Evolutionary Robotics and Open-Ended Design Automation

Hod Lipson, Cornell University

Can a computer ultimately augment or replace human invention?

IMAGINE A LEGO SET AT YOUR DISPOSAL: Bricks, rods, wheels, motors, sensors and logic are your "atomic" building blocks, and you must find a way to put them together to achieve a given high-level functionality: A machine that can move itself, say. You know the physics of the individual components' behaviors; you know the repertoire of pieces available, and you know how they are allowed to connect. But how do you determine the combination that gives you the desired functionality? This is the problem of *Synthesis*. Although engineers practice it and teach it all the time, we do not have a formal model of how open-ended synthesis can be done *automatically*. Applications are numerous. This is the meta-problem of engineering: Design a machine that can design other machines.

The example above is confined to electromechanics, but similar synthesis challenges occur in almost all engineering disciplines: Circuits, software, structures, robotics, control, and MEMS, to name a few. Are there fundamental properties of design synthesis that cut across engineering fields? Can a computer ultimately augment or replace human invention?

While we may not know how to synthesize thing automatically, nature may give us some clues: After all, the fascinating products of nature were designed and fabricated autonomously.

Introduction

In the last two centuries, engineering sciences have made remarkable progress in the ability to analyze and predict physical phenomena. We understand the governing equations of thermodynamics, elastics, fluid flow, and electromagnetics, to name but a few domains. Numerical methods such as finite elements allow us to solve these differential equations, with good approximation, for many practical situations. We can use these methods to investigate and explain observations, as well as to predict the behavior of products and systems long before they are ever physically realized.

But progress in systematic *synthesis* has been much slower. For example, the systematic synthesis of a kinematic machine for a given purpose is a long-standing problem, and perhaps one of the earliest general synthesis problems to be posed. Robert Willis, a professor of natural and experimental philosophy at Cambridge, wrote in 1841 [32]:

> *[A rational approach to synthesis is needed] to obtain, by direct and certain methods, all the forms and arrangements that are applicable to the desired purpose. At present, questions of this kind can only be solved by that species of intuition that which long familiarity with the subject usually confers upon experienced persons, but which they are*

totally unable to communicate to others. When the mind of a mechanician is occupied with the contrivance of a machine, he must wait until, in the midst of his meditations, some happy combination presents itself to his mind which may answer his purpose."

Robert Willis, *Principles of Mechanism* [32]

Almost two centuries later, a rational method for the synthesis in many domains is still not clear. Though many best-practice design methodologies exist, at the end of the day they rely on elusive human creativity. Product design is still taught today largely through apprenticeship: Engineering students learn about existing solutions and techniques for well-defined, relatively simple problems, and then – through practice – are expected to improve and combine these to create larger, more complex systems. How is this synthesis process done? We do not know, but we cloak it with the term "creativity".

The question of how synthesis of complex systems occurs has been divided in a dichotomy of two views: One view is that complex systems emerge through successive adaptations coupled with natural selection. This Darwinian process is well accepted in Biology, but is more controversial in engineering [2,35]. The alternative explanation is intelligent design, mostly rejected by Biology, but still dominant in Engineering – as the celebrated revolutionary inventor.

The process of successive adaptation by improvement and recombination of basic building blocks is evolutionary in its nature. Unlike classical genetic algorithms [10], however, it is *open ended*: We do not know *a-priory* what components we will need and how many of them. The permutation space is exponential, and complexity is unbounded. This is perhaps a subtle but key difference between optimization [23], and synthesis. In optimization problems we tune the values of a set of parameters in order to maximize a target function. The set of parameters, their meaning and their ranges are predetermined. Synthesis, on the other had, is an open-ended process where we can add more and more components, possibly each with their own set of parameters. Consider for example a case where we need to design a new electronic circuit that performs some target function. One approach would be to manually provide a basic layout of resistors, capacitors, and coils, and then try to automatically tweak their values so as to maximize performance. Alternatively, we could start with a bucket of components, and use an algorithm to automatically compose them into a circuit that performs the target function. The former case would be optimization, and the later would be an example of synthesis.

Structure of this overview

As we shall see in the next few sections, we can use many of the ideas of biological evolutionary adaptation to inspire computational synthesis methods. To keep things intuitive, we shall describe some of these methods in the context of designing electromechanical machines, such as robots, and in particular legged robots. But these methods can be (and indeed have been) applied to numerous engineering application areas. This text is not intended to be a comprehensive review of evolutionary robotics or of evolutionary design research. Instead I have chosen a small set of results that portray an interesting perspective of the field and where it is going. These results are not necessarily in chronological order – scientific discoveries are not always made in an order most conducive for learning. Interested readers are encouraged see the "further reading" section for more in-depth and broader reviews.

A simple model of evolutionary adaptation

There are a variety of computational models of open-ended synthesis loosely inspired by natural evolutionary adaptation. Perhaps the simplest approach uses a direct representation. We start off with a large set of initial candidate designs – this is the initial *population*. These designs may be random, blank, or may be seeded with some prior knowledge in the form of solutions we think are good starting points. We then begin evolving this population through repeated *selection* and *variation*. To perform selection, we first measure the performance of each solution in the population. The performance, termed *fitness* in evolutionary terminology, captures the merit of the design with respect to some target performance we are seeking as the designers. The fitness metric needs to be solution-neutral, i.e measure *the extent* to which the target task has been achieved, regardless of *how* it was achieved. We select better solutions (*parents*) and use them to create a new generation of solutions (*offspring*). The offspring are variations of the parents, created through variation operators like mutation and recombination. The process is repeated generation after generation until good solutions are found.

In practice, there are many modifications to the simple process described above: We use special representations, clever selection methods and sophisticated variation and evaluation methods, as well as multiple co-evolving populations. Most interestingly, we let the representations and the evaluation methods evolve too, to allow for a more open-ended search. We will discuss these later; but for now lets look at some simple examples.

Machine bodies and brains

Many systems, including robotic systems in particular, are often viewed as comprising two major parts: The morphology, and the controller. The morphology is the physical structure of the system, and the controller is a separate unit that governs the behavior of the morphology by setting actuators states and reading sensory data. In nature, we often refer to these as the body and brain, respectively. In control theory we refer to these as the plant and the control (the term *plant,* as in “manufacturing plant”, is used because of the original industrial applications). In computer engineering terms this often translates into software and hardware. This distinction is semantic; we simply tend to refer to the part which is more easily adaptable as control, and the part that is fixed as the morphology. In practice both the morphology and control contribute to the overall behavior of the system, and the distinction between them is blurred. Very often a particular morphology accounts for some of the control, and the control is embedded in the morphology. Nevertheless, in describing the application of evolutionary design to systems we find this distinction pedagogically useful.

In the following sections we will see a series of examples of applications of evolutionary processes to open-ended synthesis. These examples were chosen to illustrate the design of robotic systems for their intuitiveness, starting at the control and moving on to both the control and morphology. Following these examples, we will take a look at the common principles, and future challenges.

Evolving controllers

It is perhaps easier, both conceptually and technically, to explore application of evolutionary techniques to the design of robot controllers before using it to evolve their morphologies too. Robot controllers can be represented in any one of a number of ways: As logic functions ("if-then-else" rules), as finite state machines, as programs, as sets of differential equations, or as neural networks, to name a few. Many of the experiments that follow represent the controller as a neural network that maps sensory input to actuator outputs. These networks can have many architectures, such as feed-forward or recurrent. Sometimes the choice of architecture is left to the synthesis algorithm.

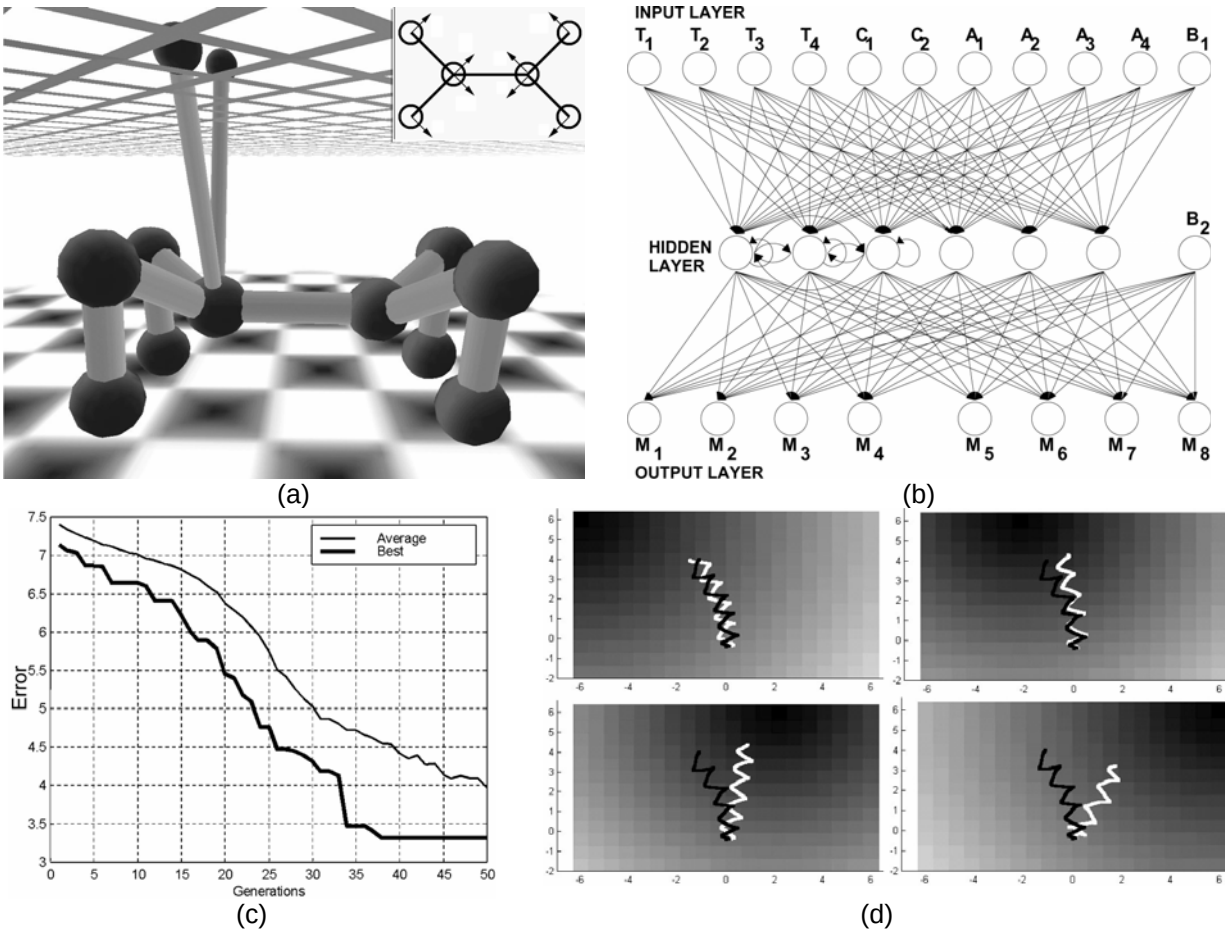

Figure 1: Evolving a controller for a fixed morphology: (a) The morphology of the machine contains four legs actuated with eight motors, four ground touch sensors, four angle sensors, and two chemical sensors. (b) The machine is controlled by a recurrent neural net whose inputs are connected to the sensors and whose outputs are connected to the motors; (c) Evolutionary progress shows how the target misalignment error reduces over generations, (d) White trails show the motion of the machine towards high concentration (darker area). Black trail show track when the chemical sensors are turned off. From [6].

Nolfi and Floreano [22] describe many interesting experiments evolving controllers for wheeled robots, but let us look at some examples with legged robots. Consider a case where we have a legged robot morphology fitted with actuators and sensors, and we would like to use evolutionary methods to evolve a controller that would make this

machine move (locomote) towards an area of high chemical concentration. Bongard [6] explored this concept on a legged robot in a physically-realistic simulator. The robot has four legs and eight rotary actuators as shown in Figure 1a. It has four touch sensors at the feet, which output a binary signal depending on weather or not they are touching the ground. The machine also has four angle sensors at the knees, outputting a graded signal depending on the actual angle of the knee. There are two chemical sensors at the top, which output a value corresponding to the chemical level they sense locally.

The behavior of the machine is determined by a neural controller that maps sensors to actuators, as shown in Figure 1b. Inputs of candidate neural controllers were connected to the sensors, and their output connected directly to the eight motors. Machines were rewarded for their ability to reach the area with high concentration. The fitness was evaluated by trying out a candidate controller in four different concentration fields, and summing up the distance between the final position of the robot and the highest concentration point. The shorter the distance the better – and in this sense the total distance is a performance error. In this experiment, 200 candidate controllers were evolved for 50 generations. The variation operators could decide if and how to connect the neurons. Figure 1c shows the progress of this error over generational time. The performance of one successful controller in four different chemical concentration fields is shown in Figure 1d. The white trails, which mark the progress of the center of mass of the robot over time, show clearly how the robot moves towards high concentration.

But what is more striking about this experiment is that the robot learned to perform essentially two tasks: To locomote, and to change orientation towards the high concentration. When the chemical sensors are disabled, the robot moves forward but not towards the chemical concentration (see black trail in Figure 1d). This shows that the network evolved two *independent* functions: Locomotion and gradient tracking.

Can this process also work for a real (not simulated) legged robot? We recently tried evolving controllers for a dynamical, legged robot [29]. The nine-legged machine composed of two Stewart platforms back to back. The platforms are powered by twelve pneumatic linear actuators, with power coming from an onboard 4500psi paintball canister. While most robotic systems are use position-controlled actuators whose exact extension can be set, pneumatic actuators of the kind used here are force-controlled. Like biological muscle, the controller can specify the force and duration of the actuation, but not the position. It is therefore a challenging control problem. The controller architecture for this machine was an open-loop pattern generator that determines when to open and close pneumatic valves. The on-off pattern was evolved; Candidate controllers were evaluated by trying them out on the robot in a cage, and measuring fitness using a camera that tracks the red ball on the foot of one of the legs of the machine (see inset in Figure 2b for a view from the camera). Snapshots from one of the best evolved gates are shown in Figure 2c.

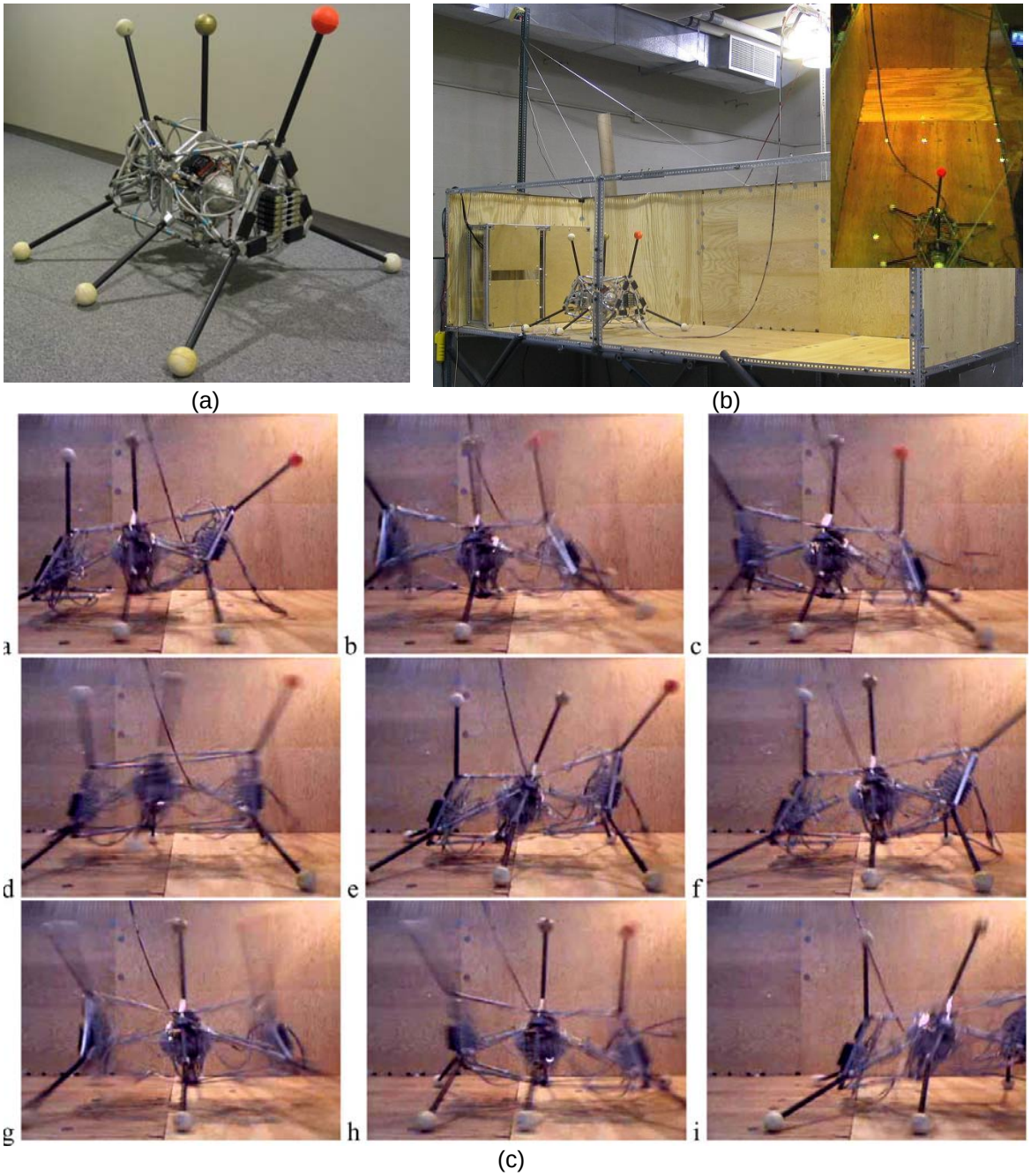

Figure 2: Evolving a controller for physical dynamic legged machine: (a) The nine-legged machine is powered by twelve pneumatic linear actuators arranged in two Stewart platforms. The controller for this machine is an open-loop pattern generator that determines when to open and close pneumatic valves. (b) Candidate controllers are evaluated by trying them out on the robot in a cage, and measuring fitness using a camera that tracks the red foot (see inset). (c) Snapshots from one of the best evolved gates. From [36].

Manual design of a neural controller for a legged machine of this sort is possible, but not easy. The advantage of design automation here is that a design was found without supplying prior information on how it should be done. We could now reverse engineer the evolved controller to find out exactly how it works – like biologists. Should the morphology or the task change, we can have the process redesign new controllers.

Evolving controllers and some aspects of the morphology

Designing a controller for a dynamic bipedal robot it difficult – to the point where we have not seen to date any physical successful implementations. Paul and Bongard [25] used evolutionary adaptation to design controllers for such a machine in simulation, shown in Figure 3a. The machine comprises the bottom half of a walker with six motors (two at each hip and one in each knee), a touch sensor at each foot and an angle sensor at each joint. The fitness of a controller was the net distance it could make a machine travel. The controllers had an architecture similar to that shown in figure 1b, with the appropriate number of inputs and outputs.

Evolving 300 controllers over 300 generations generated various controllers that could make the machine move while keeping it upright. Figure 3b shows the maximum fitness per generation for a number of independent runs. While many did not make much progress, some runs were able to find good controllers, as evident by the curves with high fitness. More importantly, however, was that this time evolutionary process was also allowed to vary the mass distribution of the robot morphology and that this new freedom allowed it to find good solutions. This may suggest that evolving a controller for a fixed morphology may be too restrictive, and that better machines might be found if both the controller and the morphology are allowed to co-evolve, as they do in nature. This lends some credibility to the notion of concurrent engineering, where several aspects of a product are engineered in concert, rather than sequentially. Some small changes to the morphology may make the controller design task much simpler.

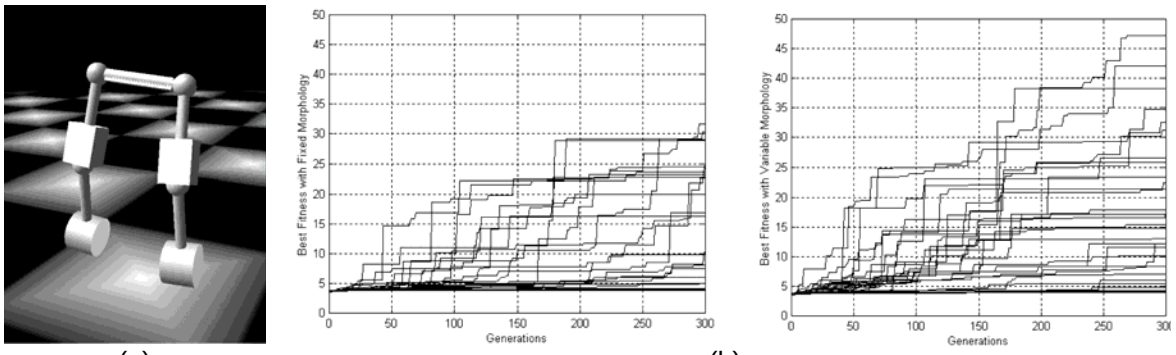

Figure 3: Evolving a controller and some morphology parameters for bipedal locomotion: The morphology of the machine consists of six motors (4 at the hip, two at the knees), six angle sensors and two touch sensors. The controller is a recurrent network similar to **Figure 1**b. (a) One of the evolved machines, (b) A comparison of fitness over generations for the fixed morphology (left) and a variable morphology (right). From [25].

Evolving bodies and brains

One may wonder what happens if the evolutionary process is given even more freedom in the design of both the morphology and control. Karl Sims [28] explored this idea in simulation using 3D cubes and oscillators as building blocks. Inspired by that work, we were interested in exploring physically-realizable machines and start with lower-level building blocks, such as simple neurons and 1D elements [17]. We used a design space consisting of bars and linear actuators for the morphology and neurons for the control (Figure 4a). The design space we used was comprised of bars and actuators as building

blocks of structure and artificial neurons as building blocks of control. Bars connected with free joints can potentially form trusses that represent arbitrary rigid, flexible and articulated structures as well as multiple detached structures, and emulate revolute, linear and planar joints at various levels of hierarchy. Similarly, sigmoidal neurons can connect to create arbitrary control architectures such as feed-forward and recurrent nets, state machines and multiple independent controllers. The bars connect to each other through ball-and-socket joints, neurons can connect to other neurons through synaptic connections, and neurons can connect to bars. In the latter case, the length of the bar is governed by the output of the neuron, by means of a linear actuator. No sensors were used. Variation operators used in the evolutionary process were allowed to connect, disconnect, add, remove or modify any of the components.

Starting with a population of 200 blank machines that were comprised initially of zero bars and zero neurons, we conducted evolution in simulation. The fitness of a machine was determined by its locomotion ability: The net distance its center of mass moved on an infinite plane in a fixed duration. The process iteratively selected fitter machines, created offspring by adding, modifying and removing building blocks, and replaced them into the population. This process typically continued for 300 to 600 generations. Both body (morphology) and brain (control) were thus co-evolved simultaneously. The simulator we used for evaluating fitness supported quasi-static motion in which each frame is statically stable. This kind of motion is simpler to transfer reliably into reality, yet is rich enough to support low-momentum locomotion.

Typically, several tens of generations passed before the first movement occurred. For example, at a minimum, a neural network generating varying output must assemble and connect to an actuator for any motion at all (see sequence in Figure 4a for an example). A sample instance of an entire generation, thinned down to unique individuals is shown in Figure 4b. Various patterns of evolutionary dynamics emerged, some of which are reminiscent of natural phylogenic trees. Figure 4c presents examples of extreme cases of convergence, speciation, and massive extinction, and Figure 4d shows progress over time of one evolutionary run. Figure 4e shows some of the fitter machines that emerged from this process; these machines were “copied” from simulation into reality using rapid-prototyping technology (Figure 4f). The machines performed in reality, showing the first instance of a *physical* robot whose entire design – both morphology and control – were evolved.

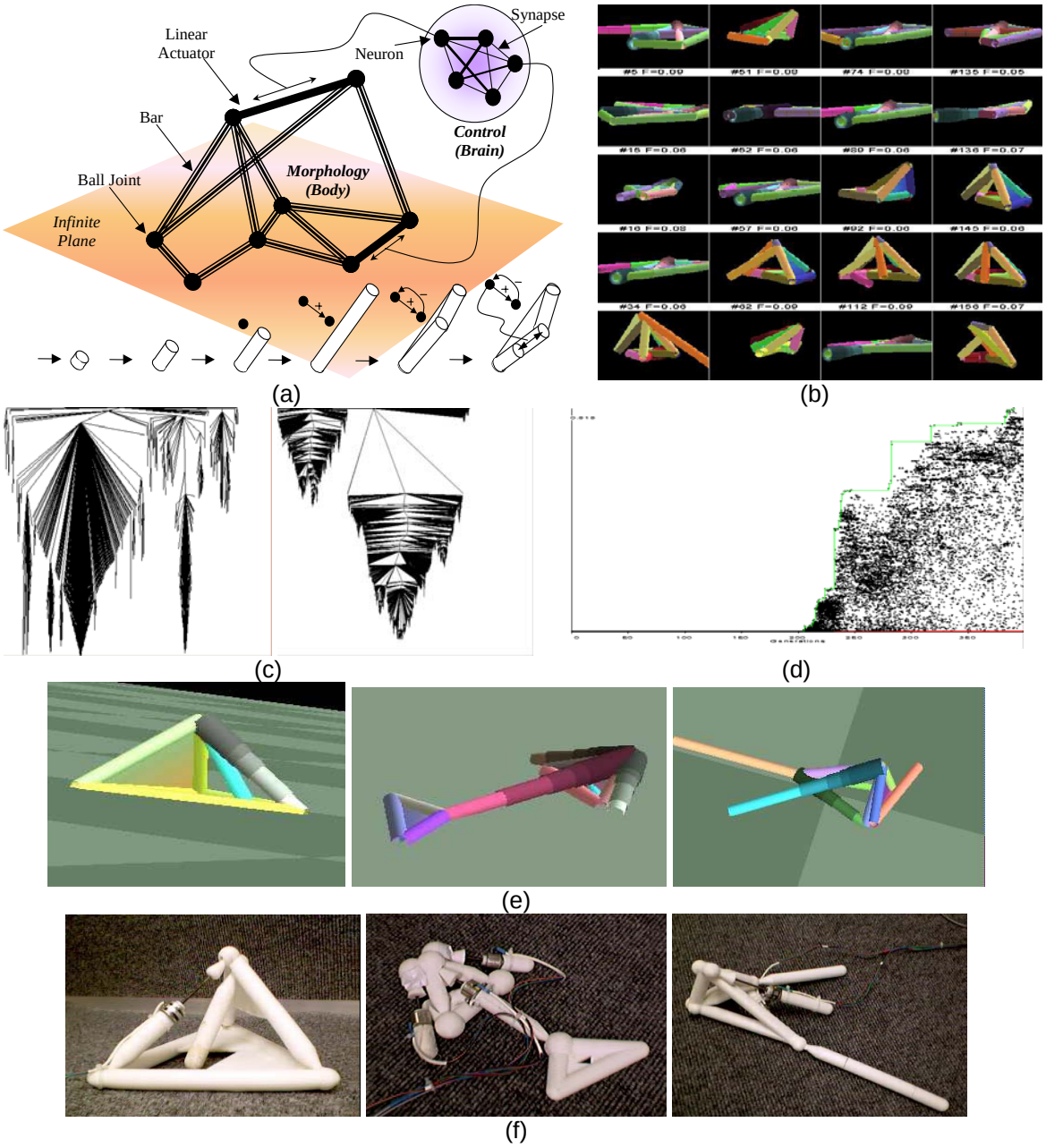

Figure 4: Evolving bodies and brains: (a) Schematic illustration of an evolvable robot, (b) An arbitrarily sampled instance of an entire generation, thinned down to show only significantly different individuals, (c) Phylogenetic trees of two different evolutionary runs, showing instances of speciation and massive extinctions, (d) Progress of fitness versus generation for one of the runs. Each dot represents a robot (morphology and control). (e) Three evolved robots, in simulation (f) the three robots from Fig c. reproduced in physical reality using rapid prototyping. From [17].

In spite of the relatively simple task and environment (locomotion over an infinite horizontal plane), surprisingly different and elaborate solutions were evolved. Machines typically contained around 20 building blocks, sometimes with significant redundancy (perhaps to make mutation less likely to be catastrophic). Not less surprising was the fact that some exhibited symmetry, which was neither specified nor rewarded for anywhere in the code; a possible explanation is that symmetric machines are more likely to move in a

straight line, consequently covering a greater net distance and acquiring more fitness. Similarly, successful designs appear to be robust in the sense that changes to bar lengths would not significantly hamper their mobility. The three samples shown in Figure 4d exploit principles of ratcheting, anti-phase synchronization and dragging. Others (not shown here) used a sort of a crawling bi-pedalism, where a body resting on the floor is advanced using alternating thrusts of left and right "limbs". Some mechanisms use sliding articulated components to produce crab-like sideways motion. Other machines used a balancing mechanism to shift friction point from side to side and advance by oscillatory motion.

Morphology representations

The examples above used mostly a direct encoding – a representation of the morphology and control that evolution to explicitly modify each aspect of the design, adding, removing and modifying components and parameters. Clearly, however, such an approach would not work in nature, for an average animal body contains billions of cells. Nature typically uses a more compact representation – a *genotype* – to encode for a much more complex machine – the *phenotype*. The genotype does not directly encode the phenotype, but instead it encodes information for growing, or *developing*, a phenotype. This is one form of an *indirect* representation that maps between a genotype and a phenotype. In nature, these maps are evolving themselves, and several hierarchical layers of mappings are used before a real DNA yields a working phenotype.

The use of a genotype-phenotype mapping allows for may advantages, primarily the compactness of a description and the ability to reuse components (more on that later). But how can we use these representations computationally?

Mechanisms and neural networks can both be described as graphs. Luke and Spector [18] survey a number of different representations used to describe or 'grow' graphs, such as neural networks. Some methods use context free grammars, L-systems, and parse trees operating on nodes and edges. Most of the existing representations for encoding networks generate highly connected architectures that are suitable for computational networks, but which are less suitable for kinematic machines because they over-constrain the motion and create deadlocked mechanisms. Using these representations, the likelihood of generating a mechanism with a specific number of degrees of freedom is vanishingly small. In order to allow an evolutionary algorithm to explore the space of one degree-of-freedom (DoF) mechanisms more efficiently, a more suitable representation is required.

A second consideration in the choice of representation is that of evolvability. Many of the representations cited above result in context-sensitive and order-sensitive description of a network. For example, the structure generated by a branch in Gruau's cellular encoding depends on whether it is parsed before or after its sibling branch. If that branch is transplanted by crossover into another tree it may produce an entirely different structure. Such behavior hampers the effectiveness of recombinative operators by precluding the formation of modular components that are discovered by the search in one place and then reused elsewhere. A representation where the structure produced by a branch of the tree is minimally affected by its context may thus be more evolvable.

Tree representations

Tree-based representations can describe a set of operations to construct a phenotype in a top-down or bottom-up manner. A top-down representation starts with an initial structure (an embryo) and specifies a sequence of operations that progressively modify it into its final form. Figure 5a shows a top-down tree that specifies the construction of an electric circuit, starting with an initial circuit and recursively replacing circuit segments with serial and parallel arrangements of electrical components [14]. Each node of the tree is either an operator that modified the circuit and passes segments to its child nodes, or a terminal electrical component. The specific parallel and serial operators cannot be used for construction of mechanisms as they will immediately create over- and under-constrained kinematic chains. Because of the physics of electric circuits, ordering of children under a parent does not matter. This tree is thus both order independent and context independent. In a top-down tree, parent nodes must be constructed before their children. Figure 5a also shows a bottom-up construction of a symbolic expression. Here terminal nodes represent constants or variables, and parent nodes represent mathematical op-erators. Because of the nature of mathematical expressions, parsing order is important, and swapping order of some child nodes would result in a mathematically different expression. The terms are unchanged, however, by the content of their siblings. This tree is thus order dependent but context independent. In a bottom-up tree, child nodes must be constructed before their parents.

How could a tree-representation be used to describe robot morphologies? Top-down construction of a mechanism starts with an embryonic kinematic basis with the desired number of degrees of freedom (DoF's), such as the four-bar mechanism shown in Figure 5c. A tree of operators then recursively modifies that mechanism by replacing single links (DoF = -1) with assemblies of links with an equivalent DoF, so that the total number of DoF remains unchanged. Two such transformations are shown in Figure 5c: The *D* and *T* operators. The *D* operator creates a new node and connects it to both the endpoints of a given link, essentially creating a rigid triangular component. The *T* operator replaces a given link with two links that pass through a newly created node. The new node is also connected to some other existing node. In both operators, the position of the new nodes is specified in coordinates local to link being modified. The T operator specified the external connecting node by providing coordinates relative to link being modified; the closest available node from the parent structure is used. This form of specification helps assure the operators remain as context and order independent as possible. Figure 5c shows how a certain sequence of operators will transform a dyad into a tryad. Figure 5c also shows how application of a tree of operators to the embryonic mechanism, will transform it into an arbitrary compound mechanism with exactly one DoF. Terminals of the tree are the actual links of the mechanism.

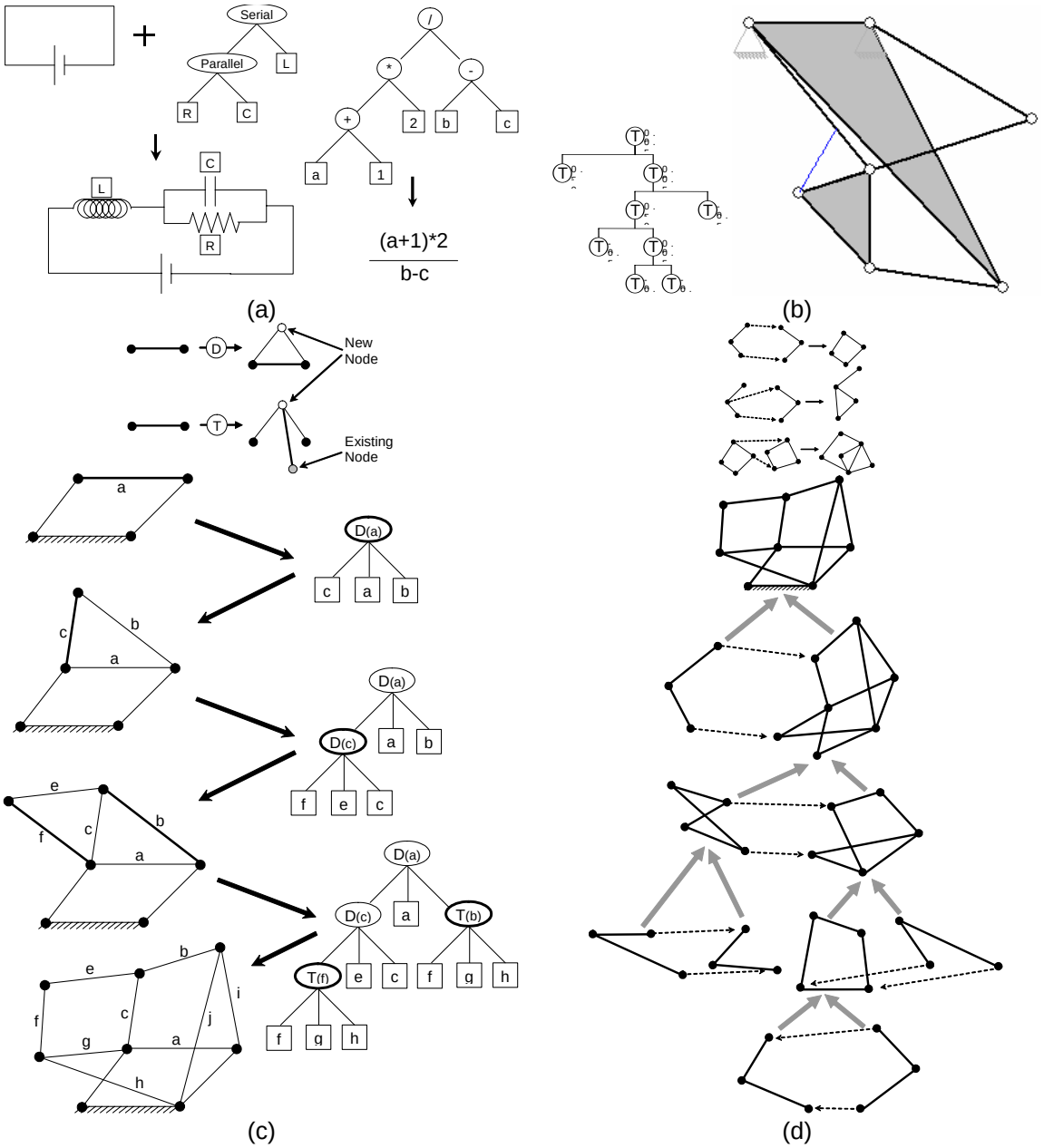

Figure 5: A language to represent kinematic machines (a) Top down and bottom up trees used to represent structure. (b) A tree used to represent a kinematic machine; This machines traces a nearly-exact straight line. These mechanisms can be represented as top-down trees (c) or as bottom up trees (d). From [15].

Alternatively, bottom-up construction of a one-DoF mechanism begins at the leaves of the tree with atomic building blocks and hierarchically assembles them into components. The atomic building block is a dyad as shown in Figure 4a, and has exactly one DoF when grounded. The composition operator ensures that the total number of DoF is not changed when two subcomponents are combined, and thus the total product of the tree will also be a mechanism with exactly one DoF. When combining two components each of one DoF, the resulting assembly will have five DoF (one DoF from each, plus three DoF released by ungrounding one of the components). The total DoF is restored to

one by eliminating four DoF through the merging of two point pairs. An example of this process is shown in Figure 5d. Note that points must be merged in a way that avoids overlapping constraints, such as causing two links to merge. The components may need to be scaled and oriented for the merger to work. The ground link of the entire structure is specified at the root of the tree.

Figure 5b shows an application of this representation to the design of a single DoF mechanism that when actuated traces a nearly exact straight line, without reference to an existing straight line. This problem may seem somewhat arbitrary, but it was of major practical importance in the 19th century and many notable inventors including James Watt, spent a considerable amount of time developing mechanisms to meet this requirement as the bootstrap of precision manufacturing. It therefore serves as a nice benchmark for the "invetiveness" of the algorithm. Using evolutionary computation based on tree representations, we were able to evolve machines, from scratch, that infringe and outperform previous established designs [15].

Developmental representations

Other types of representations allow the robot's morphology to develop from a basic "seed" and a set of context-free development rules. Consider, for example, the two rules "A→B" and "B→AB". If we start with the seed "A", and apply these two rules wherever they are applicable, the seed will develop as follows: A→B→AB→BAB→ABBAB→ BABABBAB… and so forth. A seed and two simple rules can thus create very complex and elaborate structures. This type of representation, similar to an L-system or cellular automaton, can be applied to evolving morphologies and controllers of robots.

We start with a constructor can build a machine from a sequence of build commands. The language of build commands is based on instructions to a LOGO-style turtle, which direct it to move forward, backward or rotate about a coordinate axis. Robots are constructed from rods and joints that are placed along the turtle's path (Figure 6a). Actuated joints are created by commands that direct the turtle to move forward and place an actuated joint at its new location, with oscillatory motion and a given offset. The operators "[" and "]" push and pop the current state – consisting of the current rod, current orientation, and current joint oscillation offset – to and from a stack. Forward moves the turtle forward in the current direction, creating a rod if none exists or traversing to the end of the existing rod. Backward goes back up the parent of the current rod. The rotation commands turn the turtle about the Z-axis in steps of 60°, for 2D robots, and about the X, Y or Z axes, in steps of 90°, for 3D robot. Joint commands move the turtle forward, creating a rod, and end with an actuated joint. The parameter to these commands specify the speed at which the joint oscillates, using integer values from 0 to 5, and the relative phase-offset of the oscillation cycle is taken from the turtle's state. The commands "Increase-offset" and "decrease-offset" change the offset value n 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.

For example, the string *{ joint(1) [joint(1) forward(1)] clockwise(2)}(3)* produces the robot in Figure 6b, through the development process shown in Figure 6a. Constructed robots do not have a central controller; rather each joint oscillates independent of the others. In Figure 6 large crosses are used to show the location of

actuated joints and small crosses show unactuated joints. The left image shows the robot with all actuated joints in their starting orientation and the image on the right shows the same robot with all actuated joints at the other extreme of their actuation cycle. In this example all actuated joints are moving in phase.

These strings were generated using an L-system. The L-systems is a set of rules like the "A→B" and "B→AB" rules discussed above. However, this time these "rewrite" rules are parametric (i.e. may pass parameters), and have conditions (are executed only when the parameters meet some conditions).

For example, the L-system to produce the robot in Figure 6b consists of two rules with each rule containing two condition-successor pairs:

P0(n): $n > 2$ → {P0($n - 1$) }(n)

$n > 0$ → *joint*(1) P1($n \times 2$) *clockwise*(2)

P1(n): $n > 2$ → [P1($n / 4$)]

$n > 0$ → *joint*(1) *forward*(1)

If the L-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)

Which produces the robot in Figure 6b.

An evolutionary algorithm was used to evolve individual L-systems, that when executed produced a build sequence which produced the machine. Approximately half the runs produce "interesting" viable results. The two main forms of locomotion found used one or more oscillating appendages to push along, or had two main body parts connected by a sequence of rods that twisted in such a way that first one half of the robot would rotate forward, then the other. Some examples of successful machines are shown in Figure 6c and their physical instantiations are shown in Figure 6d.

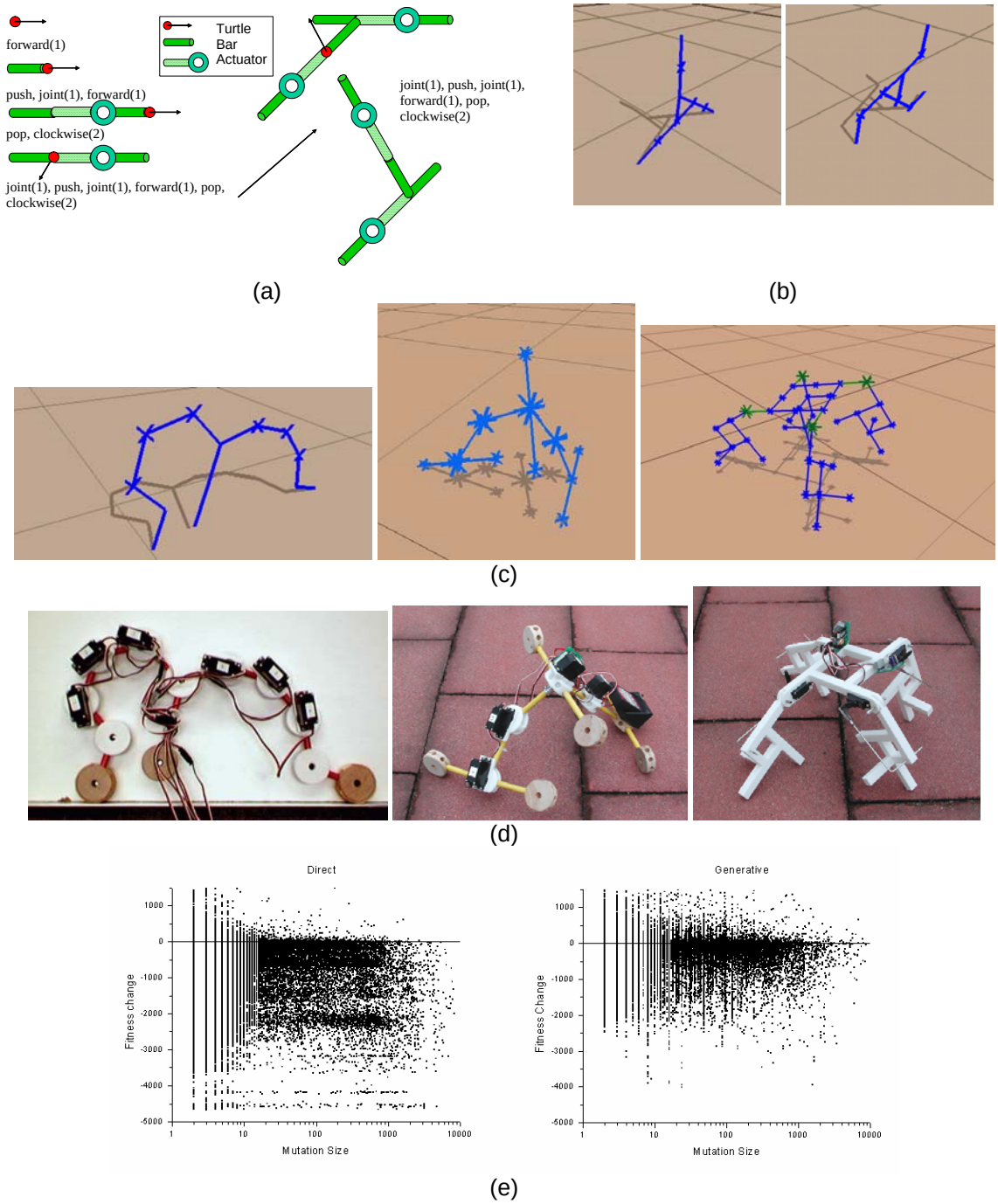

Figure 6: Evolving bodies and brains using generative encodings: (a) Schematic illustration of an construction sequence and (b) the resulting robot with actuated joints. (c) Three examples of robots produced by evolving L-systems that produce construction sequences, and (d) their physical instantiations. (e) A comparison of effects of mutation in the direct encoding versus the generative encoding, shows that the generative encoding has transformed the space in a way that makes mutation more effective. From [12].

A comparison of robots evolved using the developmental encoding to robots whose construction sequence was evolved directly revealed that robots evolved with the generative representation not only had higher average fitness, but tended to move in a more continuous manner. In general, robots evolved using the generative representation

increased their speed by repeating rolling segments to smoothen out their gaits, and increasing the size of these segments/appendages to increase the distance moved in each oscillation.

One of the fundamental questions is whether the actual grammar evolved in the successful L-systems has captured some of the intrinsic properties of the design space. A way to quantify this is to measure the correlation between fitness change and a random mutation of various sizes, and compare this with the correlation observed in random mutations on the non-generative representation as a control experiment. If the observed correlation is distinguishable and better for the generative system than it is for the blind system, then the generative system must have captured some useful properties.

The plot in Figure 6e is a comparison of the fitness-mutation correlation between a generative representation and a random control experiment on the same substrate and on the same set of randomly selected individuals. For this analysis, 80,000 individuals were selected uniformly from 16 runs and over 100 generations using a generative representation. Each point represents a particular fitness change (positive or negative) associated with a particular mutation size. The points on the left plot of Figure 6e were carried out on the non-generative representation generated by the generative representation and serve as the control set. For these points, 1 to 6 mutations were applied so as to approximate mutations of similar phenotypic-size as those on the generative representation. Each mutation could modify or swap a sequence of characters. The points on the right of Figure 6e were also carried out randomly but on the generative representations of the same randomly selected individuals. Only a single mutation was applied to the generative representation, and consisted of modifying or swapping a single keyword or parameter. Mutation size was measured in both cases as the number of modified commands in the final construction sequences.

The two distributions in Figure 6e have distinct features. The data points separate into two distinguishable clusters, with some overlap. Mutations generated on the generative representations clearly correlate with both positive fitness and negative fitness changes, whereas most mutations on the non-generative representation result in fitness decrease. Statistics of both systems, averaged over 8 runs each, reveal that the two means are different with at least 95% confidence. Cross-correlation showed that in 40% of the instances where a non-generative mutation was successful, a generative mutation was also successful, whereas in only 20% of the instances where a generative mutation was successful, was a non-generative mutation successful too. In both cases smaller mutations are significantly more successful than larger mutations. However large mutations (>100) were an order of magnitude more often successful in the generative case than in the non-generative case. All these measures indicate that the generative representation is more efficient in exploiting useful search paths in the design space.

Regulatory network representations

The way that morphologies of organisms develop in biology is not only dependent on their genotype; many other environmental effects play an important role. The ontology of an organism depends on chains of productions that trigger other genes in a complex regulatory network. Some of these triggers are intracellular, such as one set of gene products resulting in expression of another group of genes, while other product may

inhibit certain expressions creating feedback loops and several tiers of regulation. Some signaling pathways transduce extracellular signals that allow the morphology to develop in response to particular properties of its extracellular environment. This is in contrast to the representations discussed earlier, where the phenotype was completely defined by the genotype. Through these regulatory pathways, a genotype may encode a phenotype with variations that can compensate, exploit and be more adaptive to its target environment.

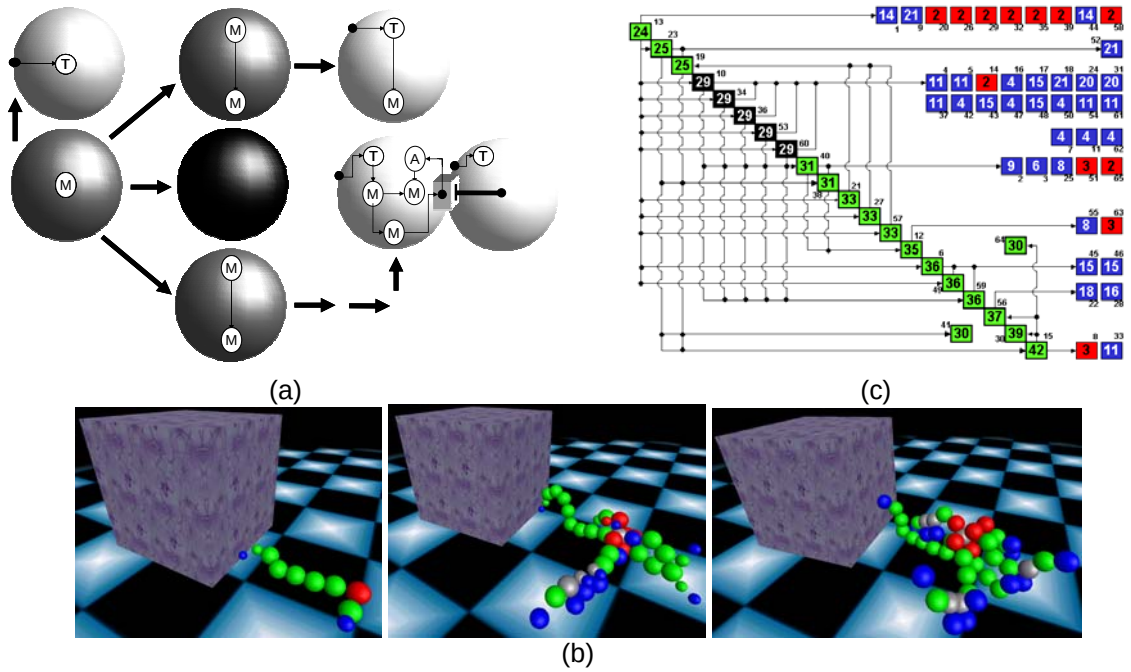

Figure 7: Artificial ontogeny: Growing machines using gene regulatory networks (a) An example of cells that can differentiate into structural, passive cells (dark) or active cells (bright) which contains neurons responsible for sensing (T=touch, A=angle) and motor actuation (M). The connectivity of the neurons is determined by propagation of 'chemicals' expressed by genes and sensors, who are themselves expressed in response to chemicals in a regulatory network. (b) Three machines evolved to be able to push a block, (c) The distribution of genes responsible for neurogenesis (red) and morphogenesis (blue) shows a clear separation that suggests an emergence of a 'body' and a 'brain'. From [8].

Bongard and Pfeifer [8] explored a regulatory network representation for evolving both a body and a brain of a robot. The machines were composed of spherical cells, which could each contain several angular actuators, touch sensors, and angular sensor, as seen in Figure 7a. The actuators and sensors were connected through a neural network as in Figure 1b, but the specific connectivity of the network was determined by an evolved regulatory network. The regulatory network contained genes which could sprout new connections and create new spherical cells, as well as express or inhibit 'chemical' signals that would propagate through the structure. These 'chemical' signals could also trigger the expression of other genes, giving rise to complex signaling and feedback pathways. Some machines evolved in response to a fitness rewarding the ability to push a block forward, are shown in Figure 7b. These machines grow until they reach the block and have a firm grasp of the ground; their regulatory nature would allow them to attain a slightly different morphology if they would be growing in presence of a slightly differently shaped block. It is interesting to note that an analysis of the regulation pattern

(who regulates who, Figure 7c) shows that genes that regulate growth of neurons (colored red) and genes the regulate growth if new cells (colored blue) are relatively separated, suggesting an initial emergence of what we call “body’ and ‘brain’.

Evolving machines in physical reality

Though many robotic experiments are carried out in simulation, a robot must ultimately reside in physical reality. Applying evolutionary processes to physical machines is difficult for two reasons. First, even if we are only evolving controllers for a fixed machine, each evaluation of a candidate controller involves trying it out in reality. This is a slow and costly process that also wears out the target system. Performing thousand of evaluations is usually impractical. Second, if we are evolving morphology as well, then how would these morphological changes take place in reality? Changes to the controller can be done simply by reprogramming, but changes to the morphology require more sophisticated processes. Nature has some interesting solutions to this problem, such as growing materials, or self-assembling and self-replicating basic building blocks like cells. Let’s examine these two approaches.

Evolving controllers for physical morphologies

One approach to evolving controllers for fixed morphologies is to make a simulator that is so perfect, that whatever works in simulation will work in reality equally well. Unfortunately, such a simulator does not yet exist. It is unlikely that one would exist, given that machine dynamics are inherently chaotic and sensitive to initial conditions and many small parameter variations. But even if such simulators existed, creating accurate models would be painstakingly difficult, or may be impossible because the target environment is unknown.

An alternative approach to “crossing the reality gap” is to use a crude simulator that captures the salient features of the search space. Techniques have been developed for creating such simulators and using noise to cover uncertainties so that the evolved controllers do not exploit these uncertainties [13]. Yet another approach is to use plasticity in the controller: Allow the robot to learn and adapt in reality. In nature, animals are born with mostly predetermined bodies and brains, but these have some ability to learn and make final adaptations to whatever actual conditions may arise.

A third approach is to co-evolve simulators so that they are increasingly predictive. Just as we use evolution to design a controller, we can use evolution to design the simulator so that it captures the important properties of the target environment. Assume we have a rough simulator of the target morphology, and we use it to evolve controllers in simulation. We then take the best controller and try it – once – on the target system. If successful, we are done; but if the controller did not produce the anticipated result (as is likely to happen since the initial simulator was crude), then we observed some unexpected sensory data. We then evolve a new set of simulators, whose fitness is their ability to reproduce the actual observed behavior when the original controller is tested on them. Simulators that correctly reproduce the observed data are more likely to be predictive in the future. We then take the best simulator, and use to evolve a new controller, and the cycle repeats: If the controller works in reality, we are done. If it does not work as expected, we now have more data to evolve better simulators, and so forth.

The co-evolution of controllers and simulators is not necessarily computationally efficient, but it dramatically reduces the number of trials necessary on the target system.

The co-evolutionary process consists of two phases: Evolving the controller (or whatever we are trying to modify on the target system) – we call this the exploration phase. The second phase tries to create a simulator, or model of the system – we call this the estimation phase. To illustrate the estimation-exploration process, consider a target robot with some unknown, but critical, morphological parameters, such as mass distribution and sensory lag times. Fifty independent runs of the algorithm were conducted against the target robot. Figure 8a shows the 50 series of 20 best simulator modifications output after each pass through the estimation phase. Figure 8a makes clear that for all 50 runs, the algorithm was better able to infer the time lags of the eight sensors than the mass increases of the nine body parts. This is not surprising in that the sensors themselves provide feedback about the robot. In other words, the algorithm automatically, and after only a few target trials, deduces the correct time lags of the target robot's sensors, but is less successful at indirectly inferring the masses of the body parts using the sensor data. Convergence toward the correct mass distribution can also be observed. But even with an approximate description of the robot's mass distribution, the simulator is improved enough to allow smooth transfer of controllers from simulation to the target robot. Using the default, approximate simulation, there is a complete failure of transferal: the target robot simply moves randomly, and achieves no appreciable forward locomotion. It is interesting to note that the evolved simulators are not perfect; they capture well only those aspects of the world that are important for accomplishing the task.

The exploration-estimation approach can be used for much more than transferring controllers to robots – it could be used by the robot itself to estimate its own structure. This would be particularly useful if the robot may undergo some damage that changes some of its morphology in unexpected ways, or some aspect in its environment changes. As each controller action is taken, the actual sensory data is compared to that predicted by the simulator, and new internal simulators are evolved to be more predictive. These new simulators are then used to try out new, adapted controllers for the new and unexpected circumstances. Figure 8b shows some results applying this process to design controllers for a robot which undergoes various types of drastic morphological damage, like loosing a leg, motor, or sensor, or combinations of these. In most cases, the estimation-exploration process is able to reconstruct a new simulator that captures the actual damage using only 4-5 trials on the target robot, and then use the adapted simulator to evolve compensatory controllers that recover most of the original functionality. There are numerous applications to this identification and control process in other fields.

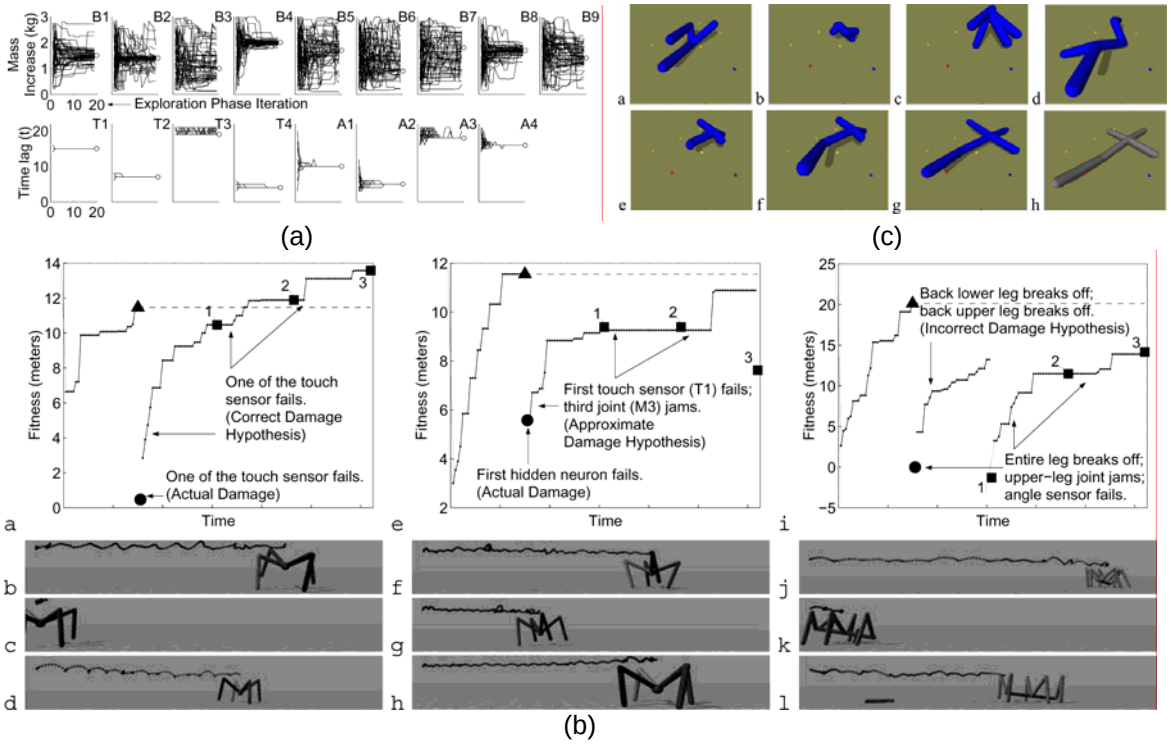

Figure 8: Co-evolving robots and simulators: (a) Convergence toward the physical characteristics of the target robot. Each pass through the estimation phase produces a set of mass changes for each of the nine body parts of the robot (top row) and a set of time lags for each of the eight sensors (bottom row). The open circles indicate the actual differences between the target robot and the starting default simulated robot [4]. (b) Three typical damage recoveries. a: The evolutionary progress of the four sequential runs of the exploration EA on the quadrupedal robot, when it undergoes a failure of one of its touch sensors. The hypotheses generated by the three runs of the estimation EA (all of which are correct) are shown. The dots indicate the fitness of the best controller from each generation of the exploration EA. The triangle shows the fitness of the first evolved controller on the target robot (the behavior of the 'physical' robot with this controller is shown in b); the filled circle shows the fitness of the robot after the damage occurs (the behavior is shown in c); the squares indicate the fitness of the 'physical' robot for each of the three subsequent hardware trials (the behavior of the 'physical' robot during the third trial is shown in d). e-h The recovery of the quadrupedal robot when it experiences unanticipated damage. i-l The recovery of the hexapedal robot when it experiences severe, compound damage. The trajectories in b-d, f-h and j-l show the change in the robot's center of mass over time (the trajectories are displaced upwards for clarity) [5]. (c) The simulator progressively learns the entire robot morphology from scratch. Panels (a-g) are progressive intermediate self-inference stages, panel (h) is the true target system [6].

Making morphological changes in hardware

An evolutionary process may require a change of morphology, or production of a new physical morphology altogether. One approach for generating new morphology is to use reconfigurable robots [34]. Reconfigurable robots are composed of many modules that can be connected, disconnected and rearranged in various topologies to create machines with variable body plans. Self-reconfigurable robots are able to rearrange their own morphology, and thus adapt in physical reality. Figure 9a shows one example of a self-reconfiguring robot composed of eight identical cubes [21]. Each cube can swivel around its (1,1,1) axis, and connect and disconnect to other cubes using electromagnets on its faces. Though this robot contains only 8 units, it is conceivable that future machine will

be composed of hundreds and thousands of modules of smaller modules, allowing much greater control and flexibility in morphological change.

An alternative approach to varying morphology is to produce the entire robot morphology automatically. For example, the robots shown in Figure 4f were produced using rapid prototyping equipment: These are 3D printers, that deposit material layer by layer to gradually build up a solid object of arbitrary geometry, as shown in Figure 9b. This "printer", when coupled to an evolutionary design process, can produce complex geometries that are difficult to produce any other way, and thus allow the evolutionary search much greater design flexibility. But even when using such automated fabrication equipment we needed to manually insert the wires, logic, batteries and actuators. What if the printer could print these components too? Future rapid prototyping systems may allow deposition of multiple integrated materials, such as elastomers, conductive wires, batteries and actuators, offering evolution an even larger design space of integrated structures, actuators and sensors, not unlike biological tissue. Figure 9c shows some of these printed components [19].

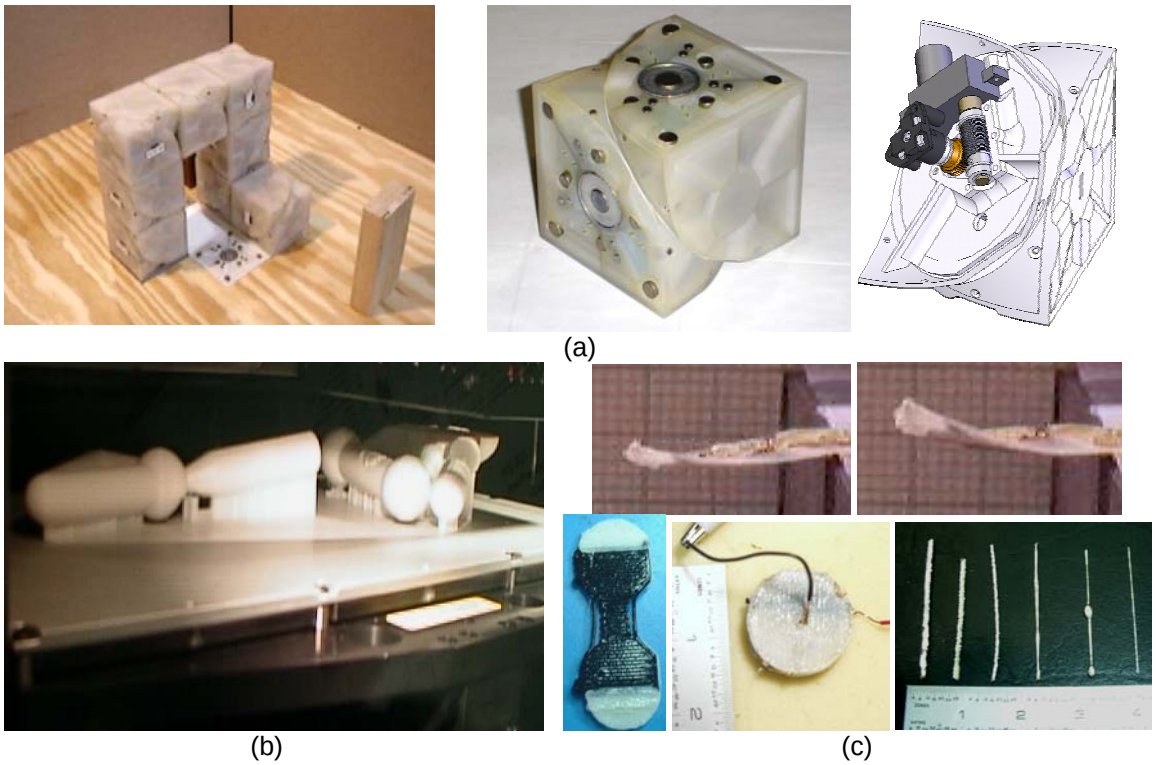

Figure 9: Transferring morphological changes into reality (a) Reconfigurable *molecube* robots [21], (b) Rapid prototyping, (c) Future rapid prototyping systems will allow deposition of multiple integrated materials, such as elastomers, conductive wires, batteries and actuators, offering evolution a larger design space of integrated structures, actuators and sensors, not unlike biological tissue. From [19].

The economy of design automation

The examples shown so far are all related to design and fabrication of robotic systems, but the principles described here are applicable in many other domains. Is there a way to know *a priori* where these methods will be successful? Several decades of experience have shown that there are a number of conditions that suggest such problem domains.

- Known physics. Most evolutionary systems use some form of simulation to determine the consequence of various design choices. Evolutionary algorithms are fruitful when the physics are understood well enough that simulations are predictive, there is no question about the underlying physical phenomena, and that simulation can be carried out in reasonable time.
- Well-defined search space. The basic 'atomic' building blocks comprising potential solutions are known, and it is clear how they are allowed to fit together. These two aspects define a search space in which the evolutionary algorithm can operate. Knowing the building blocks and interfaces does not imply knowing the solution. It is important to realize that 'building blocks' are not necessarily discrete components – they can be features of a solution, or partial solutions.
- Little formal design knowledge. Evolutionary algorithms are 'knowledge sparse'; they essentially generate knowledge through search. They are thus able to work in the absence of formal knowledge in the problem domain. Given enough time and resources, one may be able to design a specialized algorithm that takes advantage of specific domain knowledge and outperforms an evolutionary algorithm, but often this is time consuming, costly, and too difficult.
- An approximate solution will suffice. Evolutionary algorithms do not provide guarantees on the solution optimality, and do not necessarily find the optimal solution. But in many practical problem domains we do not require strict optimality; a solution is good enough if it is better than the competition.

Computer scientists often compare algorithms on the basis of their computational efficiency – how fast can the algorithm solve a problem. But a second factor affecting the usefulness of algorithms is their economy: How many resources do you need to invest in designing and implementing an algorithm before it can produce useful results. Evaluating algorithms based on performance alone is equivalent to pricing products without amortizing their development costs. As the cost of development labor increases and the cost of computing power decreases, we begin to favor algorithms that are easy to implement and require little formal knowledge in the problem domain, even if they are computationally less efficient. These trends are becoming more pronounced as we venture into new scientific and engineering domains where human intuition is poor.

Robotics is one area where these criteria are met: The physics are well understood, the building blocks are well known, there is little formal design knowledge, and approximate solutions will suffice. But there are many more examples, especially in emerging fields. An example of a domain where these criteria are met is micro-scale design. For example, microphotonics devices manipulate light (like microelectronic devices manipulate electrons). Their function depends on manipulating photons at the quantum level. The physics is well understood, as we can predict the behavior of photons by solving Maxwell's equations; the building blocks are well defined, as we know the capabilities of microfabrication tools; little design knowledge exists as few people have the intuition to design structures that manipulate light at sub-wavelength scales; and approximate solutions will suffice as current solutions are suboptimal anyway.

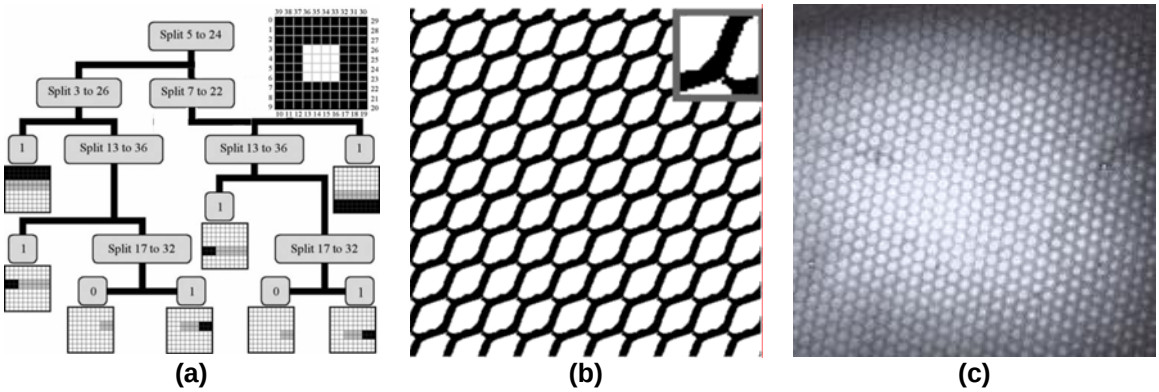

Figure 10. Evolving photonic crystal geometries: (a) A tree representation is used to encode geometry of a photonic cell by specifying a hierarchy of partition lines. The tree shown encodes the square ring shown at the top right. (b) Evolving structures with large photonic bandgaps produced this structure (unit cell shown in inset). This structure has a bandgap that is 10% larger than any human-designed pattern [26]. (c) A transmission electron micrograph of the Sea Mouse spine (notoseta). The dark areas are chitin and the light areas are voids, with a spacing of 510nm, From [24].

One notable challenge is the design of regular (periodic) structures that confine light, known as *photonic crystals*. Figure 10a shows a way of representing the geometry of a photonic cells as a hierarchy of partitions. Evolving cell representations and checking their ability to confine light using a simulator, produced an interesting pattern shown in Figure 10b. This pattern outperforms human-designed patterns by 10% (bandgap size). It is interesting to note that similar skewed-hexagonal pattern also appear in nature for the same purpose (Figure 10b).

Future challenges

The field of evolutionary design is seeking to push the limits of what can be designed automatically. The question of how complex systems can be synthesized is fundamental from three perspectives: AI research interested in automating discovery processes, engineering research in understanding the design process, and biology research interested in the origin of complexity. These perspectives are captured well in these statements:

> "*One may wonder, [...] how complex organisms evolve at all. They seem to have so many genes, so many multiple or pleiotropic effects of any one gene, so many possibilities for lethal mutations in early development, and all sorts of problems due to their long development.*"

(J. T. Bonner, *The evolution of complexity* [9], 1988, p. 173.)

> "*Today more and more design problems are reaching insoluble levels of complexity… these problems have a background of needs and activities which is becoming too complex to grasp intuitively… The intuitive resolution of contemporary design problems simply lies beyond a single individual's integrative grasp.*"

(C. Alexander, *Notes on the synthesis of form* [1], 1964, pp. 3-5)

Scalability of open-ended evolutionary processes depends on their ability to exploit functional modularity, structural regularity and hierarchy [16]. Functional modularity creates a separation of function into structural units, thereby reducing the amount of coupling between internal and external behavior on those units and allowing evolution to reuse them as higher-level building blocks. Structural regularity is the correlation of patterns within an individual. Example of regularity are repetition of units, symmetries, self-similarities, smoothness, and any other form of reduced information content. Regularity allows evolution to specify increasingly extensive structures while maintaining short description lengths. Hierarchy is the recursive composition of function and structure into increasingly larger and adapted units, allowing evolution to search efficiently increasingly complex spaces.

The existence of modular, regular and hierarchical architectures in naturally evolved systems is well established [30,11]. Though evolutionary processes have been studied predominantly in biological contexts, they exist in many other domains, such as language, culture, social organization and technology [2,35], among many others. But principles of modularity, regularity and hierarchy are nowhere as dominant as they are in engineering design. Tracing the evolution of technology over generations of products, one can observe numerous instances of designs being encapsulated into modules, and those modules being used as standard higher-level building blocks elsewhere. Similarly, there is a pressure to reduce the information content in designs, by repeating or reusing the same modules where possible, using symmetrical and regular structures, and standardizing on components and dimensions. These and other forms of regularity translate into reduced design, fabrication and operation costs. The organization of engineering designs, especially as complexity increases, is typically hierarchical. The hierarchy is often organized such that the amount of information is distributed uniformly across levels, maintaining a 'manageable' extent of information at each stage. These principles of modularity, regularity and hierarchy are cornerstones of engineering design theory and practice [e.g. 29]. Though these principles are well established, there is – like biological evolution – still a lack of a formal understanding of how and why modular, regular and hierarchical structures emerge and persists, and how can we computationally emulate these successful principles in the design automation processes.

- Functional modularity is the structural localization of function.

  In order to measure functional modularity, one must have a quantitative definition of function and structure. It is then possible to take an arbitrary chunk of a system and measure the dependency of the system function on elements of that chunk. The more that the dependency *itself* depends on elements outside the chunk, the less the function of that chunk is localized, and hence the less modular it is. If we represent dependencies as second derivatives of function with respect to pairs of parameters (i.e. the Hessian matrix of the fitness), then modules will be collections of parameters that can be arranged with lighter off-diagonal elements [33].

- Structural regularity is the compressibility of the description of the structure.

The more the structure contains repetitions, near-repetitions, symmetries, smoothness, self similarities, etc, the shorter its description length will be. The amount of regularity can thus be quantified as the inverse of the description length or of its Kolmogorov complexity.

- Hierarchy of a system is the recursive composition of structure and/or function.

  The amount of hierarchy can be quantified given the connectivity of functional or structural elements (e.g. as a connectivity graph). The more the distribution of connectivities path lengths among pairs of elements approximates a power law distribution, the more hierarchical the system is.

Principles of design

Modularity and regularity are independent principles. Principles of modularity and regularity are often confused in the literature through the notion of *reuse*. Indeed, modularity has several advantages, one of which is that modules can be used as building blocks at higher levels, and therefore can be *repeated*. Nonetheless, it is easy to imagine a system that is composed of modules, where each module appears only once. For example, opening the hood of a car reveals a system composed of a single engine, a single carburetor, and a single transmission. Each of these units appears only once (i.e., is not reused anywhere else in the system), but can be considered a module as its function is localized. Its evolutionary advantage is that it can be adapted more independently, with less impact of the adaptation on the context. A carburetor may be swapped to a newer technology, without affecting the rest of the engine system.

Similarly, there are instances of regularity without modularity: The smoothness of the hood of the car, for example, reduces the information content of the structure but does not involve the reuse of a particular module.

Though these principles are independent, they often appear in tandem and hence the confusion: we tend to speak of useful modules being reused as building blocks, and indeed recurrence of a pattern may be an indication of its functional modularity, though not a proof of it.

An inherent tradeoff exists between modularity and regularity through the notion of *coupling*. Modularity by definition reduces coupling, as it involves the localization of function. But regularity increases coupling as it reduces information content. For example, if a module is reused in two different contexts, then the information content of the system has reduced (the module needs to be described only once and then repeated), but any change to the module will have an effect on both places. Software engineers are well aware of this tradeoff: As a function is encapsulated and called from an increasing number of different contexts in a program, so does modifying it become increasingly difficult because it is entangled in so many different functions.

The tradeoff between modularity and reuse is also observed in engineering as the tradeoff between modularity and optimality. Modularity often comes at the expense of optimal performance. Systems that are less modular, i.e. more integrated, can be more efficient in their performance as information, energy and materials can be passed directly within the system, at the expense of increased coupling. Software engineers are familiar

with ‘long jumps’ and ‘global variables’ that have this effect; similarly, mechanical products will often achieve optimality of performance or cost through integration of parts into monolithic components wherever possible. The increased performance gained by reduction of modularity is often justified in the short term, whereas increased modularity is often justified in longer time scales where adaptation becomes a dominant consideration.

It is not clear whether modularity, regularity and hierarchy are properties of the system being evolved (i.e. the ‘solution’), or of the target fitness specification (i.e. the ‘problem’). It may well be that there is a duality between these viewpoints. The evolutionary computation literature contains several instances of test functions that are themselves modular (separable, e.g. Royal Roads [20]), hierarchical (e.g. Hierarchical-IFF [31]), and regular (e.g. one-max). It is not surprising then to see corresponding algorithms that are able to exploit these properties and find the solutions to these problems.

Engineers often go to great lengths to describe design goals in a way that is solution-neutral, i.e., that describes target functionality while placing the least constraints on the solution. Indeed engineering design is notorious for having multiple – even many – solutions to any given problem, without any solution being clearly superior. The fact that modular, regular and hierarchical solutions are more attractive is because – we conjecture – the design process itself tends to prefer those for reasons of scalability. It is therefore plausible that in search of scalable algorithms for synthesizing solutions bottom up, we should avoid test functions that have an inherent modular or hierarchical reward, and have these solution properties emerge from the search process itself.

Research methodology

Though robotic systems provide an intuitive and appealing substrate to explore many of these questions, they also pose many difficulties: They are computationally expensive to simulate and difficult to construct physically. But more importantly, they contain – like biology – many beautifully complex but arbitrary details that obscure the universal principles that we are looking for. There is always a temptation to increase the fidelity of the simulators, adding more biologically-realistic details, in hopes that this would lead to more life-like behaviors. However, it is sometimes more fruitful to investigate these questions in a simpler, more transparent substrate. In fact we look for the *minimal substrate* that still exhibits the effects we are investigating. Many insights can be gained by looking at these simplified systems, and the lessons learned brought to bear on the complex problems of practical importance. Thus much of the research in evolutionary design and evolutionary robotics is disguised as experiments in much more abstract systems.

Concluding Remarks

We have followed through a number of cases where principles of biological evolution have been used to automate the design of machines – from relatively simple examples in controller design, to design and fabrication of complete functional machines in physical reality, sometimes outperforming the human designs. Unlike other forms of biomimicry, however, we are not seeking to *imitate the solutions* that present themselves in nature –

like the gecko's feet, a bird's wing or a human's muscle – because these solutions were optimized for very specific needs and circumstances that may not reflect our requirements and unique capabilities. Instead, we chose to *imitate the process* that led to these solutions, as biology's design process has shown time and again its ability to discover new opportunities.

It is clear that the complexity of engineering products is increasing to the point where traditional design processes are reaching their limits. More manpower is being invested in managing and maintaining large systems than designing them, and this ratio is likely to increase because no single person can fathom the complexities involved. Alexander's quote (above) is truer today than it was in the '60s: Engineering and science are moving into scales and dimensions where people have little or no intuitions and the complexities involved are overwhelming. One way out of this conundrum is to design machines that can design for us – this is the future of engineering.

Further Reading

Digital Biology, Peter Bentley, Simon & Schuster, 2004

Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines, Stefano Nolfi and Dario Floreano, Bradford Books, 2004

Out of Control: The New Biology of Machines, Social Systems and the Economic World, Kevin Kelly, Perseus Books Group, 1995

References

1. Alexander C. A., (1970) *Notes on the Synthesis of Form*, Harvard University Press
2. Basalla G., (1989) *The Evolution of Technology*, Cambridge University Press
3. Bentley, P. J. and Kumar, S. (1999). Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem. Genetic and Evolutionary Computation Conference (GECCO '99), July 14-17, 1999, Orlando, Florida USA, pp.35-43. RN/99/2
4. Bongard J. C., Lipson H., (2004) "Once More Unto the Breach: Automated Tuning of Robot Simulation using an Inverse Evolutionary Algorithm", Proceedings of the Ninth Int. Conference on Artificial Life (ALIFE IX), pp.57-62
5. Bongard J., Lipson H. (2004), "Automated Damage Diagnosis and Recovery for Remote Robotics", IEEE International Conference on Robotics and Automation (ICRA04), pp. 3545-3550
6. Bongard, J. and Lipson, H. (2004) "Integrated Design, Deployment and Inference for Robot Ecologies", Proceedings of Robosphere 2004, November 2004, NASA Ames Research Center, CA USA
7. Bongard, J. C. (2002) Evolved Sensor Fusion and Dissociation in an Embodied Agent, in Proceedings of the EPSRC/BBSRC International Workshop Biologically-Inspired Robotics: The Legacy of W. Grey Walter, pp. 102-109

8. Bongard, J. C. and R. Pfeifer (2003) Evolving Complete Agents Using Artificial Ontogeny, in Hara, F. and R. Pfeifer, (eds.), Morpho-functional Machines: The New Species (Designing Embodied Intelligence) Springer-Verlag, pp. 237-258
9. Bonner J.T., (1988) *The Evolution of Complexity by Means of Natural Selection*, Princeton University Press
10. Goldberg, D. E., (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley
11. Hartwell L.H., Hopfield J.H., Leibler S. and Murray A.W., 1999, "From molecular to modular cell biology", Nature 402, pp. C47-C52
12. Hornby G.S., Lipson H., Pollack. J.B., 2003 "Generative Encodings for the Automated Design of Modular Physical Robots", IEEE Transactions on Robotics and Automation, Vol. 19 No. 4, pp 703-719
13. Jakobi, N. (1997). Evolutionary robotics and the radical envelope of noise hypothesis. Adaptive Behavior, 6(1):131–174.
14. Koza J., (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press
15. Lipson H. (2004) "How to Draw a Straight Line Using a GP: Benchmarking Evolutionary Design Against 19th Century Kinematic Synthesis", Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO 2004
16. Lipson H. (2004) "Principles of Modularity, Regularity, and Hierarchy for Scalable Systems", Genetic and Evolutionary Computation Conference (GECCO'04) Workshop on Modularity, regularity and Hierarchy
17. Lipson H., Pollack J. B. (2000) Automatic design and manufacture of artificial lifeforms. *Nature*, 406:974–978
18. Luke, S. and L. Spector. 1996. Evolving Graphs and Networks with Edge encoding: Preliminary Report. In Late Breaking Papers at the Genetic Programming 1996 Con-ference (GP96). J. Koza, ed. Stanford: Stanford Bookstore. 117-124
19. Malone E., Lipson H., (2004) "Functional Freeform Fabrication for Physical Artificial Life", Ninth Int. Conference on Artificial Life (ALIFE IX), Proceedings of the Ninth Int. Conference on Artificial Life (ALIFE IX), pp.100-105
20. Melanie Mitchell, (1996) An introduction to genetic algorithms, MIT Press
21. Mytilinaios E., Marcus D., Desnoyer M., Lipson H., (2004) "Designed and Evolved Blueprints For Physical Self-Replicating Machines", Proceedings of the Ninth Int. Conference on Artificial Life (ALIFE IX), pp.15-20
22. Nolfi S., Floreano D. (2004), Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines, Bradford Books
23. Papadimitriou C.H., Steiglitz K., *Combinatorial Optimization : Algorithms and Complexity*, Dover Publications

24. Parker, A.R., McPhedran, R.C., McKenzie, D.R., Botten, L.C. and Nicorovici, N.-A.P., Aphrodite's iridescence. Nature (2001) 409, 36-37

25. Paul, C. and J. C. Bongard (2001) "The Road Less Traveled: Morphology in the Optimization of Biped Robot Locomotion", in Proceedings of The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2001), Hawaii, USA

26. Preble S.F., Lipson H., Lipson M. (2004) "Novel two-dimensional photonic crystals designed by evolutionary algorithms", in M. Lipson, G. Barbastathis, A. K. Dutta, K. Asakawa (Eds.) *Nanophotonics for Communication: Materials and Devices*, Proceedings of SPIE Volume: 5597, pp. 118-128

27. Saylor J. Walker K., Moon F.C., Henderson D.W., Daimina D., Lipson H., Cornell University Digital Library of Kinematic Models (KMODDL), http://kmoddl.library.cornell.edu

28. Sims K. "Evolving 3D morphology and behaviour by competition". Artificial Life IV, pages 28–39, 1994.

29. Suh N.P. 1990 *The Principles of Design*, Oxford University Press, Oxford UK

30. Wagner, G.P., Altenberg L., 1996 "Complex adaptations and the evolution of evolvability" Evolution 50:967-976

31. Watson, R.A. and Pollack, J.B. (1999). "Hierarchically-Consistent Test Problems for Genetic Algorithms", Proceedings of 1999 Congress on Evolutionary Computation (CEC 99). Angeline, Michalewicz, Schoenauer, Yao, Zalzala, eds. IEEE Press, pp.1406-1413

32. Willis, R., (1841), *Principles of Mechanism*, London (available online at KMODDL [27])

33. Wyatt D, Lipson H., (2003) "Finding Building Blocks Through Eigenstructure Adaptation", Genetic and Evolutionary Computation Conference (GECCO '03)

34. Yim, M., Zhang, Y. and Duff, D., "Modular Reconfigurable Robots, Machines that shift their shape to suit the task at hand," IEEE Spectrum Magazine cover article, Feb. 2002

35. Ziman J, (2003) *Technological Innovation as an Evolutionary Process*, Cambridge University Press

36. Zykov V., Bongard J., Lipson H., (2004) "Evolving Dynamic Gaits on a Physical Robot", Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO'04