Computer Evolution of Buildable Objects for *Evolutionary Design by Computers*

by Pablo J. Funes and Jordan B. Pollack

1 Introduction

This chapter describes our work in evolution of buildable designs using miniature plastic bricks as modular components. Lego[1] bricks are well known for their flexibility when it comes to creating low cost, handy designs of vehicles and structures. Their simple modular concept make toy bricks a good ground for doing evolution of computer simulated structures which can be built and deployed.

Instead of incorporating an expert system of engineering knowledge into the program, which would result in more familiar structures, we combined an evolutionary algorithm with a model of the physical reality and a purely utilitarian fitness function, providing measures of feasibility and functionality.

Our algorithms integrate a model of the physical properties of Lego structures with an evolutionary process that freely combines bricks of different shape and size into structures that are evaluated by how well they perform a desired function. The evolutionary process runs in an environment that has not been unnecessarily constrained by our own preconceptions on how to solve the problem.

The results are encouraging. The evolved structures have a surprisingly alien look: they are not based in common knowledge on how to build with brick toys; instead, the computer found ways of its own through the evolutionary search process. We were able to assemble the final designs manually and confirm that they accomplish the objectives introduced with our fitness functions.

This chapter discusses background and related work first (section 2), then goes on to describe our methods; first the model we use to simulate Lego structures (sections 3-4), then the representation and evolutionary algorithms (section 5). The results sections (6-7) discuss applications, showing the results of several evolutionary runs and illustrating with pictures of the final assembled Lego artifacts. Finally, on sections 8-9, current and future lines of work and conclusions are drawn.

2 Background

In order to evolve both the morphology and behavior of autonomous devices which can be manufactured, one must have adequate representations and simulations. The representation must provide the computer with ways to create, manipulate and modify an infinite variety of virtual architectures to be tested in simulation. The objects being simulated need to be adaptive enough to cover the gap between simulated and real world, so they will perform correctly when built. Desirable features of a software engine for evolving morphology are:

- **Universal** - the simulator should cover an infinite general space of mechanisms.
- **Conservative** - because simulation is never perfect, it should preserve a margin of safety.
- **Efficient** - it should be quicker to test in simulation than through physical production and test.
- **Buildable** - results should be convertible from a simulation to a real object

1. Lego is a registered trademark of the Lego group.

With a representation and a physical simulation that follow these ideas, we have obtained some promising results. In a first stage we worked with two-dimensional structures only (Funes and Pollack 1997). We have recently extended our framework to three dimensions; one 3D application is described here as well.

There are several fields which bear on this questions of representation and physical simulation, including qualitative physics and structural mechanics, computer graphics, evolutionary design and robotics.

2.1 Qualitative Physics

Qualitative Physics is the subfield of artificial intelligence (AI) which deals with mechanical and physical knowledge representation. It starts with a logical representation of a mechanism, such as a heat pump (Forbus, 1984) or a string (Gardin and Meltzer, 1989), and produces simulations, or envisionments, of the future behavior of the mechanism. QP has not to our knowledge been used as the simulator in an evolutionary design system.

2.2 Computer Graphics.

The work of Karl Sims (Sims, 1994 and 1994b) was seminal in the fields of evolutionary computation and artificial life. Following Ngo and Marks (Ngo and Marks, 1993), Sims evolved virtual creatures that have both physical architecture and control programs created by an evolutionary computation process.

Despite their beautiful realism, Sims' organisms are far from real. His simulations do not consider the mechanical feasibility of the articulations between different parts, which in fact overlap each other at the joints, nor the existence of real world mechanisms that could produce the forces responsible for their movements. There was no attempt to emulate a real environment that could house mechanical counterparts of those virtual creatures.

2.3 Structural Mechanics/Structural Topology

The engineering field of structural mechanics is based on methods, such as finite element modelling (Zienkiewicz, 1977) to construct computable models of continuous materials by approximating them with discrete networks. These tools are in broad use in the engineering community, carefully supervised and oriented towards particular product designs, and are often quite computationally intensive. Applications of genetic algorithms to structural topology optimization (Chapman *et al.*, 1993; Shoenauer, 1996) are related to our work. This type of application uses genetic algorithms as a search tool to optimize a shape under clearly defined preconditions. The GA is required, for example, to simultaneously maximize the stiffness and minimize the weight of a piece subject to external loads (Chapman *et al.*, 1993).

2.4 Evolutionary Design

Evolutionary Design, the creation of new designs by computers, using evolutionary computation methods (Bentley, 1996), is a new research area with an enormous potential. Among the different approaches and techniques represented in the present volume, our direction is to exploit modular components to create complete functional structures.

While other research focuses in evolution of abstract shapes or optimization of one part or component, the line we are proposing is to let the evolutionary process take care of the entire design process by means of recombination of available components and evaluation of functionality through physics simulation.

2.5 Evolutionary Robotics

Work in evolutionary robotics has traditionally focused in the evolution of robot controllers to provide a given robot platform — either real or simulated — with a custom brain that, once downloaded, will produce an adequate behavior (Mataric and Cliff, 1996). The process of adaptation through evolutionary search allows these artificial life forms with evolved brains to perform in the environments they inhabit. Some experiments rely on carefully designed simulations (Cliff *et al.*, 1996), while others apply evolution directly in the real robot (Floreano and Mondada, 1994). Hybrid techniques (Lund, 1995) are a mixture of the two.

Whereas some of the most interesting work in artificial life — Karl Sims' for example — involves evolution of morphology and control together, researchers in evolutionary robotics use human designed robot machines and try to evolve control programs for them. The observation can be made, however, that evolution of a creature's controlling brain addresses just one part of the problem of artificially evolving life forms: a creature that adapts to an environment needs an adequate body to inhabit. In nature, the brain for a body, and the body for a brain are exquisitely intertwined and co-adapted after millions of years of coevolution.

The idea of co-evolving bodies and brains is becoming popular. Recent work by Lund, Hallam and Lee (Lund *et al.*, 1997; Lee *et al.* 1996), for example, evolves in simulation a robot control program simultaneously with some parameters of its morphology such as sensor number and positioning and body size. Our work attempts to build from the opposite shore: we are using evolutionary techniques to create structures, physical forms, adapted to perform correctly in the physical world. This is a step on the way to the full co-evolution of morphology and behavior we believe is necessary for the development of robots and brains with higher complexity than humans can engineer.

3 Modelling Lego structures under stress

We begin the description of our methods with the modelling procedure used to test in simulation the behavior of virtual structures produced by an evolutionary process of genetic crossover and mutation.

The resistance of the plastic material (ABS-acrylonitrile butadiene styrene) of Lego bricks far surpasses the force necessary to either join two of them together or break their unions. This makes it possible to design a model that ignores the resistance of the material and evaluates the strain forces over a group of bricks only at their union areas. If a Lego structure fails, it will generally do so at the joints, but the actual bricks will not be damaged.

This characteristic of Lego structures makes their discretization for modelling an obvious step. Instead of imposing an artificial mesh for simulation purposes only —as in finite elements methods, for example— these structures are already made of relatively large discrete units.

3.1 Networks of Torque Propagation

We begin considering two-dimensional systems of forces. We measured the amount of stress that different linear (1×1, 2×1, 3×1, etc., as in fig. 2) unions of brick pairs can support (table 1). The main simplification comes from the observation that a "fulcrum" effect, the angular torque exerted over such a joint, constitutes the principal cause for the breakage of a stressed pair of Lego bricks. Thus a critical abstraction for the purpose of modelling has come from disregarding radial forces such as vertical pulls, and describing the system of static forces inside a complex structure of Lego bricks as a network of "rotational" joints located at each union between brick pairs and subject to loads coming from the weight of each brick (fig. 1).

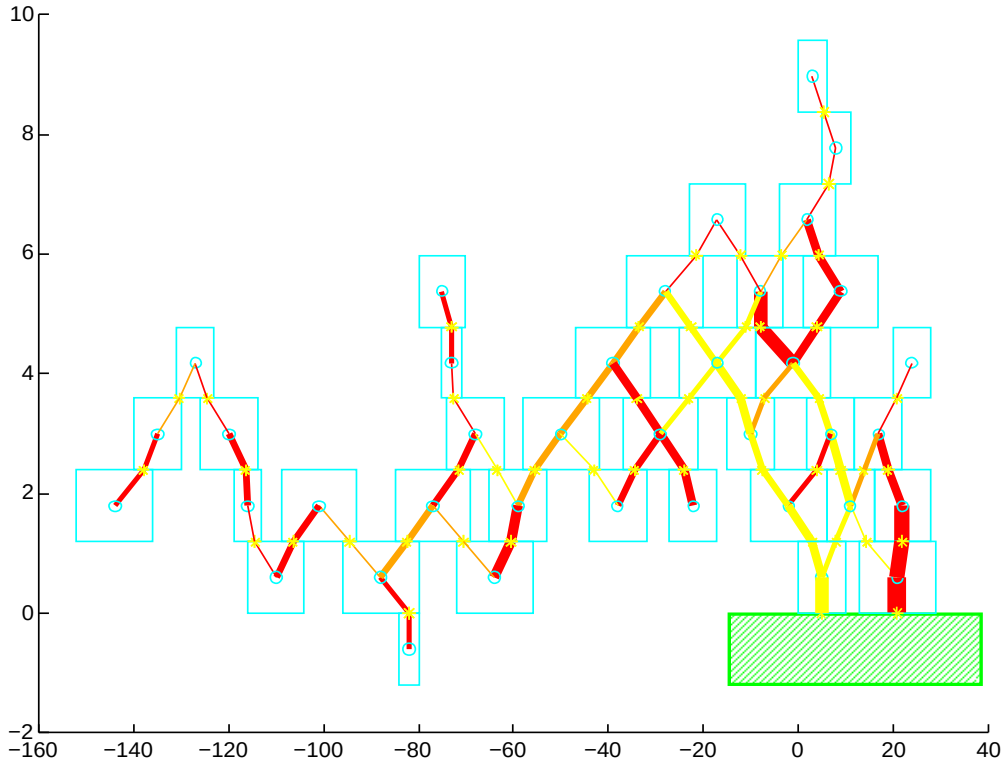

fig. 1. Model of a 2D lego structure showing the brick outlines (rectangles), centers of mass (circles), joints (diagonal lines, with axis located at the star), and "ground" where the structure is attached (shaded area). The thickness of the joint's lines is proportional to the strength of the joint. A distribution of forces was calculated: highly stressed joints are shown in **yellow/light** color, whereas those more relaxed are **red/darker**. Note that the *x* and *y* axis are in different scales.

Joint size(knobs)	Approximate torque capacity (N-m $\times 10^{-6}$)
1	10.4
2	50.2
3	89.6
4	157.3
5	281.6
6	339.2
7	364.5

Table 1. Estimated minimal torque capacities of the basic types of joints

Given a structure formed by a combination of bricks, our model builds a network with joints of different capacities and external forces that must be in static equilibrium if the structure is not going to collapse. Each idealized joint is located at the center of the area of contact between a pair of bricks.

Each force applied to a brick, either its own weight or an external load, has to be cancelled by one or more reaction forces if the brick is stable — otherwise it would be falling. Such reaction forces can originate in any of the joints that connect it to neighbor bricks. In that case the force is transmitted through the joint to a connected brick. Thus a load is propagated through the network until finally absorbed by a fixed body — the "ground".

If a solution to this network exists, it means that there is a way to distribute all the forces along the structure. Our operating heuristic is this: As long as there is a way to distribute the weights among the network of bricks such that no joint is stressed beyond its maximum capacity, the structure will not break.

From this strategy of modelling an algorithmic problem arises. Where nature simply distributes work dynamically through small deformations along the structure, our model needs an algorithm to determine the existence of solutions. We have not found a complete algorithm for this problem, but a greedy (Cormen, *et al.*, 1989, p. 239) technique, not always capable of finding the solution when there is one, guarantees the stability of the structure in the numerous cases when a solution is actually found. Our model is thus conservative. It might be wrong in predicting the breakage of a structure, but any shape approved by it is guaranteed to resist the required loads.

3.2 From 2- to 3-dimensional networks

To extend our model of networks of torque propagation to cover three-dimensional brick structures, our definition of joint needs to be extended. Where before all brick unions could be described with one integer quantity, the number of knobs that join two bricks, in the three dimensional case these unions will be *n*-by-*m* rectangles. Two 2×4 bricks for example can be stuck together in 8 different types of joints: 1×1, 1×2, 1×3, 1×4, 2×1, 2×2, 2×3, 2×4. We know already, from the one dimensional case, how *n*×1 unions respond to forces acting along the *x* axis alone. A 2×1 union supports more than double the torque admitted by a 1×1, the reason being that the brick itself acts as a fulcrum (fig. 2). The distance from the border to the first knob is shorter than the distance to the second knob, resulting in a lower multiplication of the force for the second knob. This fulcrum effect does not happen when the force is orthogonal to the line of knobs. A 2×1 union can be considered as two 1×1 unions, or as one joint with double the strength of a 1×1 (fig. 3).

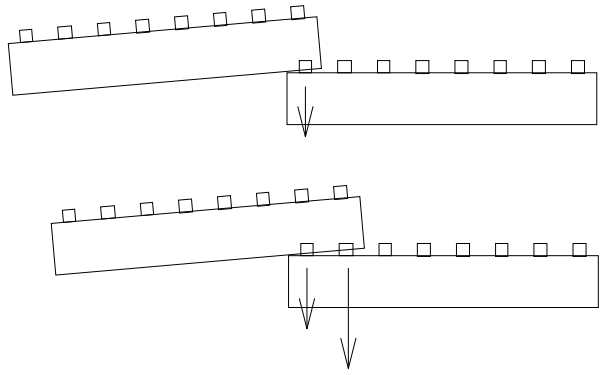

fig. 2. Fulcrum effect: a 1×2 union resists more than twice the load of a 1×1 because the second knob is farther away from the axis of rotation.

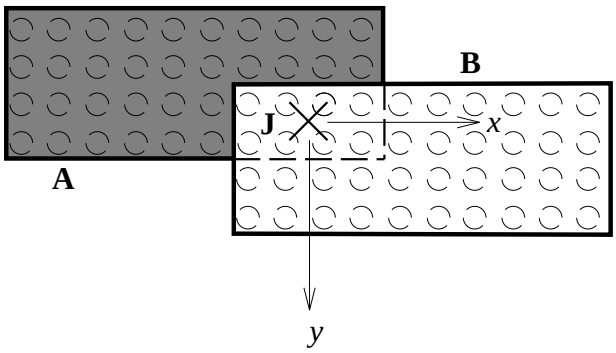

fig. 3. Two-dimensional brick joint: bricks A and B overlap in a 4×2 joint J. Along *x* the joint is a double 4×1 joint. Along the *y* axis it is a quadruple 2×1-joint.

Following these ideas we enunciate the following rule: Two bricks united by $n \times m$ overlapping knobs will form a joint with a capacity *Kx* along the *x* axis equal to *m* times the capacity of one *n*-joint and *Ky* along the *y* axis equal to *n* times the capacity of an *m*-joint.

To test the resistance of this composite joint to any spatial force *f* we have to separate it into its two components, *fx* on the *xz* plane and *fy* on the *yz* plane. These components induce two torques τx, τy. To break the union either τx must be larger than *Kx* or τy larger than *Ky*. With this procedure we induce, from a 3-dimensional brick structure, two separate 2-dimensional systems of joints, one for the *x* components of torques and joints and the other for the *y* components. The structure is stable if and only if both 2-dimensional projections are stable networks (figs. 5 and 6).

We are assuming that a dimensional independence hypothesis is true; it could be the case, however, that a force exerted along one axis will either weaken or strengthen the resistance in the orthogonal dimension. We made some exploratory experiments that suggested that the presence of stress along one axis does not modify the resistance along the other. It is probably the case that the rectangular shape of the joint makes it stronger for diagonal forces, justifying this simplification.

3.3 Limitations of modelling

We know that this kind of naive modelling is not a complete description of the highly complex physical interactions that are taking place. However, we expect that considering an adequate margin of error we will be able to produce useful approximations to the true behavior of actual Lego bricks.

The properties of Lego bricks are variable. Differences in construction, age, dirt, temperature, humidity and other unpredictable factors produce seemingly random variations on the measurements of their behavior. These factors have to be taken into account in order to have buildable results.

So far we have accounted for this problem using a “safety margin” of 20%. This means that our model actually assigns 20% less resistance to all the joints involved. Any model for modular structures will have to contemplate this kind of safety margins to compensate for the random variability of the generic properties of the average brick, but the value of 20% was set intuitively and may require further study, especially as our structures scale up in size and complexity.

Evolutionary roboticists have found similar unpredictabilities when attempting to simulate the environment for a real robot (Jakobi *et al.*, 1995). This has lead to the incorporation of random noise to the simulators in order to generate robust behaviors suitable to be transferred to the real world.

Our model provides only an approximation to the complex physical properties of Legos. It may be possible to complicate it a great deal to have more accurate physics. But because of the

variable, noisy properties of Legos, a highly accurate model would not remove the need for a safety margin. Noise means that real entities can not be simulated beyond the limit of variability in the measurements of their physical parameters.

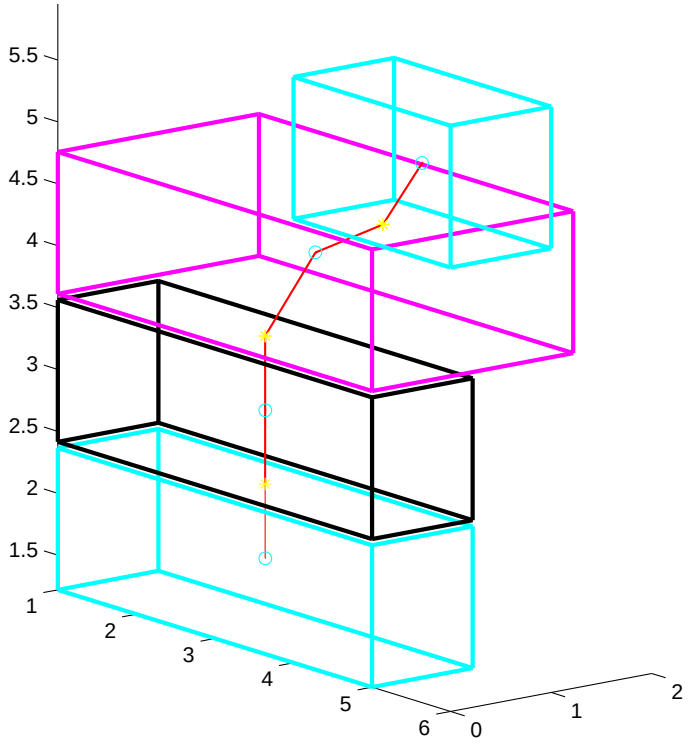

fig. 4. 3D model for a few Lego bricks. For every pair of bricks we model a "joint" located at the center of the area of contact, whose resistance depends on its x and y dimensions (see fig. 3). Each joint is labelled with a star and each center of mass with a circle.

4 Solving a model

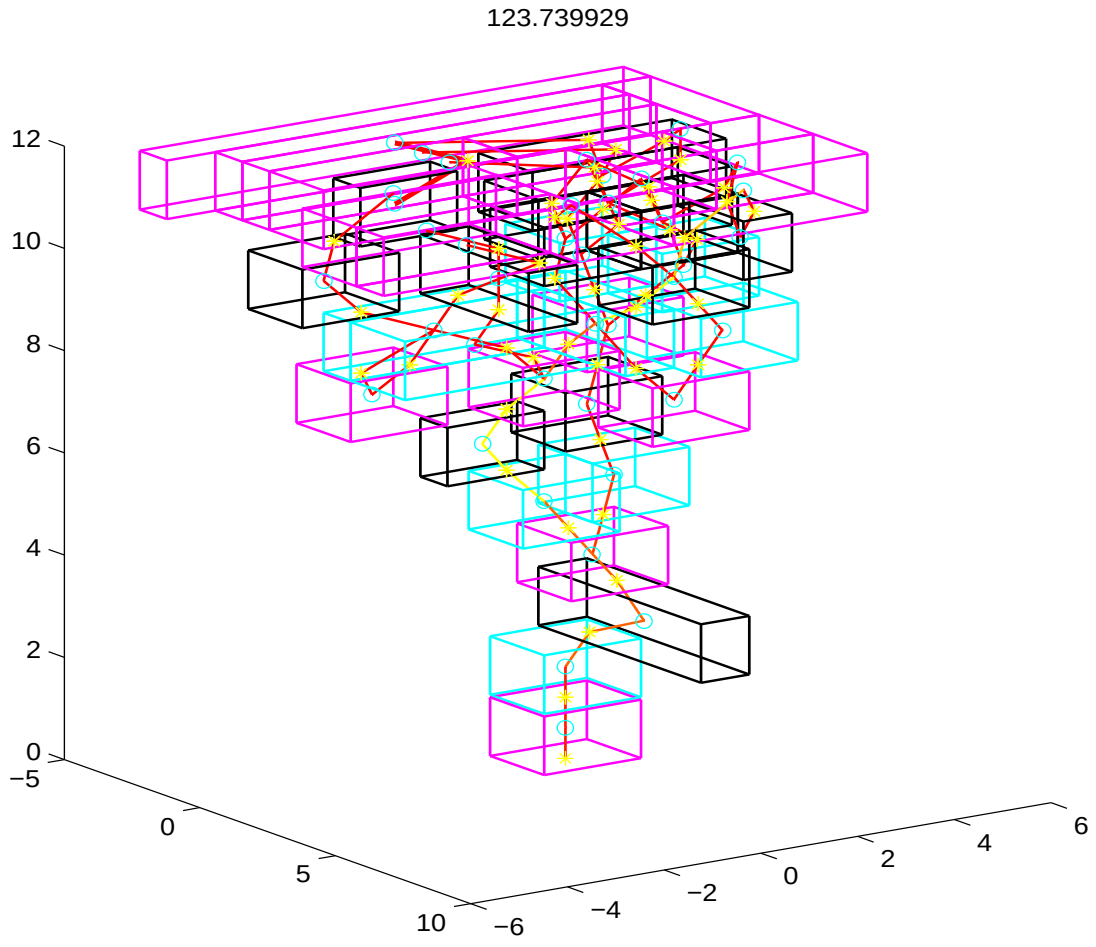

fig. 5. Structure of Lego bricks generated by our evolutionary process. The underlying physical model is shown.

Our model for a 2D structure of bricks generates a set of simultaneous interval equations that can be satisfied if and only if the structure is stable. Each force, either the weight of one of the bricks or an external load, will have to be absorbed by the joints in the structure and transmitted to the ground. The torque exerted by each joint must lie in the interval [-*K*, *K*], where *K* represents its maximum capacity as deducted from the number and disposition of overlapping knobs between two bricks.

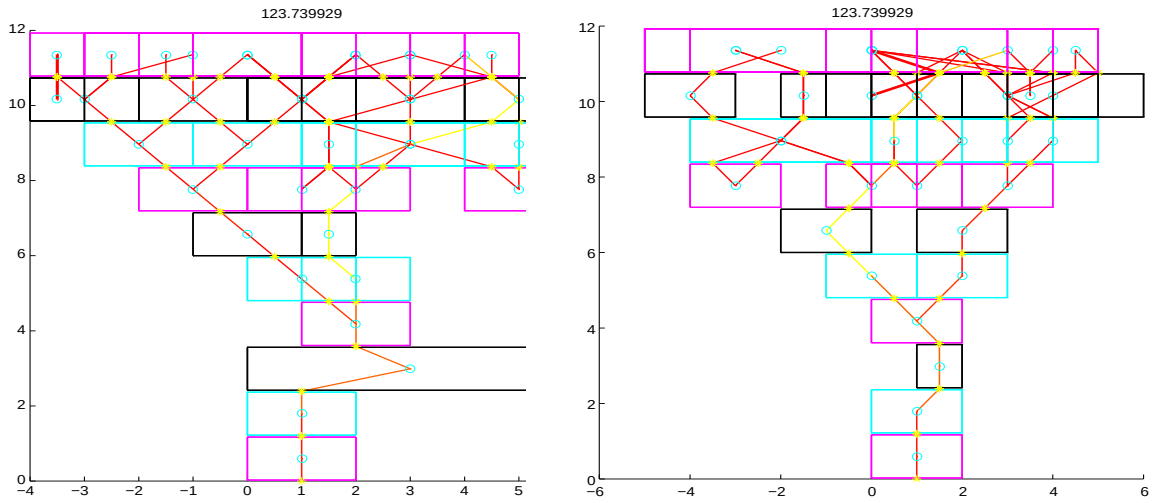

fig. 6. Projecting the 3D structure of fig. 5 to the *xz* and *yz* planes two 2D networks are obtained that can be solved independently.

It is not clear to us yet which algorithm can solve the equations and find, for a given structure and set of forces, whether or not there is a valid distribution of loads such that all forces are in equilibrium and at the same time all torques lie inside the valid intervals of the joints. But in the case where only one force is present, the problem is describable as a network flow algorithm (NFA) (Cormen *et al.*, 1989, ch. 12) and can be solved by known methods. The possibility of adapting generalized versions of network flow algorithms (Iusem and Zenios, 1995; Leighton *et al.*, 1995) to our problem remains to be explored.

By separating each 3D joint into two orthogonal and independent 2D joints, which receive the *x* and *y* components of each external force, we can project an entire 3D network model of a bricks and joints structure into two orthogonal planes, *xz* and *yz*, generating two 2D networks that can be solved separately (figs. 5 and 6). Thus the problem of solving a 3D network does not add any more complexity to the existing problem of solving 2D networks of bricks and joints.

4.1 NFA for solving a 2D network with a greedy algorithm

For each given force we consider the network of all the joints in the structure as a flow network that will absorb it transmit it to the ground. Each joint *j* can support a certain fraction α of such a force, given by the formula

$$\alpha_{j,F} = \frac{K_j}{d(j,F)f} \tag{1}$$

where K_j is the maximum capacity of the joint, *d(j,F)* is the distance between the line generated by the force vector and the joint, and *f* the magnitude of the force.

If a given force *F* is fixed and each edge on the graph is labeled with the corresponding $a_{j,b}$ according to (1), a network flow problem is obtained where the source is the brick to which the force is applied and the sinks are all the connections to the ground. A net flow of 1 represents a valid distribution of the force *F* throughout the structure.

The complete problem is not reducible, however, to an NFA, due to the fact that there are multiple forces to be applied at different points, and the capacity of each joint relative to each one varies with the magnitude of the force and the orthogonal distance between force and joint.

Leaving aside the study of better algorithmic implementations, we are using a greedy algorithm: once a solution has been found for the distribution of the first mass, it is fixed, and a remaining capacity for each joint is computed that will conform a reduced network that must support the next force, and so on.

While there may be a better algorithm for solving the weight distribution for a stable Lego structure, an incomplete algorithm could be enough for many applications. Any structure that is approved as "gravitationally correct" by our simulation possesses a load distribution that does not overstress any joint, and thus will not fall under its own weight. Our evolutionary algorithm might be limited by the simulation when it fails to approve a structure that was physically valid, but still may succeed working only in the space of "provable" solutions.

5 Genetic coding for Lego structures

To evolve structures in the computer, a genetic representation is required. The evolutionary algorithm manipulates this genotype to create new alternatives by recombination and mutation of previous ones. The new variations are then tested with the simulation machinery, as described in the previous sections, to evaluate their properties and select or discard them accordingly.

Our representation borrows the standard tree mutation and crossover operators from genetic programming (Koza, 1992). We have implemented tree representations of 2D and 3D Lego structures. Each node on the tree represents a brick and has a size type parameter indicating the size of the brick and a list of descendants, which are new bricks physically attached to the parent. Each descendant node has positional parameters that describe the position of the new brick relative to the parent.

5.1 Coding for 2D and 3D structures

In the 2D version each brick node has a size type parameter (4, 6, 8, 10, 12 or 16, corresponding to the Lego bricks of size 1×4 through 1×16) and four potential sons, each one representing a new brick linked at one of its four corners (lower left, lower right, upper right, upper left). Each non-nil descendant has a "joint size" parameter indicating the number of overlapping knobs in the union.

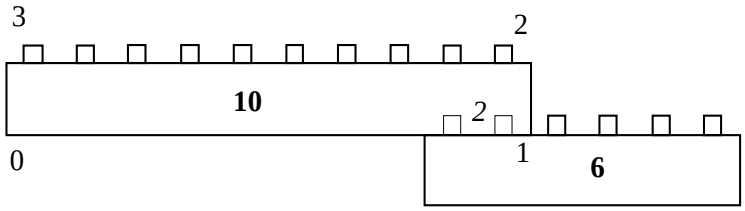

fig. 7. Example of 2D genetic encoding of bricks

The diagram on fig. 7. represents a 10-brick with its 4 joint sites labeled 0, 1, 2, 3, that is linked to a 6-brick by two overlapping knobs. The corresponding tree could be written in pseudo-Lisp notation as

(10 nil (2 (6 nil nil nil)) nil nil) (2)

In the extension to 3D we add more size types to incorporate bricks other than 1×*n* (the bricks currently available are 1×2, 1×4, 1×6, 1×8, 1×10, 1×12, 1×16, 2×2, and 2×4), and use a list of descendants, each one representing a new brick to be plugged into the parent. Each descendant brick has 3 parameters: The integer *(x, y, z)* coordinates of the new brick (relative to its parent, so for a descendant of an *n*×*m* brick, $0 \le x < n$, $0 \le y < m$ and $z \in \{-1,1\}$); a rotation parameter that specifies the orientation of the descendant relative to the parent (0˚, 90˚, 180˚ or 270˚), and the size of the descendant. As an example, the structure in fig. 4 can be codified as

$$(1\times4\ ((0,0,1)\ 0^\circ\ (1\times4\ ((0,0,1)\ 0^\circ\ (2\times4\ ((3,0,1)\ 0^\circ\ (1\times2)))))) \quad (3)$$

5.2 Mutation and Crossover

Mutation operates by either random modification of a brick's parameters (size, position, orientation) or addition of a random brick. The basic crossover operator involves two parent trees out of which random subtrees are selected. The offspring generated has the first subtree removed and replaced by the second.

After mutation or crossover operators are applied, a new, possibly invalid specification tree is formed. The result is expanded one node at a time and overlapping is checked. Whenever an overlap is found the tree is truncated at that site. With this procedure, a maximum spatially valid subtree is built from a crossover or mutation. Branches that could not be expanded are discarded.

Once a valid tree has been obtained, the physical model is constructed and the structure tested for stress stability. If approved, fitness is evaluated and the new individual is added to the population.

A problem with our representations, similar in origin to the problem of valid function parameters in genetic programming, is that it is underconstrained: Only some trees will encode valid Lego structures. Many trees describe impossible, overlapping structures. The following mutation of (3), for example, is illegal because two bricks would share the same physical space ($z = -1$ after the second brick means that the third one goes below it, but the first brick is already there).

$$(1\times4\ ((0,0,1)\ 0^\circ\ (1\times4\ ((0,0,-1)\ 0^\circ\ (2\times4\ ((3,0,1)\ 0^\circ\ (1\times2)))))) \quad (4)$$

5.3 Evolutionary Algorithm

We use a straightforward steady-state genetic algorithm, initialized with a population of one single brick. Through mutation and crossovers a population of 1000 individuals is generated and then evolved:

1. While maximum fitness < Target fitness
2. Do Randomly select mutation or crossover.
3. Select 1 (2 for crossover) random individual(s) with fitness proportional probability.
4. Apply mutation or crossover operator
5. Generate physical model and test for gravitational load
6. If the new model will support its own weight.
7. Then replace a random individual with it (chosen with inverse fitness proportional probability).

6 Evolving two-dimensional Lego structures

In this section we summarize the experiments done and the Lego designs obtained. Our first assay was the “Lego bridge”: evolving a structure attached to a table to reach over the void to a neighboring table. With appropriate fitness functions we went on to evolve other 2D structures, including longer bridges, scaffolds and cranes. Finally, our first 3D project is a table.

6.1 Reaching a target point: Bridges and Scaffolds

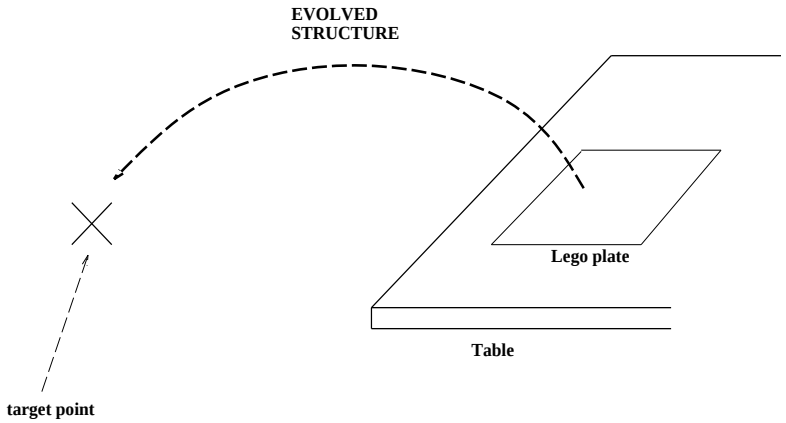

fig. 8. Basic setup: The structure starts over a Lego plate affixed to a table and has to reach a target point supporting its own weight.

In our first experiments we conceived a Lego plate affixed to a table (fig. 8) and evolved 2D structures to reach a target point, using as fitness function a normalized distance to the target point,

$$Nd\,(S,\,T) \;=\; 1 - \frac{d\,(S,\,T)}{d\,(0,\,T)} \tag{5}$$

(where S is the structure, T the target point and d the euclidean distance).

Structures not approved by the physical model where discarded. Those capable of supporting themselves — according to our simulation — where incorporated to the evolving population and selected according to this straightforward fitness measure.

With a target point located horizontally and away from the plate we generated a “Lego Bridge” (figs. 1 and 9), moving it to a remote position we obtained the “long bridge” (fig. 10), and putting it below we generated a descending structure, a “scaffold” (fig. 11).

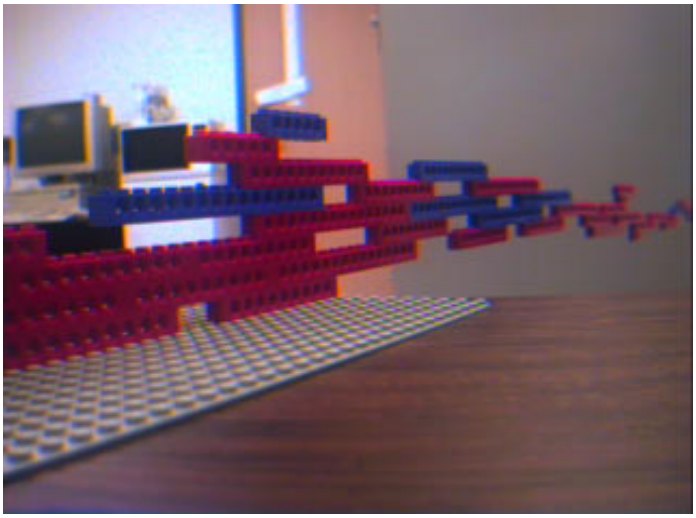

fig. 9. The “Lego bridge” defined by the scheme of fig. 1, built on our lab table.

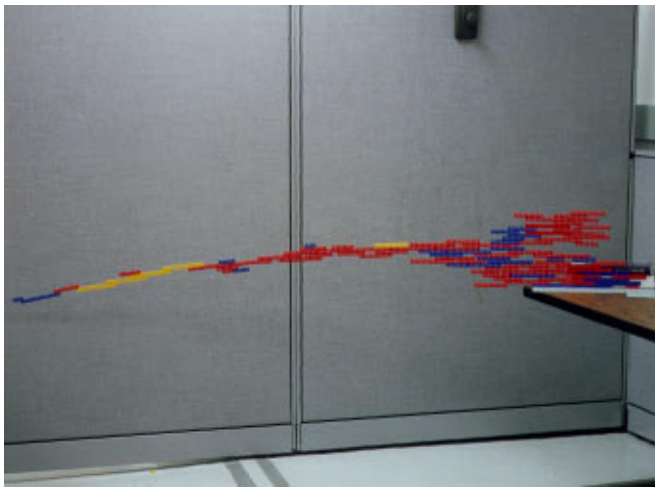

fig. 10. Long Bridge

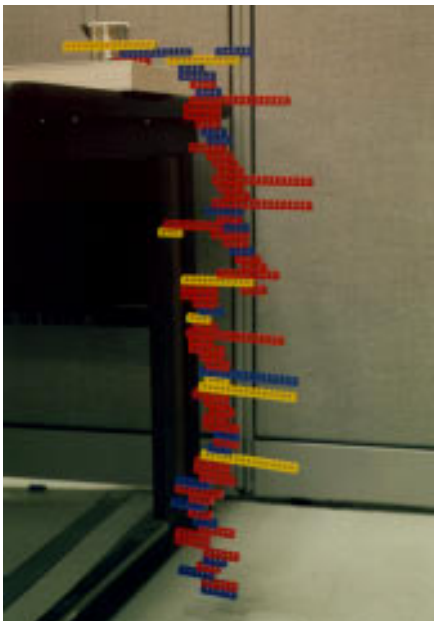

fig. 11. Scaffold.

6.2 External Loads: Horizontal crane arm

With a two-step fitness function that gives one point for reaching the target point as in equation (5) above and, if reached, additional points for supporting an external weight hanging from the last brick, we evolved a crane arm (fig. 12).

Since our algorithm gives a yes/no answer for the stability of a structure, we add the weight in small increments and test repeatedly to create a fitness function in this case. For example, for a target load *m* we can use the following fitness function

1. For i=1 to 100
2. Add a weight *m*/100 to the structure at *T*
3. If the structure does not resist, return (i-1)/100
4. return 1.0

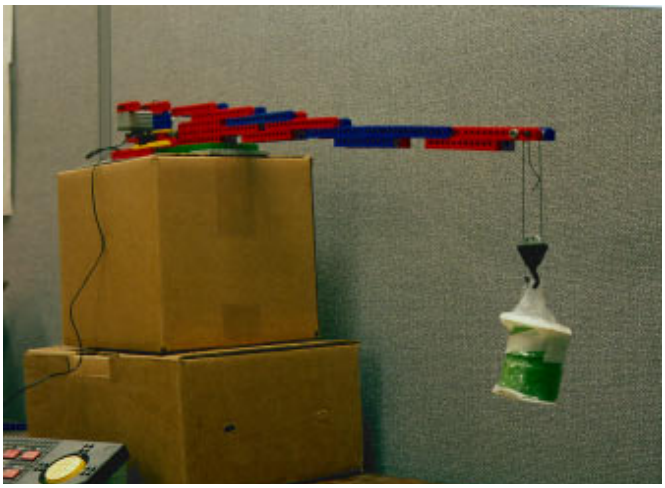

fig. 12. Crane with evolved horizontal crane arm.

6.3 Constraining the space: Diagonal crane arm

For a different type of crane we constrained the space where bricks can be located to the diagonal subspace $\{-x<y\}$. In order to evolve a crane arm that would support a weight of 250g we wrote a fitness function whose value is the fraction of 250g supported times the length of the arm along the x axis. Since no bricks can be placed below the diagonal, the resulting arm goes diagonally up and away (figs. 13 and 14).

fig. 13. Crane with diagonal crane arm.

6.4 Optimization

A comment that we often received was that our final structures are not optimized: They have useless bricks that do not serve for any apparent purpose. Of course, these irregularities are useful during the search process. Since we are not rewarding nor punishing the number of bricks used, the evolutionary search will freely generate variations with different numbers of bricks. All of them are potentially useful in the process of finding new combinations with higher fitness.

In a new run of the diagonal crane arm experiment, we added a little reward for lightness, inversely proportional to the number of bricks, but three orders of magnitude smaller than the raw fitness function. Fig. 14 shows two solutions for a crane arm the same length (a fitness value of 24). The structure on the right has a bigger premium, so we will prefer it.

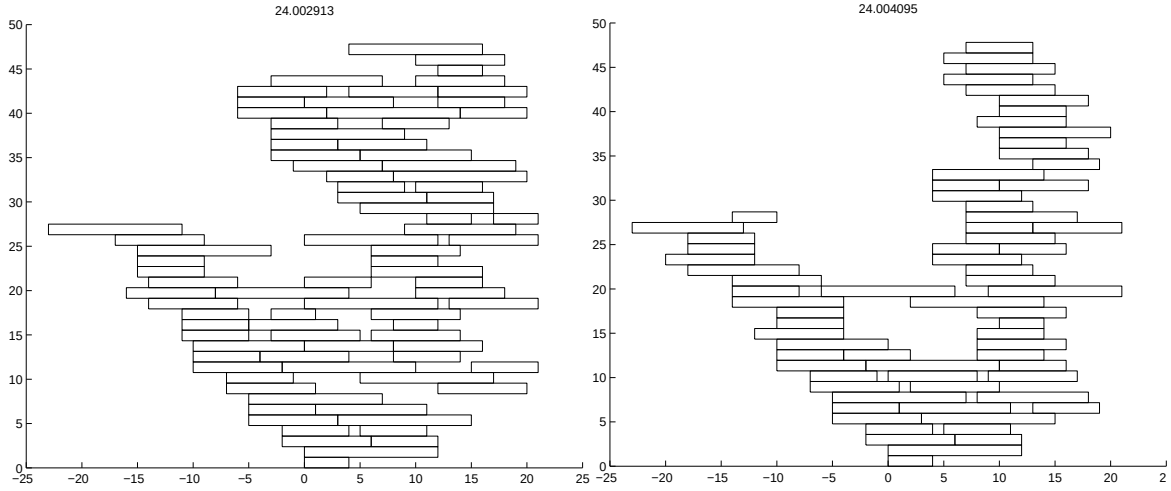

fig. 14. Optimization: Among several structures found with a raw fitness of 24, a small premium in the fitness function allows us to choose the one that uses less bricks (right). [Note that the tall column on the right cannot be eliminated because it acts as a counterbalance for the load that will be placed at the left tip of the crane].

There is a reason why the weight of the fitness of the "simplicity" factor should be small compared with the raw fitness measure (length of the arm): we are willing to sacrifice everything else for the size of the crane, which is what we are really trying to maximize. Among cranes of the same size and resistance, however, we prefer those with a smaller number of bricks. The evolutionary process must not be biased against heavier versions of the crane. In the example shown in fig. 14, fitness values of 24.0029 and 24.0040 have nearly identical chances of being selected in a fitness proportional selection scheme. But among otherwise identical cranes, the premium for optimality allows us to keep the one that is cleanest.

7 Evolving three-dimensional Lego structures

7.1 First project: a Lego table

We have run our first experiments in evolution of 3D Lego designs. Our initial project is a "table". We start with a fixed plate as in fig. 8, and want to obtain a table 10 bricks tall, with a support surface of 9×9 and capable of supporting a weight of 50 grams anywhere. There are four objectives to fulfill:

1. The height of the structure must be as required.
2. The surface most cover the target area.
3. The desired weight has to be supported all over the surface.
4. All other conditions met, a minimal number of bricks should be used.

To cover all the objectives we wrote a step fitness function that gives between 1 and 2 points for the first objective partially fulfilled, between 2 and 3 for the first objective completed and partial

satisfaction of the second, and so on. With this setup, the algorithm builds upwards first, then broadens to cover the surface, later secures that all points of the surface support a load of 50g and finally tries to reduce the number of bricks to a minimum.

One of the solutions we obtained is shown in figs. 5 and 6, and a picture of the Lego table built in fig. 15.

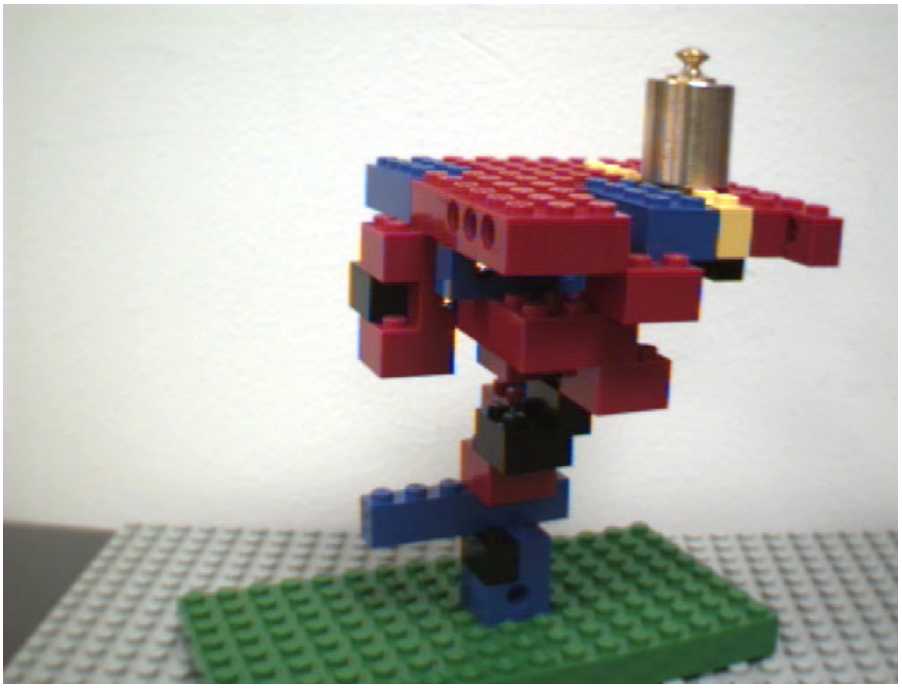

fig. 15. Lego table as specified by the diagram of fig. 5, holding a 50g weight.

7.2 Problems defining the fitness function

A first attempt to evolve a table failed to satisfy objective 2 (covering the entire surface). The reasons for this failure require further investigation. One problem with our representation is that the distance between genotype and phenotype is big, making most mutations too radical. Also, not providing 1×1 bricks complicates matters (but we did so because our current set of Lego does not include them). Finally, there is little selective pressure as fitness values between 1.9 and 2.0 are nearly identically selected. The raw value in the range [1, 5] was expanded by an exponential function to add selective pressure (so for example the fitness value of 123.74 in fig. 5 corresponds to a raw fitness of 4.8), but this did not solve the problem of full coverage. For the successful run pictured above the coverage objective was redefined as “covering at least 96% of the target area”.

The use of stepped fitness functions might not be ideal; Pareto optimality aware GA techniques (Goldberg, 1989, ch. 5) should improve performance in multiobjective problems such as this one.

8 Problems and future research directions

The algorithm being used to solve our models does not find all possible solutions. More sophisticated algorithmic tools may provide ways to fully solve the system of equations. The use of appropriate heuristics may in any case help a good deal. This is a critical factor as we may be wasting many plausible structures just because we are not capable of proving their stability.

The tree representation for Lego structures is a limiting factor. An improved description should bring genotype and phenotype closer, providing a better ground for evolution of objects of higher complexity.

The crossover operator provides a primitive way to reuse successful parts, modules, that may spread over the population. A better representation combined with modular recombination tools (Angeline and Pollack, 1994) could allow composite block structures — such as the bricklayers pattern which holds increased stress — to be discovered and replicated as new basic components.

We believe that we can reach some understanding of the dynamic stresses which would be involved in basic Lego mechanisms driven by Lego motors. This would open the field for evolving active pieces of machinery, including vehicles.

The use of more advanced evolutionary techniques including multiobjective optimization, speciation, automatic functional decomposition or landscape models (Goldberg, 1989; Darwen, 1996), may improve over the performance of our minimal steady-state GA.

9 Conclusions

We have shown that under some constraints, a simulator for objects can be used in an evolutionary computation, and then the objects can be built. Our belief is that in machine learning/evolving systems, the more interesting results, such as Sims' creatures or expert backgammon players (Tesauro, 1995; Pollack *et al.*, 1996), are due more to features of the learning environment than to any sophistication in the learning algorithm itself. By keeping inductive biases and *ad hoc* ingredients to a minimum, we have also demonstrated that interesting real-world behavior can come from a simple virtual model of physics and a basic adaptive algorithm.

The use of modular building elements with predictable — within an error margin — properties allows evolutionary algorithms to manipulate physical entities in simulation in ways similar to what we have seen, for example, in the case of robot control software. The bits in our artificial chromosomes are not limited to codifying just bits; they are capable of representing the building blocks of an entire physical structure.

We believe to have only scratched the surface of what is achievable. Combined with suitable simulators, the recombination of modular components guided by an artificial selection algorithm is a powerful framework capable of designing complex architectures ready to be built and used.

References

Angeline, P. J. and Pollack, J. B. (1994). Coevolving High-Level Representations. In C. Langton, (ed.) *Proceedings of the Third Artificial Life Meeting*.

Chapman, C. D., Saitou, K. and Jakiela, M. J. (1993) Genetic Algorithms as an Approach to Configuration and Topology Design, in *Proceedings of the 1993 Design Automation Conference*, DE-Vol. 65-1. Published by the A.S.M.E., Albuquerque, New Mexico, p. 485-498.

Cliff, D., Harvey, I., Husbands, P. (1996). Artificial Evolution of Visual Control Systems for Robots. *From Living Eyes to Seeing Machines.* M. Srinivisan and S. Venkatesh (eds.), Oxford University Press.

Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1989). *Introduction to Algorithms*. MIT press - McGraw Hill.

Darwen, P. J. (1996) *Co-evolutionary Learning by Automatic Modularisation with Speciation*. University of New South Wales, 1996.

Floreano, D. and Mondada, F. (1994). Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural Network Driven Robot. In D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson (Eds.), *From Animals to Animats III*, Cambridge, MA. MIT Press.

Forbus, K. (1984). Qualitative process theory. In *Artificial Intelligence* 24, 85-168.

Funes, P. and Pollack, J. (1997). Computer Evolution of Buildable Objects. *Fourth European Conference on Artificial Life.* P. Husbands and I. Harvey, eds., MIT Press. pp 358-367.

Gardin, F. and Meltzer, B. (1989). Analogical Representations of Naive Physics. *Artificial Life* 38, pp 139-159.

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

Iusem, A. and Zenios, S. (1995). Interval Underrelaxed Bregman's method with an application. In *Optimization*, vol. 35, iss. 3, p. 227.

Jakobi, N., Husbands, P. and Harvey, I. (1995). Noise and the Reality Gap: The use of Simulation in Evolutionary Robotics, in *Advances in Artificial Life: Proceedings of the 3rd European Conference on Artificial Life*, Moran, F., Moreno, A., Merelo, J., Chacon, P. (eds.) Springer-Verlag, Lecture Notes in Artificial Intelligence 929. pp. 704-720.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press.

Lee, W., Hallam, J. and Lund, H. (1996). A Hybrid GP/GA Approach for Co-evolving Controllers and Robot Bodies to Achieve Fitness-Specified Tasks. In *Proceedings of IEEE 3rd International Conference on Evolutionary Computation*. IEEE Press.

Leighton, T., Makedon, F., Plotkin, S., Stein, C., Tardos, E. and Tragoudas, S. (1995). Fast Approximation Algorithms for Muticommodity Flow Problems. *Journal of Computer and Syst. Sciences* 50. p. 228-243.

Lund, H., (1995). Evolving Robot Control Systems. In J. T. Alander (ed.) *Proceedings of 1NWGA*, University of Vaasa, Vaasa.

Lund, H., Hallam, J and Lee, W. (1997). Evolving Robot Morphology. Invited paper in *Proceedings of IEEE Fourth International Conference on Evolutionary Computation.* IEEE Press, NJ.

Mataric, M and Cliff, D. (1996). Challenges In Evolving Controllers for Physical Robots. In *Evolutional Robotics*, special issue of *Robotics and Autonomous Systems*, Vol. 19, No. 1. pp 67-83.

Ngo, J.T., and Marks, J. (1993). Spacetime Constraints Revisited. In *Computer Graphics,* Annual Conference Series. p. 335-342.

Pollack, J. B., Blair, A. and Land, M.(1996). Coevolution of A Backgammon Player. *Proceedings Artificial Life V,* C. Langton, (Ed), MIT Press.

Shoenauer, M. (1996). Shape Representations and Evolution Schemes. In L. J. Fogel, P. J. Angeline and T. Back, Editors, *Proceedings of the 5th Annual Conference on Evolutionary Programming*, MIT Press, to appear.

Sims, K. (1994). Evolving Virtual Creatures. In *Computer Graphics,* Annual Conference Series.

Sims, K. (1994b). Evolving 3D Morphology and Behavior by Competition. In *Artificial Life IV Proceedings*, MIT Press.

Tesauro, G. (1995) Temporal difference learning and TD-Gammon. *Communications of the ACM,* 38(3): 58-68.

Zienkiewicz, O.C. (1977). *The Finite Element Method in Engineering Science.* McGraw-Hill, New York, 3rd edition.