they produce must be made of re-used modules. Our approach is based on the use of a generative design grammar subject to an evolutionary process. Unlike a direct encoding of a design, a generative design specification can re-use components, giving it the ability to create more complex modules from simpler ones. Re-used modules are also valuable for improved efficiency in testing and construction. We describe a system for creating generative specifications capable of hierarchical modularity by combining Lindenmayer systems with evolutionary algorithms. Using this system we demonstrate for the first time a generative system for physical, modular, 2D locomoting robots and their controllers.

1 Introduction

Recent research has demonstrated the ability for automatic design of engineering products using techniques inspired by biological evolution, [1], [2], [3], [4], [5] and [6]. The main criticism for the use of evolutionary computation for design is that it is doubtful whether it will reach the high complexities necessary for practical engineering. Since the search space grows exponentially with the size of the problem, search algorithms that use a direct encoding for designs will not scale to large designs. An alternative to a direct encoding is a generative specification, which is a grammatical encoding that specifies how to construct a design, [7] and [8]. Similar to a computer program, a generative specification can allow the definition of re-usable sub-procedures allowing the design system to scale to more complex designs than can be achieved with a direct encoding [9].

Ideally an automated design system would start with a library of basic parts and would iteratively create new, more complex modules, from ones already in its library. The principle of modularity is well accepted as a general characteristic of good design, as it typically promotes decoupling and reduces complexity [10]. In contrast to a design in which every component is unique, a design built with a library of standard modules requires less time to verify and test each unique component in the design because there are fewer unique components. Reduced complexity makes manufacturing easier because there are fewer unique components to construct, which also leads to a smaller stockpile of spare parts necessary for maintenance and repair. Similarly, decoupling leads to higher ease of adaptability.

In the following section we outline the design space and describe the components of our generative design system. We then demonstrate both virtual and physical working robots whose morphology and control has been designed by the system.

2 Method

Our design space consists of a Tinker-Toy^{TM} like world where robots are built from bars of regular length and both fixed and actuated joints, figure 1. Actuated joints cycle through 60° with variable frequency and relative phase offset. Static joints accommodate bars at fixed 60° angles.

A Lindenmayer system (L-system) is used as the generative specification language for designs and is optimized by an evolutionary algorithm (EA). Using this system we produce 2D mechanisms and their controllers for locomotion in simulation. Designs produced by our system start with a basic library of sin-

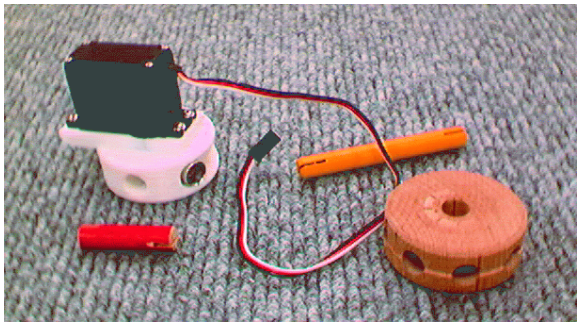


Figure 1: Basic building blocks of the system: bars of regular length and fixed and actuated joints.

gle bars and actuated joints with which they create, and use, more complex modules. The resulting robots are easily constructed from our basic Tinker-Toy-like components.

EAs are a stochastic search and optimization technique inspired by natural evolution [11] and [12]. An EA maintains a population of candidate solutions from which it performs search by iteratively replacing poor members of the population with individuals generated by applying variation to good members of the population.

L-systems are a grammatical rewriting system introduced to model the biological development of multicellular organisms [13]. Rules are applied in parallel to all characters in the string just as cell divisions happen in parallel in multicellular organisms. Complex objects are created by successively replacing parts of a simple object by using the set of rewriting rules.

L-systems and EAs have been used both on their own and together to create designs. L-systems have been used mainly to construct tree-like plants, [14]. However, it is difficult to hand-make an L-system to produce a desired form. EAs have been used to evolve bridges [4], wing shapes [3], flywheel designs and others [5]. Although useful for creating good and novel solutions to problems, the results are strongly dependent on the representation being evolved, with most representations directly parameterizing a prespecified model. L-systems have been combined with EAs in previous work – such as the evolution of plant-like structures [14], [15], and [16] and architectural floor designs [17] – but only limited results have been achieved. Except for [4], only a shape rendered on a computer was created.

Automatic creation of robot morphology and controllers has been done previously by [18], [19] and [6], all of which used EAs to simultaneously create the morphology and a neural controller in simulation. Whereas [18] and [19] were not concerned with the feasibility of their creations in reality, the focus of [6] was to show that creatures created through evolution in simulation could be successfully transferred to reality. This work extends the latter results by using a design encoding to achieve mechanisms with a higher piece-count and a more complex regularity of form. This demonstrates that our system is re-using discovered modules thereby allowing it to scale to more complex designs. We show that this method is feasible for the production of real robots by successfully transferring an evolved design to reality.

The system for creating generative designs consists of the design builder and simulator, the L-system module and the evolutionary algorithm. L-systems are evolved by the evolutionary algorithm. Individual L-systems are scored for their goodness by the design builder and simulator. These modules are described in the following sections.

2.1 Design Builder and Simulator

The design constructor builds a model from a sequence of build commands. Once built, a model is simulated and evaluated.

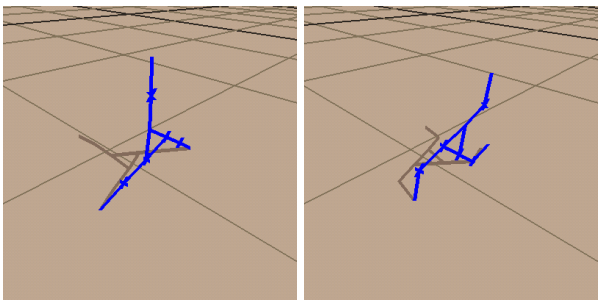
Table 1: Design Language

Command	Description
[ ]	push/pop orientation to stack
{ *block* }(n)	repeat enclosed block n times
forward(n)	add bar of length n
backward(n)	move up n levels of parents
joint(n)	forward, end with an actuated joint which moves at speed n
clockwise(n)	rotate heading clockwise $n \times 60°$
counter-clockwise(n)	rotate heading counterclockwise $n \times 60°$
increase-offset(n)	increase phase offset by $n \times 25\%$
decrease-offset(n)	decrease phase offset by $n \times 25\%$

The command string consists of a sequence of build commands that give instructions to a LOGO-style turtle that is used to construct a creature from bars. Commands are listed in table 1. [and] push and pop

the current state - consisting of the current bar, the orientation and joint oscillation offset - to and from a stack. Forward moves the turtle forward in the current direction, creating a bar if none exists or traversing to the end of the existing bar. Backward goes back up the parent of the current bar. Clockwise and counter-clockwise rotate the orientation in steps of 60°. And joint is the same as forward except that if a new bar is created, it ends with an actuated joint which oscillates over a range of 60°. The parameter to this commands specifies the speed at which the joint oscillates, using integer values from 1 to 5, and the relative offset of the oscillation cycle is taken from the turtle's state. increase-offset and decrease-offset change the offset value in the turtle's state by ±25% of a total cycle. Command sequences enclosed by { } are repeated a number of times specified by the brackets' argument.

Once an L-system specification is executed and the resulting creature is constructed in simulation, its movements are evaluated in a quasi-static kinematics simulator, similar to that used by Lipson and Pollack [6]. The kinematics are simulated by computing successive frames, differing by small angular increments of actuators. An update consists of moving each actuated joint by joint-speed×0.024°, and then re-settling the structure to a stable orientation.



(a) (b)

Figure 2: Sample L-creature.

For example, the string,
{ *joint*(1) [*joint*(1) *forward*(1)] *clockwise*(2) }(3),
is interpreted as: *joint(1) [joint(1) forward(1)] clockwise(2) joint(1) [joint(1) forward(1)] clockwise(2) joint(1) [joint(1) forward(1)] clockwise(2)* and produces the creature in figure 2. X's are used to show the location of actuated joints. The left image shows the creature with all actuated joints in their starting orientation and the image on the right shows the same creature with all actuated joints at the other extreme of their actuation cycle. In this example all actuated joints are moving in phase.

2.2 Parametric 0L-Systems

The class of L-systems used as the encoding is a parametric, context-free L-system (P0L-system). Formally, a P0L-system is defined as an ordered quadruplet, G = (V, Σ, ω, P) where,
V is the alphabet of the system,
Σ is the set of formal parameters,
$\omega \in (V \times \Re^*)^+$ is a nonempty parametric word called the axiom, and
$P \subset (V \times \Sigma^*) \times C(\Sigma) \times (V \times \xi(\Sigma))^*$ is a finite set of productions.
The symbols : and → are used to separate the three components of a production: the predecessor, the condition and the successor. For example, a production with predecessor A(n0, n1), condition n1>5 and successor B(n1+1)cD(n1+0.5, n0-2) is written as:

$$A(n0, n1) : n1 > 5 \rightarrow B(n1{+}1)cD(n1{+}0.5, n0{-}2)$$

A production matches a module in a parametric word iff the letter in the module and the letter in the production predecessor are the same, the number of actual parameters in the module is equal to the number of formal parameters in the production predecessor, and the condition evaluates to true if the actual parameter values are substituted for the formal parameters in the production.

For implementation reasons we add constraints to our P0L-system. The condition is restricted to be comparisons as to whether a production parameter is greater than a constant value. Parameters to design commands are either a constant value or a production parameter. Parameters to productions are equations of the form: [*production parameter* | *constant*] [+ | − | × | \ ] [*production parameter* | *constant*]. The following is a P0L-system using the language defined in table 1 and consists of two productions with each production containing two condition-successor pairs:

$$\begin{aligned} P0(n): \quad & n > 2 \rightarrow \{\ P0(n-1)\ \}(n) \\ & n > 0 \rightarrow joint(1)\ P1(n \times 2)\ clockwise(2) \end{aligned}$$

$$\begin{aligned} P1(n): \quad & n > 2 \rightarrow [\ P1(n/4)\ ] \\ & n > 0 \rightarrow joint(1)\ forward(n) \end{aligned}$$

If the P0L-system is started with P0(3), the resulting

sequence of strings is produced:

$P0(3)$
{ $P0(2)$ }(3)
{ $joint(1)$ $P1(4)$ $clockwise(2)$ }(3)
{ $joint(1)$ [$P1(1)$] $clockwise(2)$ }(3)
{ $joint(1)$ [$joint(1)$ $forward(1)$] $clockwise(2)$ }(3)

This produces the creature in figure 2.

2.3 Evolutionary Algorithm

An evolutionary algorithm is used to evolve individual L-systems. The initial population of L-systems is created by making random production rules. Evolution then proceeds by iteratively selecting a collection of individuals with high fitness for parents and using them to create a new population of individual L-systems through mutation and recombination.

Mutation creates a new individual by copying the parent individual and making a small change to it. Changes that can occur are: replacing one command with another; perturbing the parameter to a command by adding/subtracting a small value to it; changing the parameter equation to a production; adding/deleting a sequence of commands in a successor; or changing the condition equation.

Recombination takes two individuals, $p1$ and $p2$, as parents and creates one child individual, c, by making it a copy of $p1$ and then inserting a small part of $p2$ into it. This is done by replacing one successor of c with a successor of $p2$, inserting a sub-sequence of commands from a successor in $p2$ into c, or replacing a sub-sequence of commands in a successor of c with a sub-sequence of commands from a successor in $p2$.

3 Experiments and Results

We now describe how our system has been used to create modular designs that locomote in simulation and were shown to work in reality.

To create designs we run our evolutionary algorithm with 100 individuals for a maximum of 500 generations. The P0L-systems used have fifteen productions with each production having two parameters and three sets of condition-successor pairs. Fitness is a function of the distance moved by the creature's center of mass after 10 simulated seconds.

Evolutionary runs were similar in many ways. The first individuals would mainly move their center of mass while remain in a fixed location. Typically individuals with a repeating locomotion cycle would appear by generation 30 and these would compete to be the dominant design in the population. Once converged to one design the EA would try many variations of it making slow and steady improvements. Periodically a new version would have a significantly greater fitness and then the population would converge to this variant.

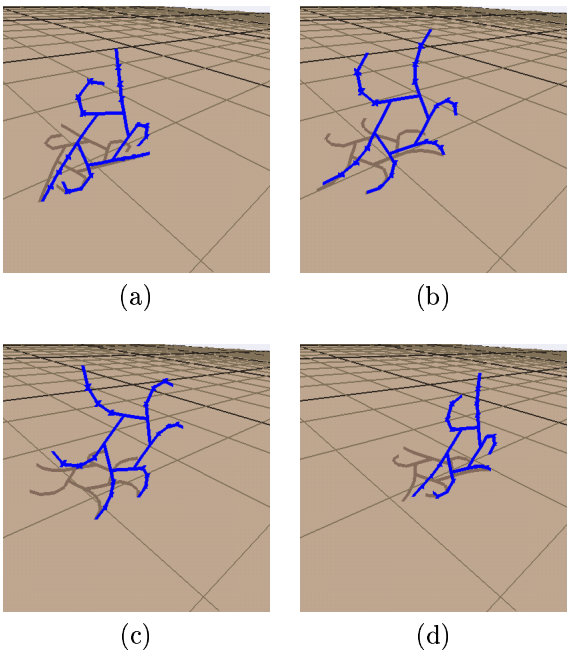


Figure 3: The locomotion cycle of a walking star creature, built from 43 bars and 24 actuated joints. Notice the regularity of structure.

Each evolutionary run was different, resulting in a variety of different creatures, but all creatures could be classed into one of the following families: crawlers who would use one appendage to drag the rest of the body forward; walkers who used legs to move with little dragging; inch-worms that inched along; and rollers that rotated their whole body to move, see figures 3 and 4.

Finally, we construct an actual robot from an evolved design, the walking M in figure 4. Figure 5 shows two parts of the locomotion cycle of a walking-M creature. With its two outer arms 25% out of phase it moves by bringing them together to lift its middle arm and then to shift its center of mass to the right. One modification to the constructed robot is the addition of sandpaper on the feet of the two outer arms to compensate for the friction modeled by our simulator and that of the actual surface used.

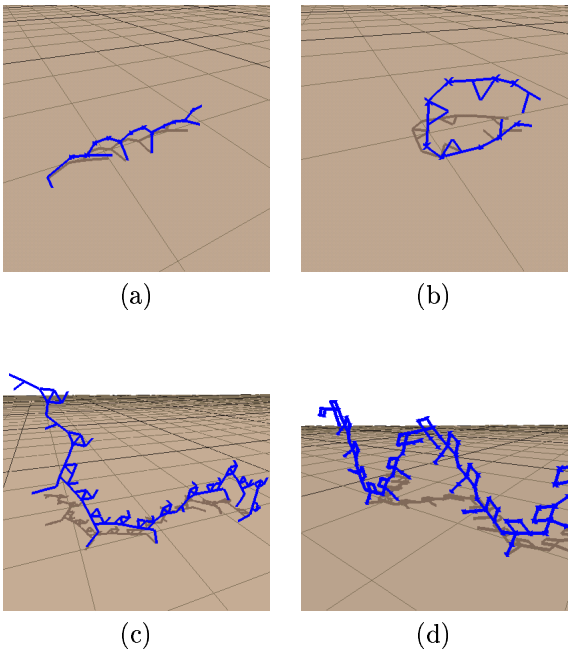


Figure 4: Evolved creatures: *a*, a multi-legged walker with 24 bars and 12 actuated joints; *b*, a rolling circle with 23 bars and 6 actuated joints; *c*, an inch-worm built from 143 bars and 16 actuated joints; and *d*, an undullating serpent with 164 bars and 61 actuated joints; Notice the re-use of components.

4 Discussion

The main difference between this and previous work is that this work combines for the first time an evolutionary algorithm with a generative grammatical encoding for the evolution of working robots. By acting as a kind of program – complete with variables, loops and subroutines – the generative encoding allows for the same part of the encoding to be re-used, which makes certain types of design changes easier. For example, early versions of the star-creature, figure 3, used three actuated joints in the legs instead of four. For EAs, or any optimization algorithm, to move to legs with four actuated joints only one change to the leg-building code needed to occur. With a direct encoding, this change would need to take place in all six uses of the leg simultaneously.

Carrying over the notion of modularity from the virtual to the physical world, our work uses modular building blocks that are hand-assembled to produce actual robots. Robots constructed with these parts can later be disassembled so that the parts can be re-used to build other robots. In principle, we are also capable of automatic manufacture of these robots in

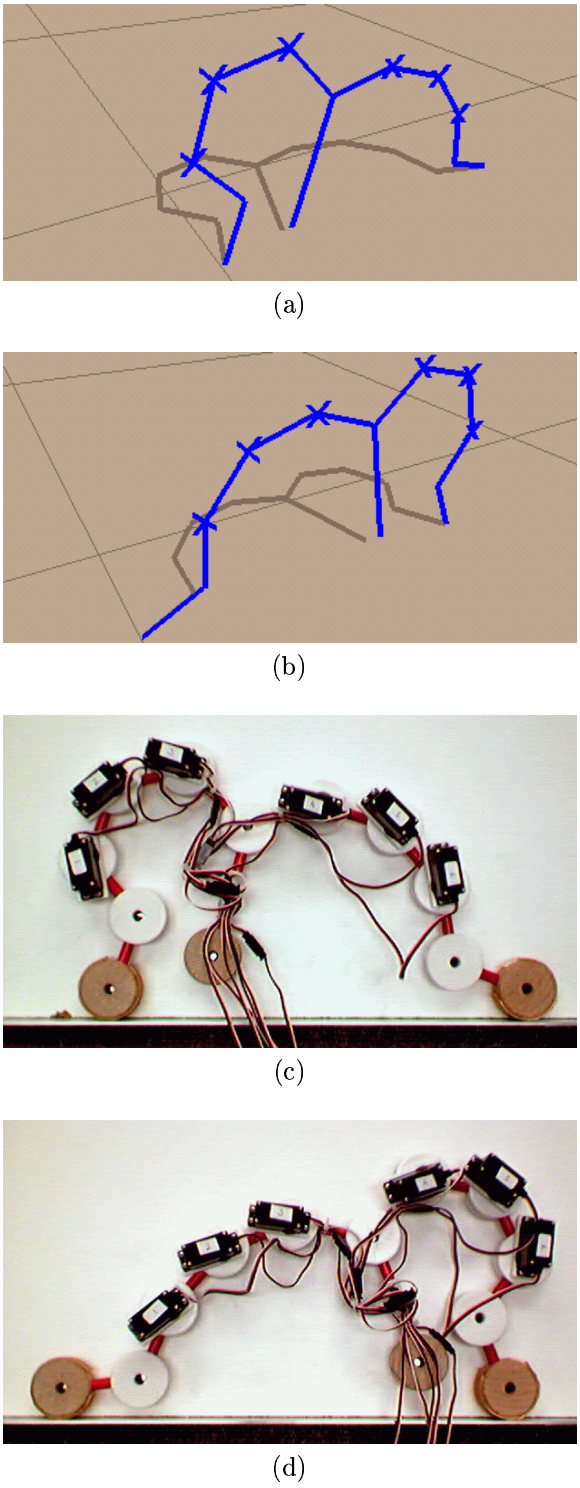


Figure 5: Two parts of the locomotion cycle of a walking creature – *a* and *b* are from simulation, *c* and *d* from the physical robot. Notice the repetition and the symmetry.

addition to our automated design using rapid prototyping techniques (although motors and circuits would still need to be hand-assembled).

This system also has applications for reconfigurable robots. Using a single robot module as a basic building block, group morphology and control strategies can be developed.

5 Conclusion

A system for automatically producing generative design systems with repeated components was achieved by using evolutionary algorithms to optimize parametric Lindenmayer-systems. Using this system 2D robots for locomotion with regular forms were designed consisting of more than 100 components. An evolved robot was successfully transferred to the real world.

Acknowledgements

This research was supported in part by the Defense Advanced Research Projects Administration (DARPA) Grant No. DASG60-99-1-0004. The authors would like to thank the members of the DEMO Lab: A. Bucci, E. DeJong, S. Ficici, P. Funes, S. Levy, O. Melnik, S. Viswanathan and R. Watson.

References

[1] Couro Kane and Marc Schoenauer. Genetic operators for two-dimentional shape optimization. In J.-M. Alliot, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificiale Evolution - EA95*. Springer-Verlag, 1995.

[2] P. J. Bentley. *Generic Evolutionary Design of Solid Objects Using a Genetic Algorithm*. PhD thesis, University of Huddersfield, 1996.

[3] P. Husbands, G. Germy, M. McIlhagga, and R. Ives. Two applications of genetic algorithms to component design. In T. Fogarty, editor, *Evolutionary Computing. LNCS 1143*, pages 50–61. Springer-Verlag, 1996.

[4] P. Funes and J. Pollack. Computer evolution of buildable objects. In Phil Husbands and Inman Harvey, editors, *Proceedings of the Fourth European Conference on Artificial Life*, pages 358–367, Cambridge, MA, 1997. MIT Press.

[5] P. J. Bentley, editor. *Evolutionary Design by Computers*. Morgan Kaufman, 1999.

[6] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978, 2000.

[7] Marc Schoenauer. Shape representations and evolution schemes. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Evolutionary Programming 5*. MIT Press, 1996.

[8] Peter Bentley and Sanjeev Kumar. Three ways to grow designs: A comparison of embryogenies of an evolutionary design problem. In Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiel, and Smith, editors, *Proc. Genetic and Evolutionary Computation Conference*, pages 35–43, 1999.

[9] Gregory S. Hornby and Jordan B. Pollack. The advantages of generative grammatical encodings for physical design. In *Congress on Evolutionary Computation*, 2001.

[10] Nam P. Suh. *The Principles of Design*. Oxford University Press, 1990.

[11] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[12] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In Richard K. Belew; Lashon B. Booker, editor, *Proc. of the Fourth Int. Conf. on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, 1991.

[13] A. Lindenmayer. Mathematical models for cellular interaction in development. parts i and ii. *Journal of Theoretical Biology*, 18:280–299 and 300–315, 1968.

[14] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[15] Christian Jacob. Genetic l-system programming. In Y. Davidor and P. Schwefel, editors, *Parallel Problem Solving from Nature, Lecture Notes in Computer Science*, volume 866, pages 334–343. Springer-Verlag, 1994.

[16] G. Ochoa. On genetic algorithms and lindenmayer systems. In A. Eiben, T. Baeck, M. Schoenauer, and H. P. Schwefel, editors, *Parallel Problem Solving from Nature V*, pages 335–344. Springer-Verlag, 1998.

[17] Paul Coates, Terence Broughton, and Helen Jackson. Exploring three-dimensional design worlds using lindenmayer systems and genetic programming. In Peter J. Bentley, editor, *Evolutionary Design by Computers*. Morgan Kaufmann, 1999.

[18] Karl Sims. Evolving 3d morphology and behavior by competition. In R. Brooks and P. Maes, editors, *Proceedings of the Fourth Workshop on Artificial Life*, pages 28–39, Boston, MA, 1994. MIT Press.

[19] Maciej Komosinski and Adam Rotaru-Varga. From directed to open-ended evolution in a complex simulation model. In Bedau, McCaskill, Packard, and Rasmussen, editors, *Artificial Life 7*, pages 293–299. Morgan Kaufmann, 2000.