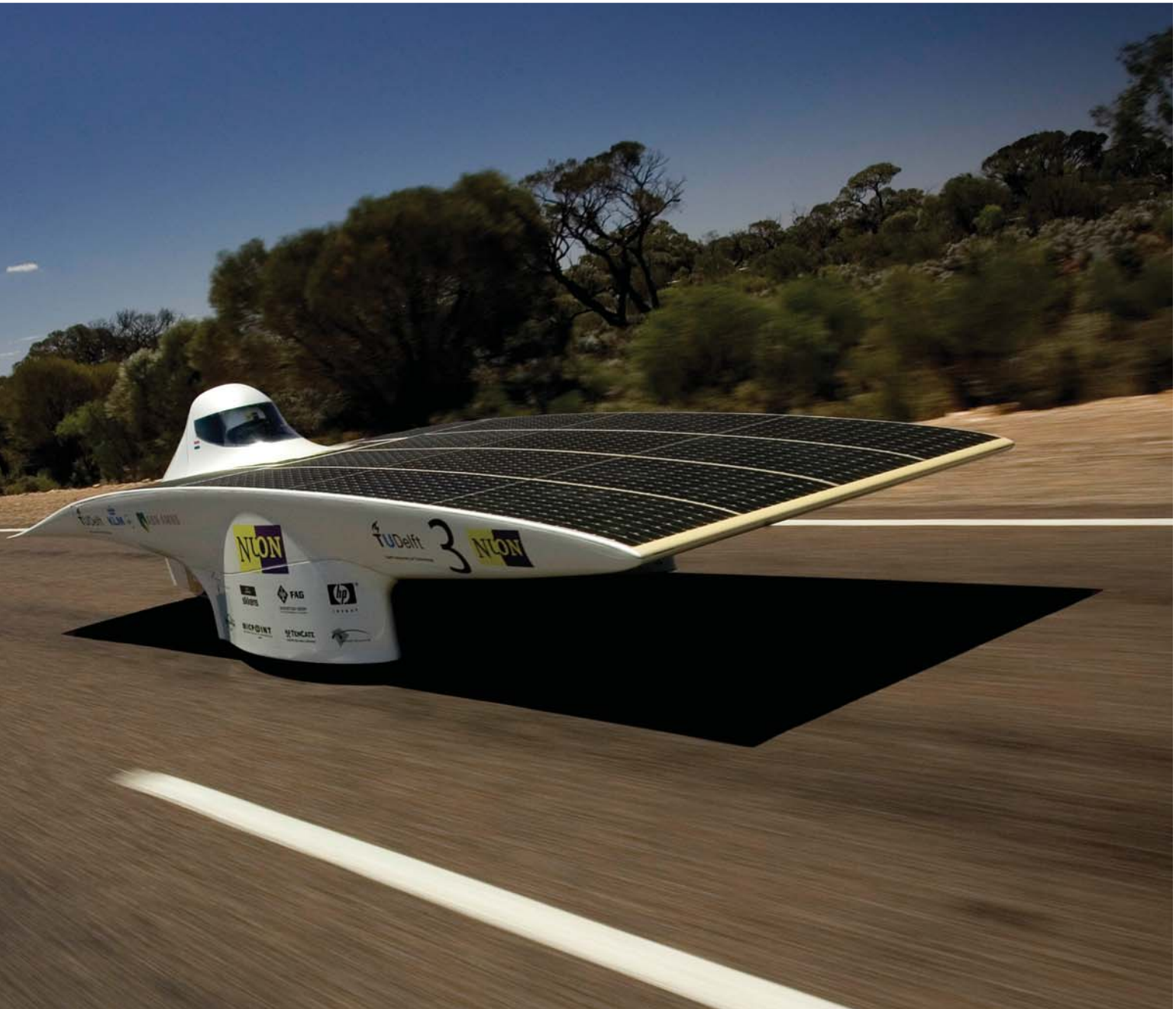
SIGEVOlution

newsletter of the ACM Special Interest Group on Genetic and Evolutionary Computation

Autumn 2007
Volume 2 Issue 3



in this issue

Editorial

A solar car hisses through the beautiful Australian desert; its racing strategy has been evolved using a genetic algorithm. Meanwhile, in a university laboratory, Lego structures are evolved using Genetic Programming. Indeed, this new issue of SIGEVOlution shows some rather unusual applications of evolutionary computation and I hope you will enjoy reading it!

In the first paper, Vincent de Geus and the Nuon team from the Delft University of Technology tell us how they applied a genetic algorithm to optimize the racing strategy for their solar car leading them to win the World Solar Challenge. In the second paper, Pablo Funes shows us how we can evolve Lego structures using Genetic Programming while, in the third and last paper, Stewart Wilson provides a gentle introduction to accuracy-based classifier systems. The subsequent columns provide information about a recently discussed PhD thesis, the new issues of EC journals, and the calendar of EC events.

The cover photo is by Hans-Peter van Velthoven. More photos of the Nuon team are available here, while photos of the other competing teams are available from the World Solar Challenge homepage.

This issue is brought to you with the help of Tine Lavrysen, Vincent de Geus, Pablo Funes, Stewart W. Wilson, Janusz Wojtusiak, and board members Dave Davis and Martin Pelikan.

And remember that SIGEVOlution needs you! So, if you have suggestions or criticisms that may improve the newsletter, just drop an email to editor@sigevolution.org.

Pier Luca
April 8th, 2008

SIGEVOlution
Autumn 2007, Volume 2, Issue 3

Newsletter of the ACM Special Interest Group on Genetic and Evolutionary Computation.

Contributors to this Issue

Vincent de Geus
Pablo Funes
Stewart W. Wilson

Contents

ISSN: 1931-8499

World Solar Challenge: the Race Strategy Explained

Vincent de Geus & the 2007 Nuon Solar Team, Delft University of Technology, The Netherlands, info@nuonsolarteam.com

In the summer of 2006 a new team consisting of students from the Delft University of Technology was formed to continue the unprecedented success of the Nuon Solar Team. The team's members represent the largest faculties of the TU Delft, these include: Aerospace, Mechanical, Maritime and Electrical Engineering as well as the faculty of Industrial Design Engineering. Besides technical prowess, the team's members also display organizational, creative, entrepreneurial, and practical skills that built on the unparalleled successes already achieved. The new team began work in September 2006 on a completely new design for a solar vehicle able to compete in the new "Challenge" class of the Panasonic World Solar Challenge, the premier solar car race in the world. Besides competing in this event, the team attend events around the European Union to promote innovation and continue the drive towards sustainability. On October 25th 2007, they crossed the finishline first in Adelaide, Australia, and became winners of the Panasonic World Solar Challenge. This was the fourth time in a row the Nuon Solar Team from Delft University of Technology won this prestigious race through the Australian continent.

The Challenge

The idea for the World Solar Challenge originates from Danish-born adventurer Hans Tholstrup. As early as the 1980s, he became aware of the need to explore sustainable energy as a replacement for limited available fossil fuel. Sponsored by BP, he designed the world's first solar car, called Quiet Achiever, and traversed the 4052 km between Sydney and Perth in 20 days. In 1987 the first World Solar Challenge took place, and since 2001 four Nunas (Nuon teams) have been first over the finish line.

Since solar cars rely only on solar power, one is forced to handle this energy in the most effective way. That is why a good energy management system, or strategy, is required. The optimization task is to minimize the race time while respecting the boundary conditions and constraints imposed by the system and the race regulations. The problem can then for example be solved using classical optimal control theory, as was done by the Australian team Aurora (Pudney, 2000).

The Nuna approach however, has always been different. In 2001 it was implemented for the first time (Trottemant, 2002) using genetic algorithm optimizer (Goldberg, 1989) techniques. Genetic algorithms are known for their good performance at optimizing complex problems with many parameters and a simple objective. In addition, in 2005 a smart adaptive cruise control was implemented with the use of Model-Predictive Control techniques (Boom and Back, 2004).

Problem definition

Each race day starts at 0800 and ends at 1700. After 1700 and before 0800 the teams are not allowed to drive anymore, but they may do 'static loading'. This means that the solar panel is held perpendicular to the sun for maximum efficiency. During the race there are 7 control points where the teams have to stop for half an hour before continuing. In this timespan, it's also possible to do 'static loading'. Since the 2007 World Solar Challenge, a rest day was added in Alice Springs. Most of the teams were able to fully recharge their batteries so that the race was actually split into two stages. This made the optimization problem less complex, but it added some tense situations because the batteries could now be drained twice. One or two hours before the end of a race day, an expected endpoint was briefed to the meteo car which was driving 100km in front, so that they could find a good place for a campsite, with enough 'static loading' possibilities. The coordinates of the campsite were sent back to the Mission Control, and now the optimization task was to arrive at this campsite as close as possible to 1700. Arriving too early would give a loss of valuable race time, whilst arriving after 1710 would result in penalty minutes.

During the optimization we have the following constraints: The battery state of charge is bounded between the maximum energy storage the batteries can contain, and a minimum value, which is a safety margin. The velocity is also bounded, because the race is driven on public roads, where local traffic rules apply.

Optimization strategy

In this section, we describe the optimization tools which were used to calculate an optimal average velocity during the race. The optimization used a database which was filled with GPS data for the whole track (3000km), giving the road direction and inclination every 100 meters. Also important was the place and time dependent weather database which combined weather information from more than 20 weather stations along the race route. This database was kept up to date by calling airports and downloading the latest report during the race (satellite phone).

Dynamic programming techniques (Bertsekas, 2005) are used to divide the problem into overlapping subproblems. Longterm strategy deals with the whole race but is less detailed than the midterm strategy, which only optimizes over 1 day. The longterm optimization problem is defined as a time optimization and ends when the whole trajectory is accomplished. The end distance is thus fixed, the start distance is not, because the optimization can be started at any point in the race. The parameter that is optimized is the total race time. This is done with respect to the velocity of the car. To make the set of possible solutions relatively small, the race trajectory is divided into intervals with almost equal properties (direction, inclination, longitude, latitude). The velocity is assumed to be constant for the whole interval. The genetic algorithm now starts building a population of members, where each member consists of the velocity of every interval. This member is inserted into the mathematical model, which is not further discussed in this article, and this results in a total race time and the number of times the battery state of charge bounds are hit. Every solution that causes the batteries to drain (hitting the lower bound) gets a penalty, and therefore this solution will lose the evolutionary struggle against a member that doesn't hit those bounds. Also, solutions that arrive at the finish line early have a better fitness than slower solutions, and therefore they stay alive and produce even better solutions in their offspring.

Exactly the same happens with the midterm strategy, but there are several differences. First, the optimizer only considers one day, but the interval is now 15 minutes. Therefore this optimizer is much more detailed than the longterm optimizer. The reason this can't be done with the longterm strategy is that there would then simply be too many possible solutions, which reduces the chance of finding the optimal one. Another important difference is the parameter that is optimized; for the midterm strategy it is the distance, with a fixed battery state of charge as a constraint. This battery state of charge is obtained from the longterm strategy and defines a power budget for the day. If this battery state of charge is reached by the end of the day, one can be sure to reach the finish line without having to stop due to empty batteries.

Bibliography

[1] D. P. Bertsekas. Dynamic Programming and Optimal Control. Athena Scientific, 3rd edition, 2005.

[2] T. Van Den Boom and T. Back. Course notes model predictive control. Technical report, Dutch Institute for Systems and Control, 2004.

[3] D. E. Goldberg. Genetic Algorithms in Search Optimization and Machine Learning. Addison Wesley, 1989.

[4] P. Pudney. Optimal energy management for solarpowered cars. PhD thesis, University of South Australia, 2000.

[5] D. M. Roche, A. E. T. Schinckel, J. W. V. Storey, C. P. Humphris, and M. R. Guelden. Speed of Light The 1996 World Solar Challenge. Photovoltaics Special Research Center University of New South Wales, 1997.

[6] E. J. Trottemant. Alpha centauris strategy during the wsc 2001. Technical report, European Space Agency, 2002

About the authors

The members of the 2007 Nuon Solar Team are: Stefan Roest (Team-leader), Tine Lavrysen (Public Relations & Ergonomics), Demian de Ruijter (Chief Engineer), Ivo Hagemans (Production & Logistics), Joep Steenbeek (Production & Ergonomics), Susan Luijten (Aerodynamics), Oliver van der Meer (Aerodynamics & Finance), Paul Beckers (Electronics & Telemetry), Vincent de Geus (Electronics & Strategy), Rabih Al Zaher (Mechanics), Hjalmar Van Raemdonck (Structural Design), and Gert Kraaij (Mechanics). The photos are by Hans-Peter van Velthoven, more photos of the Nuon team are available here.

Homepage: http://www.nuonsolarteam.com
Email: info@nuonsolarteam.com

Buildable Evolution

Pablo José Funes, Icosystem Corporation, Cambridge (MA), US, pablo@icosystem.com

The most interesting results in Artificial Life come about when some aspect of reality is captured. In the mid-1990s, Karl Sims energized the AL community with his ground-breaking work on evolved moving creatures [28, 29]. The life-like behavior of Sims' creatures resulted from combining evolved morphology with a physics simulation based on Featherstone's earlier work [9].

The question that begged asking was: can a similar thing be done in the physical world? Can we make creatures that walk out of the computer screen and into the room?

Two components were required: a language to evolve morphologies that have real-world counterparts, and a way to build them — either in simulation or by automated building and testing. We set out to demonstrate that buildable evolution was possible using a readily available, cheap building system — Lego bricks – and an ad-hoc physics simulation that allowed us to study the interaction of the object with the physical world *in silico*; with respect to gravitational forces at least. The result [10, 14, 12, 13, 15, 16, 25, 23, 26, 24, 27] is a system that can evolve a variety of different shapes and is very easy to use, set up and replicate.

Here I present an overview of the evolvable Lego structures project. Coinciding with the publication of this article, the source code is being released to the community (demo.cs.brandeis.edu/pr/buildable/source).

1 Evolving Toy Brick Structures

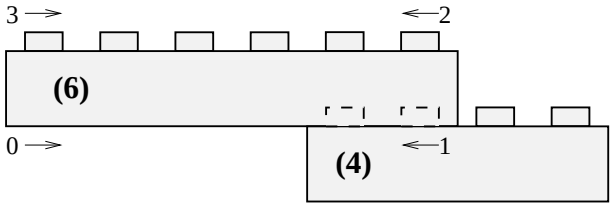
With Genetic Programming (GP) Koza introduced the notion of evolving expressions using their parsing trees [19]. GP's breakthrough was to *evolve* trees using simple operations: recombinate by cutting and pasting subtrees, mutate by changing a node's properties. The present work could be considered an *embodiment* of GP. Intuitively, it makes sense to think of Lego building as trees: to start, you hold the first brick, then grab a second one, and attach it to the first. The third brick will be attached to either of the previous ones, possibly both, and so on.

Here Lego bricks are encoded by a tree data structure where each node represents one brick. There are two versions: one for two-dimensional structures only [13] and a later one that works for 3D as well [15, 16]. "2D" Lego structures are flat, made with bricks of width 1 and different lengths.

In 2D a node has an integer parameter to define the length of the brick and four branches (Fig. 1). Branches correspond to attachment loci (lower left, lower right, upper left, upper right). The $shift$ parameter encodes the number of knobs a descendant "bites" into the parent. Fig. 2 shows an example. The 2D encoding was extended to cover 3D structures and bricks wider than 1 (Fig. 3). The 3D encoding dropped the notion of "attachment loci" in favor of an arbitrary list of descendants with (x, y, z, θ) coordinates to describe the position and rotation of attached bricks relative to the parent (Fig. 4).

brick ::= ($size$, joint, joint, joint, joint)
joint ::= nil | ($shift$, brick)
$size \in \{4,6,8,10,12,16\}^{\dagger}$
$shift \in 1..6$

Fig. 1: Grammar for 2D structures. †Available brick sizes depend on the brick set used by the experiment.

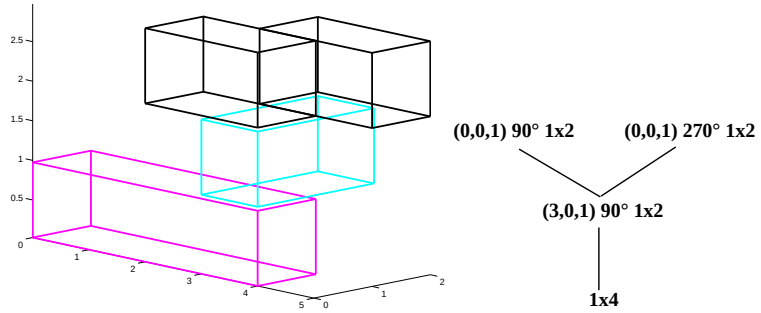


(6 nil (2 (4 nil nil nil)) nil nil)

Fig. 2: Example of 2D genetic encoding of bricks and corresponding structure. The 6×1 brick is the root and the 4×1 brick is attached at site 1, "biting" 2 knobs.

brick ::= ($n \times m$, (jointlist))
jointlist ::= $\emptyset$ | joint jointlist
joint ::= $((x,y,z,\theta)$ brick)
$n \times m \in \{2\times 1, 4\times 1, 2\times 2, 4\times 2, 6\times 1, 8\times 1, 10\times 1, 12\times 1\}^{\dagger}$
$x \in 1..n$, $y \in 1..m$, $z \in \{-1,1\}$, $\theta \in \{0^o, 90^o, 180^o, 270^o\}$

Fig. 3: Grammar for 3D structures. †Available brick sizes set by the experiment.



$$(1\times 4\,(((3,0,1,90^o)\,(1\times 2(((0,0,1,90^o)$$
$$(1\times 2\,\text{nil})(((1,0,1,270^o)\,(1\times 2\,\text{nil}))))))))$$

Fig. 4: 3D encoding example in Lisp form and tree form, and corresponding bricks structure.

1.1 Mutation and Crossover

Mutation is easy to define with the encoding just described. Choosing a brick at random and changing the brick size parameters (n,m) replaces it with a brick of a different size. Perturbing the parameters (x,y,z,θ) that define the position of the brick with respect to its parent results in a new position for the brick – and its descendants along with it. Even though this is sufficient to evolve many structures, later on we added a longer list of mutations aimed at modifying the structure in small, meaningful steps (see table 1). In order to do crossover, take two parent trees A and B and select random nodes a and b on each one. Remove a and insert b in its place. Crossover is fundamental here, as the only form of replication of parts and components. Without crossover the EA had a major drop in performance ([11] § 2.5).

1.2 Development

The result of either crossover or mutation is a new structure tree. Both operations are capable of producing *invalid* trees — trees with overlapping bricks or that violate other constraints (total number of bricks available, spatial bounds and so on). The tree is developed and pruned in order to reduce it to one representing a *valid* tree.

Shrink	Choose a random node and decrease the brick size down to the next valid size. Also, randomly choose to either let descendants slide in, following the change in size or keep descendants in the same absolute position.
Grow	Choose a random node and increase the brick size on either end. Randomly choose to move descendants along or keep their absolute positions.
Shift	Choose a random joint and modify its position by shifting it one step in the front, back, left or right direction, or turning 90^o. Randomly choose to keep descendants in place or have them move along.
Add	Choose a random node, add a new brick at a random position.

Tab. 1: List of mutations

The newly formed tree is visited in left-to-right, depth-first-search order. Before placing each brick, constraints are checked. If there is another brick there already, or the number of bricks already equals the maximum allotted, or if one of the bricks' ends falls outside of the bounds set by the experiment, then that brick cannot be placed. The corresponding node is replaced with NIL.

The following mutation of Fig. 4, for example, is invalid because the third brick (z mutated to -1) is now below the second one, but the first brick is already there.

$$(1\times 4(((3,0,1,90^o)(1\times 2(((0,0,-1,90^o)(1\times 2\,\text{nil})(((1,0,1,270^o)(1\times 2\,\text{nil}))))))))) \quad (1)$$

The tree is pruned and three bricks remain:

$$(1\times 4\,(((3,0,1,90^o)\,(1\times 2\,(((1,0,1,270^o)\,(1\times 2\,\text{nil})))))))) \quad (2)$$

This procedure insures a one-to-one correspondence between nodes in the tree and bricks in the structure. The development process inherent to brick structures allows us remove "introns" — unused parts of the genome — to obtain a lean encoding. *Bloat*, the abundance of unreachable subexpressions, is a well known problem. In GP there is no general-purpose method to know when a subexpression is unused and therefore, bloat-reducing methods must rely on heuristics instead [8].

Interestingly, bloat is still a potential problem even when the lean encoding is used, since structures frequently have bricks that do not fulfill any function. Occasionally such bloat can be useful as a form of random exploration or genetic drift that has the potential of achieving new solutions. Additionally, since recursion is not allowed in the encoding, nor any introns, sometimes non-functional "limbs" evolve that serve as a sort of genetic repository for an evolving family of structures.

A side-effect of this test-and-prune method is that often there is a reduction in the total number of bricks. As a consequence, a "remove random brick" mutation is not usually included (Table 1).

Although there is a development process associated with our representation of Lego bricks, it cannot be called a developmental representation because there is no recurrence. Reuse of elements is granted here by the reliance on crossover. Crossover generates repetitions when a copy of a subtree is spliced into a branch of itself and most of our evolved structures end up having repeated patterns.

1.3 Testing

The final step in the mutation/crossover procedure is calling the simulator to verify that the new structure can hold its own weight. If this is the case then the mutation/crossover is successful, and we proceed to evaluate the fitness.

1.4 Evolutionary Algorithm

We use a plain steady-state Evolutionary Algorithm (EA) (Table 2). Typically the first individual consists of just one brick, which guarantees that it can hold its own weight. The population size is usually 1000. We deliberately stayed away from optimizing the evolutionary algorithm too much — the point of the work being that buildable structures can be evolved and deployed, not how fast it can be done.

1.5 Buildable vs. Incrementally Buildable

The development described above is almost a building process. If the structure was to be built by adding one brick after the other, in the order specified by the tree, two things could still go wrong: the structure might not hold its own weight in some of the intermediate configurations, and also, if there's a brick above and another brick below the one you are

```
1: Create random individual I
2: Initialize population P = {I}
3: while best fitness < target fitness do
4:     Randomly select mutation or crossover
5:     Select 1 (for mutation) or 2 (for crossover) random individual(s)
           with fitness proportional probability
6:     Apply mutation or crossover operator
7:     Prune
8:     Compute gravitational stresses
9:     if (the new model can support its own weight) then
10:        if (|P|=Population Size) then
11:            Remove a random individual from P, chosen with inverse
                   fitness proportional probability
12:        end if
13:        Add the new model to P
14:    end if
15: end while
```

Tab. 2: Simple steady-state EA

trying to add, one of them has to be removed before you can do the insertion. Adding those two additional constraints would mean that the encoding represents not only the final structure, but also an algorithm for building it. We are tempted to call this a *constructable* structure.

This idea is intriguing; for example: designing a bridge to be deployed over a river is one thing, but a different one is planning a bridge that can be built entirely from one side, adding successive bricks to complete the span. You cannot cross the bridge, but you can always walk over the partial bridge to add another block.

2 Stability of Lego Structures

Lacking an automated procedure for building and testing brick structures in hardware, we had to rely on simulations to produce objects that can hold their own weight. Our simulator, although it is a very simplified version of reality, led to a computationally difficult problem for which a fully satisfactory solution is yet to be found.

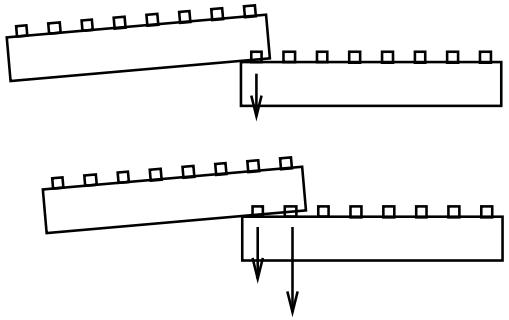


Fig. 5: Fulcrum effect: two joined bricks are easy to break apart when the point of contact is used as a fulcrum. This effect led to simulating the union as a pin joint subject to torque (cf. Table 3).

It makes intuitive sense to focus a simulation for Lego bricks under gravitational stresses on what happens at the joints between bricks. When you are trying to separate two bricks, it is hard to do so by simply pulling apart in a direction perpendicular to the plane of contact. It is much easier to snap them by using the bricks as levers (Fig. 5). The fulcrum effect led to the idea of thinking of brick unions as planar “pin” joints. We pretend the structure is a mesh of rigid bars held to each other by pins with relatively large static friction coefficients (Fig. 6). Each brick corresponds to a bar, and each area of contact to a pin. When the torque at any one pin exceeds its friction coefficient, the joint turns, breaking the structure.

With this assumption all the forces in the model became torques. Each joint has a “capacity”, implying that it can bear up to a certain amount of torque without budging. Exactly how much torque is something we can measure (Table 3). When evolving structures the gravitational constant was set to 1.2 to add a margin of safety.

From a structure formed by a combination of bricks, our model builds a network with joints of different capacities. Each load has a site of application in one node — each brick’s weight is a force applied to itself; external forces also enter the structure through one brick — and has to be canceled by one or more reaction forces for that brick to be stable. Reaction forces can come from any of the joints that connect it to neighbor bricks. But the brick exerting a reaction force becomes unstable and has to be stabilized in turn by a reaction from a third brick. The load thus “flows” from one brick to the other. Following this principle, a load is propagated through the network until finally absorbed by the ground.

Joint size (ω) knobs	Approximate torque capacity (κ_ω) N-m$\times 10^{-3}$
1	12.7
2	61.5
3	109.8
4	192.7
5	345.0
6	424.0
$n > 6$ †	$\geq 424n/6$

Tab. 3: Measured torque capacities of linear Lego joints. †A conservative estimate used for longer joints.

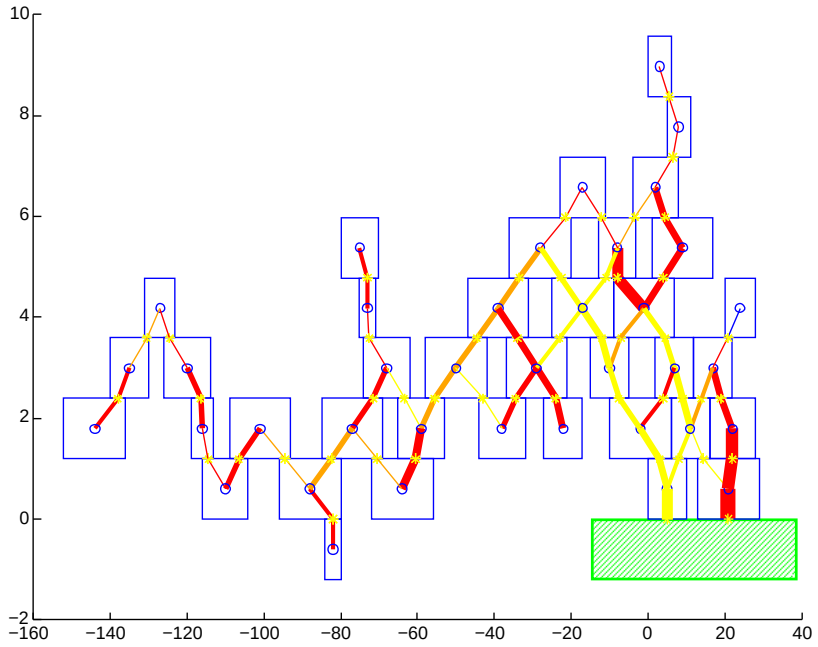


Fig. 6: Model of a 2D Lego structure showing the brick outlines (rectangles), centers of mass (circles), joints (diagonal lines, with axis located at the star), and "ground" where the structure is attached (shaded area). The thickness of the joint's lines is proportional to the strength of the joint. A load distribution was calculated for all joints. The ones nearing their capacity are in yellow, less stressed ones red.

2.1 Networks of Torque Propagation

The principle of propagation of forces described generates a set of simultaneous equations, a "Network of Torque Propagation" (NTP). A solution of the NTP describes a way that loads can distribute among the network of bricks such that no joint is stressed beyond its maximum capacity, and so the structure does not break.

Let F be the force corresponding to one of the loads in the structure. For an arbitrary joint j in the structure, if F had to be supported by j its magnitude as a torque acting on j would be proportional to the length of the arm. The fraction α of F that j can take is

$$\alpha_{j,F} = \max\left\{1, \left|\frac{K_j}{d(j,F)F_\perp}\right|\right\} \tag{3}$$

where K_j is the maximum capacity of the joint, $d(j,F)$ the length of the load arm (distance from the point of application to j) and $F_\perp$ the component of F perpendicular to the arm.

If the torque $\tau = d(j,F)F_\perp$ generated is less than the joint maximum K, then $\alpha = 1$ (i.e. the joint fully supports F); otherwise $\alpha = K/\tau$. Note that $F_\perp$ can have a positive or negative sign depending on whether the torque acts clockwise or counterclockwise.

The problem of whether the stresses generated by F can be propagated through the structure is equivalent to the well-known Network Flow Problem (NFP) [7]. Nodes in the NFP correspond to bricks in the structure; the *source* is the brick to which the force is applied, the *sink* is the ground, and each joint is a vertex between two nodes with capacity $\alpha_{j,F}$. A *maximum flow* ϕ valued 1 corresponds to a valid distribution of the force F throughout the structure such that no joint exceeds its capacity. If the maximum flow is less than one, its value is the fraction of F that the structure can hold without breaking.

When more than one force is involved, a solution for the NTP problem can be described as a set $\{\phi_F\}$ of network flows valued one, one for each force. As multiple forces acting on a joint add to each other, the *combined* torques must be equal to or less than the capacity of the joint, thus adding the additional constraint

$$\left|\sum_F \phi_F(j)d(j,F)F_\perp\right| \leq K_j \tag{4}$$

This multiple force NTP is equivalent to a more difficult problem, the *Multicommodity Network Flow Problem* (MNFP)[1, ch. 17].

2.2 NTP Algorithms

The NFP problem has well known polynomial-time solutions [7] but MNFPs are much harder, and fall into the general category of Linear Programming (LP). There is a fair amount of research on the multicommodity problem [17, 2, 18, 21] but these algorithms are still orders of magnitude slower than the NFP case.

2.2.1 Greedy Algorithm

A possible approach to solving NTPs is a greedy algorithm: take the first force F_1. If a solution to the corresponding NFP can be found then use the corresponding flow to compute a *residual capacity* K' for all joints (eq. 5). The residual capacity represents how much extra force can each joint take in addition to the stress induced by the first force.

$$K'_j = K_j - \phi_F(j)\delta(j,F)||F|| \tag{5}$$

A new NFP for the second force with respect to the residual capacities can now be computed and solved. Iterating through all forces, if successful, yields a valid set of flows compliant with eq. 4. To solve for a single force we first tried a naive algorithm (compute recursively for each joint the percentage of F it can support) until we found the NFP approach. Later we used the PRF algorithm [5].

Greedy algorithms miss some solutions but are quick, and for many experiments were enough to evolve good solutions.

2.2.2 LP Solver

A second approach to NTPs is to use an MNFP-specific algorithm. PPRN [4] was tried. This algorithm can always find the solution — if there is one — but, as other LP algorithms, takes exponential time in the worst case. In practice, evolving using PPRN turned out to to be slower than using the greedy solver, approximately by a factor of 10.

2.2.3 Embedded Solver

It is somewhat paradoxical that a study advocating evolutionary computation would run into difficulties when dealing with classic search algorithms such as LP. Can we not solve the NTP with a GA? Furthermore, if some of the computation used to solve the loads of a structure could be inherited by its descendants, it could be a major source of optimization.

The embedded solver is an attempt to do just that - instead of nested search algorithms (one evolving the locations of the bricks, an internal one solving the structure's equations) we designed an EA with two kinds of mutations: mutations for changing bricks' parameters and also mutations to distribute flows of loads within the structure. The genotype was extended to encode the position of the bricks and additionally the flow for each force from a brick to the next. So not only the layout of the structure is encoded in the genotype, but also the proof of its stability. For additional details on this approach, see [11].

The embedded solver was used successfully in several experiments. It is faster than the greedy solver; however, some results obtained with the greedy and LP solvers, notably the "long bridge" of Section 4.1 could not be replicated. We still think that an embedded solver is an intriguing idea that requires further investigation.

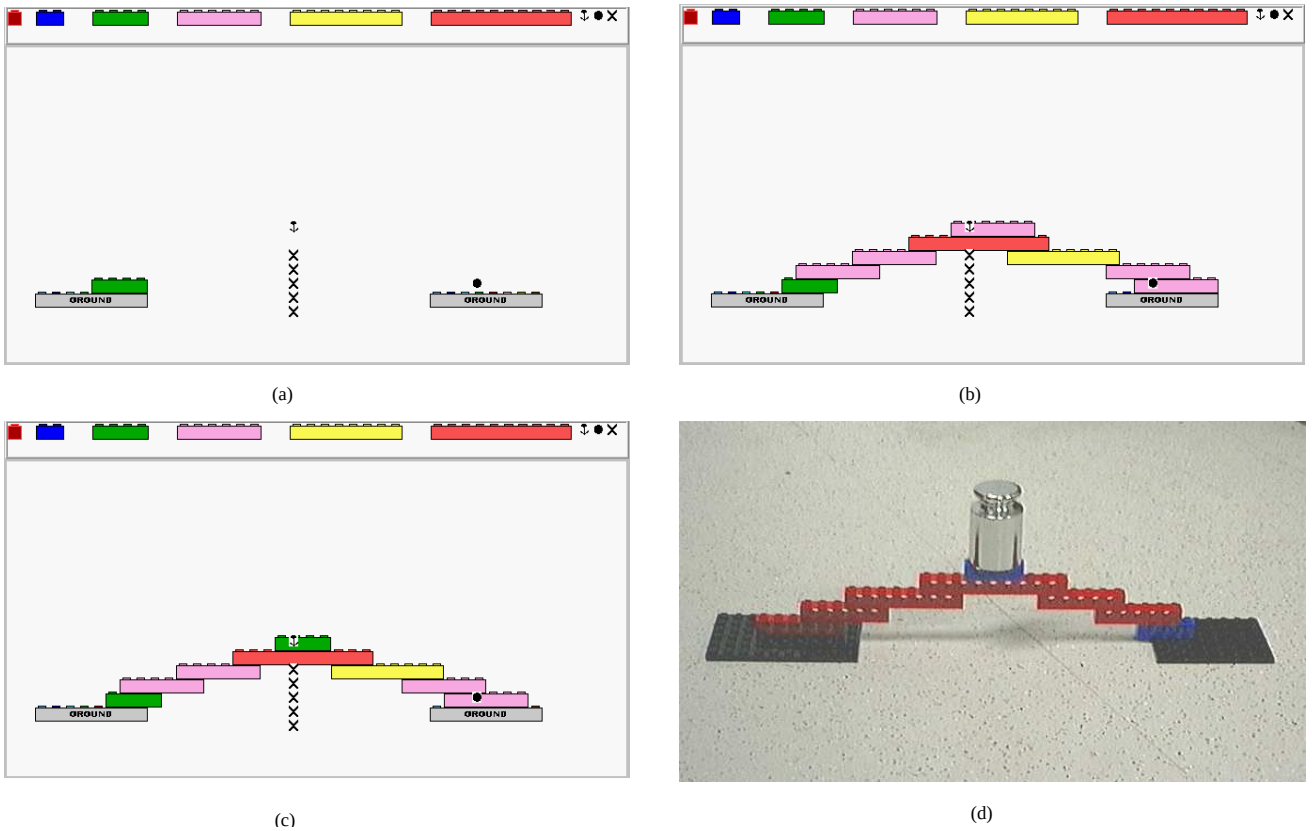
3 Evolving Simple Structures with EvoCAD

EvoCAD is a toy CAD system to design simple 2D Lego structures, testing their gravitational resistance and evolve them (Fig. 7). EvoCAD allows the user to set up two kinds of goals: *target points* are points the structure should touch and *loads* are external loads that the structure should support. Additionally, *restrictions* can be set (points that the structure cannot touch). At least one *ground* is required, and one or more initial bricks. The fitness function is inferred from target and load points:

$$\sum_{t \in \text{targets}} \frac{1}{1+d(S,t)} + \sum_{l \in \text{loads}} \frac{1}{1+d(S,l)} + \sum_{\substack{l \in \text{loads} \\ d(S,l)=0}} \text{supp}(S,l) \tag{6}$$

(where *d* is the distance between a target/load point and the structure S, and *supp* the fraction (maximum flow) of a certain load that the structure supports).

When the evolve button is pressed, the population is seeded with the current structure and evolved until all objectives are satisfied, or a timeout (20 seconds) is reached. The result is shown to the user who can make modifications to the design, press the "test" button for testing the structure, update objectives and evolve again; the new structure seeds the next round. EvoCAD uses the embedded solver for evolving speedy solutions in just a few seconds and the LP solver for the manual "test structure" button.



(a) (b) (c) (d)

Fig. 7: Sample working session with the EvoCAD program: (a) The user has defined two grounds and several evolutionary hints: restrictions (×), target (●) and load (↓); an initial brick was laid down. (b) Evolution designed a structure that fulfills all requirements. (c) The user made cosmetic corrections (d) The structure has been built with Lego bricks.

Bricks	{4,6,8,10,12,16}
Max bricks	127
Base	(0,-1)...(-39,-1)
x, y domain	(-310,41)×(-2, 80)
Target Point T	(-300,0)
Fitness	$1 - \frac{d(S,T)}{d(0,T)}$
Solver	Greedy/Recursive

Tab. 4: Long bridge problem specification

EvoCAD demonstrates the potential of Evolutionary Design [3] for a new kind of collaborative applications where the CAD system is not only an intelligent canvas, but proposes solutions of its own. Similar to what happens in Interactive Evolutionary Computation [20, 30], here the feedback loop leads to solutions that neither can do on their own.

4 Evolved Structures

4.1 Long Bridge

The idea for this experiment is to see how long a beam can be evolved, supported on a fixed Lego table, that supports its own weight. The experimental design defines a ground, 40 knobs in length, supporting the structure, and a fitness function which is the distance between structure and a faraway target point at (-300,0) (Table 4).

The EA was run for several days, until it no longer seemed to be coming up with additional improvements. The resulting structure was longer than we imagined, a cantilevered beam made entirely from Lego pieces that spans 1.70m (Figs. 9 and 8)

Interestingly, part of the solution is a smaller beam going up in the opposite direction that serves as counterbalance, alleviating part of the load on the bricks that hold the bulk of the structure.

4.2 Crane

The "crane" experiment had a more complicated setup. Here the aim was to design the arm for a crane that could lift a weight. The rotating base of the crane was designed by us; the evolved structure must attach to it via the "predefined bricks" included in the experiment (Fig. 11).

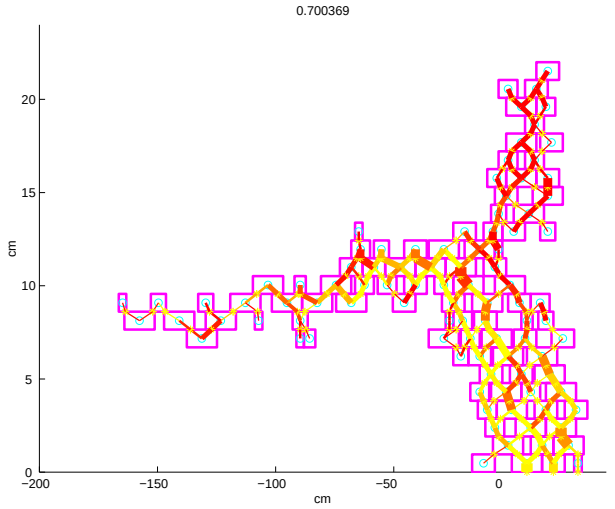


Fig. 8: Long Bridge Scheme. The network represents the distribution of loads found by the solver; thicker lines correspond to stronger joints. Yellow joints are stressed to the max, whereas red ones are not.

A crane base was manually designed with motors for rotating the arm and pulling the hook. The evolved crane attached to it using 5 "predefined bricks". The fitness value is the horizontal length x of the arm, but if the maximum load M supported at the tip is less than 250 g then x is multiplied by $M/250$, thus reducing the fitness (Table 5). The role of crossover is clearly visible in Fig. 12 where a counterbalance structure evolved first to help supporting the load and later replicated — a triangular shape appeared by chance, but was able to better support the crane's load, becoming part of the final design.

4.3 Tree

The "tree" experiment was designed to test out whether some structural characteristics of life forms (branching, symmetry) could evolve as a consequence of the environment. The design of a tree in nature is a product of conflicting objectives: maximizing the exposure to light while keeping internal stability.

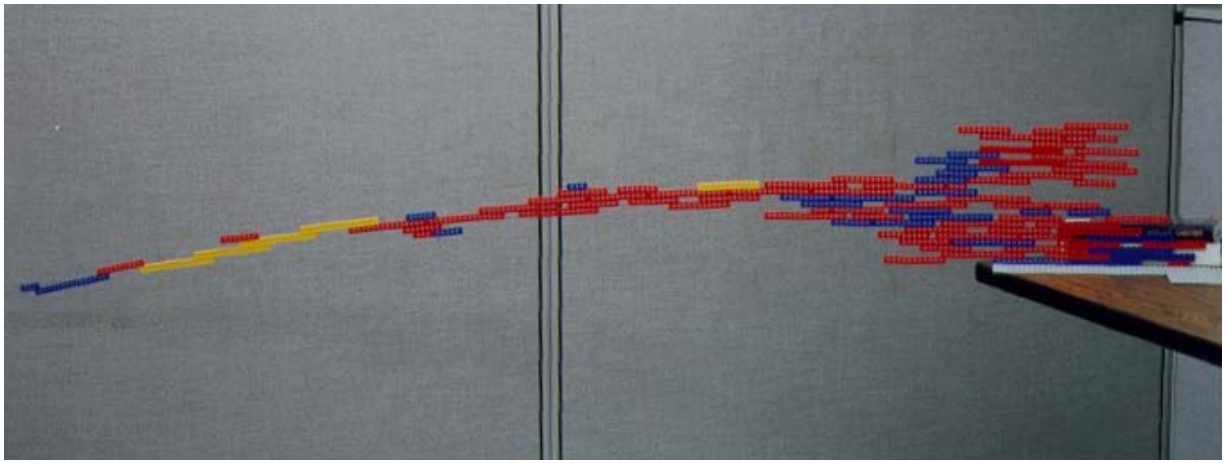


Fig. 9: Long Bridge.

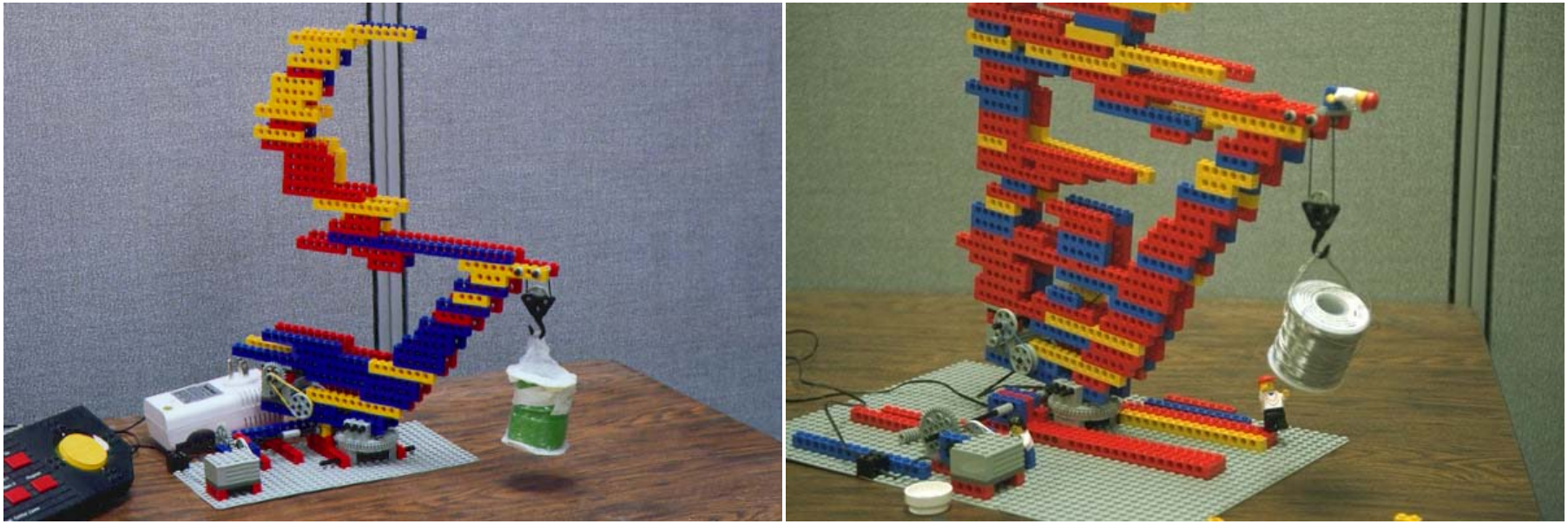


Fig. 10: Crane with a diagonal crane arm: intermediate (left) and final (right) stages.

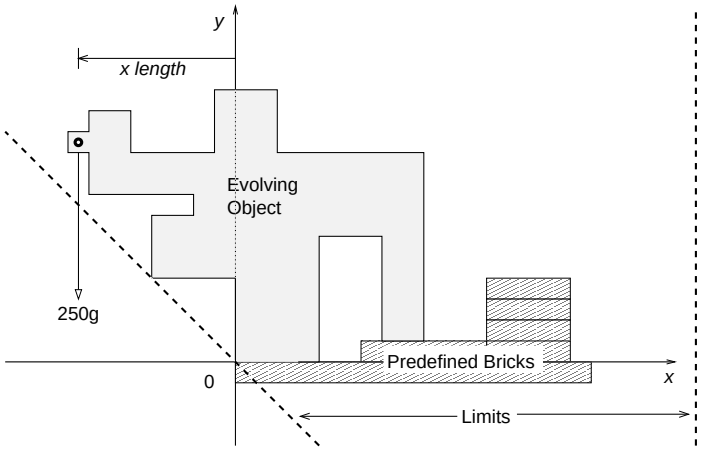


Fig. 11: Crane arm experiment

Bricks	{4,6,8,10,12,16}
Max Bricks	127
Base	(0,-1)…(-16,-1)
x, y domain	(-50,22)×(-1,40): $y > -x$
Fixed Bricks	5 (see fig. 11)
Solver	Greedy (Recursive)
Fitness	$1 + (-x)supp(L)$
x	position of leftmost brick
$supp(L)$	fraction of 250g load supported

Tab. 5: Setup of the crane experiment.

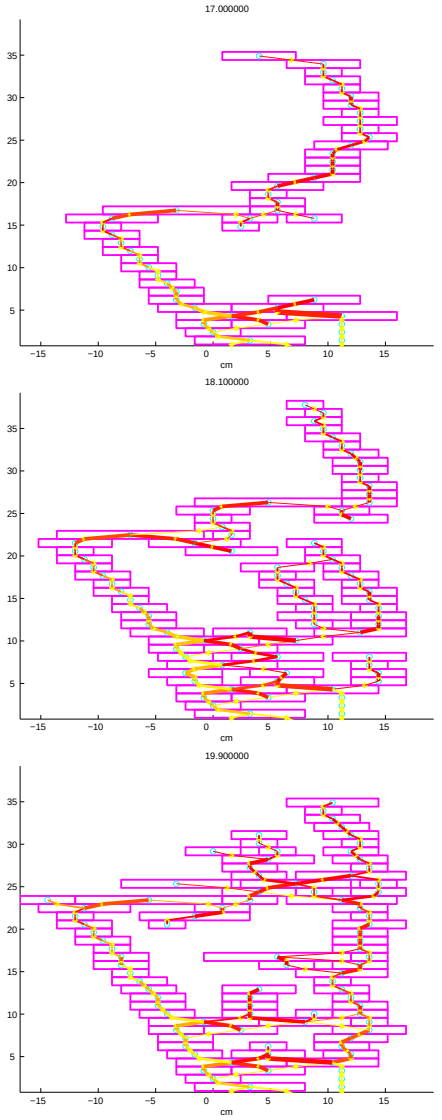


Fig. 12: Three stages on the evolution of crane show frequent reuse of subparts by means of crossover.

Bricks	{1,2,4,6,8,10,12,16}
Max Bricks	127
Base	(0,-1)–(2,-1)
x, y domain	(-50,52)×(0,45)
Solver	Embedded
Fitness	$f_L + f_R + f_T$
f_L	$f_L = \sum_{j=0}^{45} \max(0, -\min\{x : (x,j) \in S\})$
f_R	$f_R = \sum_{j=0}^{45} \max(0, \max\{x : (x,j) \in S\})$
f_T	$f_T = \sum_{i=-50}^{52} \max(0, \max\{y : (i,y) \in S\})$

Tab. 6: Setup of the tree experiment. The fitness components f_L, f_R, f_T represent "light" coming from the left, right and top (Fig. 13)

The experimental design for the Lego tree has a narrow attachment base: only three knobs. This provides very little support for cantilevering, so the structure needs to be balanced to reach out. The evolutionary goal of the structure is to maximize exposure to light by reaching high and wide. This simulates how real trees compete for light by reaching high and having a larger surface (Table 6 and Fig. 13).

There were no symmetry-oriented operators in our experiments, as could be, for example a "reverse" recombination operator that switched the orientation of a subpart. This means that symmetry is not encouraged by representational biases. Instead, the problem setup requires balancing the total weight of both sides. The tree did evolve, however, with a central symmetry with branches reaching out, by evolving the same solution independently on both sides.

The general layout of the evolved tree has several similarities with that of a real tree: there is a (somewhat twisted) trunk, with branches that become thinner as they reach out, and "leaves", bulky formations that maximize the surface at the end of the branch.

4.4 Table

To demonstrate the 3D version of our algorithm a setup somewhat similar to the tree experiment was used. Here the idea is to *design* a "table" with a flat surface (Table 7).

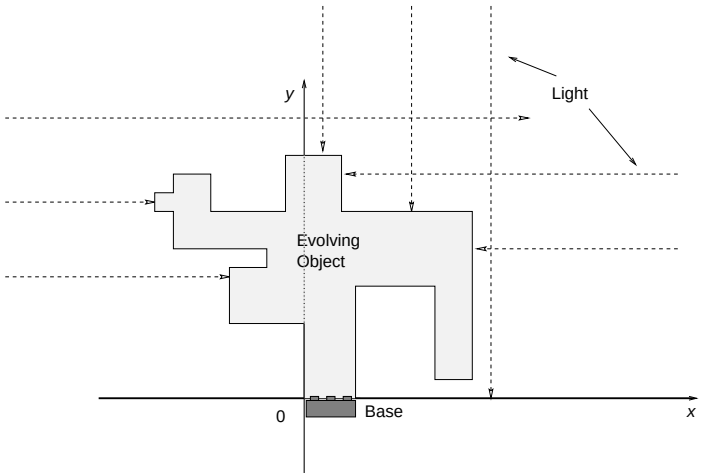


Fig. 13: Tree experiment

Bricks	$1\times1, 2\times1, 2\times2, 3\times1, 3\times2, 4\times1, 4\times2$ $6\times1, 8\times1, 10\times1, 12\times1$
Max Bricks	150
Base	$[-2\ldots2]\times[-2\ldots2]\times[-1]$
x, y, z domain	$[-9\ldots10]\times[-9\ldots10]\times[0\ldots15]$
Solver	Embedded
G	5†
Fitness	$\prod_{x,y} f(x,y) + p$
$f(x,y)$	$\frac{9+height(x,y)}{160}$
$height(x,y)$	$1 + \max\{z : (x,y,z) \in S\}$
p	$1 - \frac{mass(S)}{8 \cdot 150}$

Tab. 7: Setup of the table experiment. The fitness, somewhat convoluted, is based on height of the structure at each valid (x,y) plus a premium p for lightness. †The gravitational constant was set to 5 times the normal value to encourage a strong design.

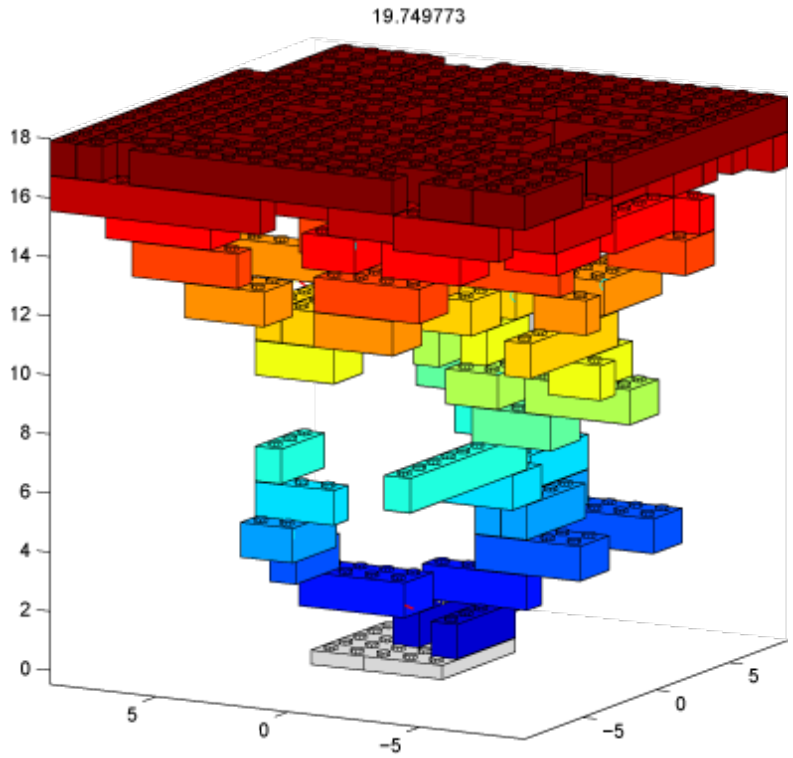


Fig. 14: Evolved table.

The fitness function rewards a structure that reaches as high as possible on every column, so the table grew high, hitting the roof (maximum z value allowed) to create a flat surface. A premium for small mass was added. The final result (Fig. 14) is a nice design, but reveals some limitations of our EA: there are some holes as the surface was not fully covered; and there are some bricks that have no apparent function. The second problem should be addressed with a multiobjective EA; the first is an interesting open question: is the EA capable of efficiently finding a tiling coverage of a surface?

5 Discussion

Evolution of robots and other physical entities is a very active field today; automated fabrication of Lego structures has not been achieved; however, rapid prototyping machines are capable of "printing" evolved structures straight out of an evolutionary algorithm. There are also some initial examples of self-assembly, where modular components latch on to each other as a result of random motion or robotic search (see [22] for an overview). Evolving Lego structures remains a cheap system that does not require expensive machinery nor electronics, only a little patience to put together the results.

Our simulator remains an ad hoc construction that would benefit from a more standard engineering approach. At the same time, the idea of embedding the solver in the representation (the embedded solver of §2.2.3) is powerful and deserves a more thorough investigation. As for the system itself, extending the functionality of the EA with the addition of a multiobjective algorithm [6] is a must, as there is usually more than one objective involved. Typically, there are at least the objectives of supporting one or more loads and minimizing the number of bricks in the structure at the same time.

Instead of devising an expert system with rules about how to divide a task into subtasks, and how to carry along with each of those, Evolutionary Design as shown here relies on lower-lever knowledge. The rules of physics, unlike the rules of design, are not an artificial creation, and this leads to novel, surprising solutions. The evolutionary algorithm explores design space in ways which are not so strongly pre-determined by our culture, and so the resulting objects have an alien look. We believe that useful inventions, no matter how weird they might look in the beginning, are eventually incorporated into the culture if they are useful. Just as today we trust our lives to an airplane (which at first glance seems incapable of flight), tomorrow we may walk over bridges designed by an evolutionary algorithm.

Acknowledgments

This research was carried out at the DEMO lab in Brandeis University under the supervision of Jordan B. Pollack (www.demo.cs.brandeis.edu).

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, Englewood Cliffs, 1993.

[2] A. Ali, R. V. Helgason, J. L. Kennington, and H. Lall. Computational comparison among three multicommodity network flow algorithms. *Operations Research*, 28:995–1000, 1980.

[3] P. Bentley, editor. *Evolutionary Design by Computers*. Morgan-Kaufmann, San Francisco, 1999.

[4] J. Castro and N. Nabona. An implementation of linear and nonlinear multicommodity network flows. *European Journal of Operational Research*, 92:37–53, 1996.

[5] B. V. Cherkassky and A. V. Goldberg. On implementing push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.

[6] C. A. Coello Coello. Metaheuristics for multiobjective optimization. a genetic multiobjective optimization tutorial. IEEE Swarm Intelligence Symposium, 2003.

[7] T. H. Cormen, C. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press - McGraw Hill, 1989.

[8] E. De Jong and J. B. Pollack. Multi-objective methods for tree size control. Genetic Programming and Evolvable Machines 4, 211-233, 2003.

[9] R. Featherstone. *Robot Dynamics Algorithms*. Kluwer Academic Publishers, Norwell, MA, 1987.

[10] P. Funes. *Evolution of Complexity in Real-World Domains*. PhD thesis, Brandeis University, Waltham, Mass., May 2001.

[11] P. Funes. Evolution of complexity in real-world domains. Brandeis University Dept. of Computer Science PhD Dissertation, 2001.

[12] P. Funes, L. B. Lapat, and J. B. Pollack. EvoCAD: Evolution-assisted design. In *Artificial Intelligence in Design'00 (Poster Abstracts)*, pages 21–24. Key Centre of Design Computing and Cognition, University of Sidney, 2000.

[13] P. Funes and J. B. Pollack. Computer evolution of buildable objects. In P. Husbands and I. Harvey, editors, *Fourth European Conference on Artificial Life*, pages 358–367. MIT Press, Cambridge, 1997.

[14] P. Funes and J. B. Pollack. Componential structural simulator. Technical Report CS-98-198, Brandeis University Department of Computer Science, 1998.

[15] P. Funes and J. B. Pollack. Evolutionary body building: Adaptive physical designs for robots. *Artificial Life*, 4(4):337–357, 1998.

[16] P. Funes and J. B. Pollack. Computer evolution of buildable objects. In P. Bentley, editor, *Evolutionary Design by Computers*, pages 387 – 403. Morgan-Kaufmann, San Francisco, 1999.

[17] M. D. Grigoriadis and L. G. Khachiyan. An exponential-function reduction method for block-angular convex programs. *Networks*, 26:59–68, 1995.

[18] A. Iusem and S. Zenios. Interval underrelaxed Bregman's method with an application. *Optimization*, 35(3):227, 1995.

[19] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, 1992.

[20] K.Sims. Artificial evolution for computer graphics. In *Computer Graphics (Siggraph '91 proceedings)*, pages 319–328, 1991.

[21] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. Fast approximation algorithms for muticommodity flow problems. *Journal of Computer and Systems Sciences*, 50:228–243, 1995.

[22] H. Lipson. Evolutionary robotics and open-ended design automation. In B. Cohen, editor, *Biomimetics*. CRC Press, 2005.

[23] J. B. Pollack, L. Hod, H. Gregory, and F. Pablo. Three generations of automatically designed robots. *Artificial Life*, 7(3):215–223, Summer 2001.

[24] J. B. Pollack, G. S. Hornby, H. Lipson, and P. Funes. Computer creativity in the automatic design of robots. *Leonardo*, 36(2):115–121, 2003.

[25] J. B. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby, and R. Watson. Evolutionary techniques in physical robotics. In J. Miller, editor, *Evolvable Systems: from biology to hardware*, number 1801 in Lecture Notes in Computer Science, pages 175–186. Springer-Verlag, 2000.

[26] J. B. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby, and R. A. Watson. Evolutionary techniques in physical robotics. In P. J. Bentley and D. W. Corne, editors, *Creative Evolutionary Systems*, pages 511–523. Morgan Kaufmann, 2001.

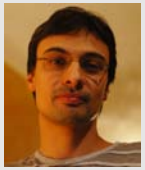
[27] J. B. Pollack, H. Lipson, P. Funes, S. G. Ficici, and G. Hornby. Coevolutionary robotics. In J. R. Koza, A. Stoica, D. Keymeulen, and J. Lohn, editors, *The First NASA/DoD Workshop on Evolvable Hardware*. IEEE Press, 1999.

[28] K. Sims. Evolving 3D morphology and behavior by competition. In R. Brooks and P. Maes, editors, *Artificial Life IV*, pages 28–39. MIT Press, 1994.

[29] K. Sims. Evolving virtual creatures. In *Computer Graphics*. Annual Conference Series, 1994.

[30] H. Takagi. Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89:1275–1296, 2001.

About the author



Pablo Funes. Born Córdoba (Argentina) 1966. Married (to Silvia Gerszkowicz, 2004), 2 children. Founded MAPA Systems, a small-business software company (1990). Graduated in Mathematics (University of Buenos Aires, 1994) while programming computers for fun and to make a living. After a stint as an socio-economic impact modeler/project analyst for the Argentinian government (1994-1995, Ministry of Economics) moved to the US to pursue a Computer Science PhD. Joined J.B. Pollack's DEMO lab in Brandeis University (PhD completed 2001). His work focused on tapping what he calls the "reality effect" - combining artificial evolution and real-world environments. His work in Evolutionary Design (using toy bricks, Lego) and in Collective Interactive Evolution (interactive online video game agents, Tron) are examples. Joined Icosystem Corporation of Cambridge, MA in 2001 where he became Director of Research in 2004. There, his work has focused on Interactive Evolution (e.g. of swarm behaviors, baby names, business simulations, etc.), Evolutionary Design (web caching algorithms, business strategies) and Collective Interactive Evolution (Postal routes), Social Networks and Text Visualization.

Homepage: http://www.icosystem.com
Email: pablo@icosystem.com

What Is a Classifier System?

A gentle introduction to accuracy-based classifier systems such as XCS

Stewart W. Wilson, Prediction Dynamics, USA, wilson@prediction-dynamics.com

A learning classifier system (LCS) is an adaptive system that learns to perform the best action given its input. By "best" is generally meant the action that will receive the most reward or reinforcement from the system's environment. By "input" is meant the environment as sensed by the system, usually a vector of numerical values. The set of available actions depends on the system context: if the system is a mobile robot, the available actions may be physical: "turn left", "turn right", etc. In a classification context, the available actions may be "yes", "no", or "benign", "malignant", etc. In a decision context, for instance a financial one, the actions might be "buy", "sell", etc. In general, an LCS is a simple model of an intelligent agent interacting with an environment.

An LCS is "adaptive" in the sense that its ability to choose the best action improves with experience. The source of the improvement is reinforcement—technically, *payoff*—provided by the environment. In many cases, the payoff is arranged by the experimenter or trainer of the LCS. For instance, in a classification context, the payoff may be 1.0 for "correct" and 0.0 for "incorrect". In a robotic context, the payoff could be a number representing the change in distance to a recharging source, with more desirable changes (getting closer) represented by larger positive numbers, etc. Often, systems can be set up so that effective reinforcement is provided automatically, for instance via a distance sensor. Payoff received for a given action is used by the LCS to alter the likelihood of taking that action, in those circumstances, in the future. To understand how this works, it is necessary to describe some of the LCS mechanics.

Inside the LCS is a set—technically, a *population*—of "condition-action rules" called *classifiers*. There may be hundreds of classifiers in the population. When a particular input occurs, the LCS forms a so-called *match set* of classifiers whose conditions are satisfied by that input. Technically, a condition is a *truth function* $t(x)$ which is satisfied for certain input vectors x. For instance, in a certain classifier, it may be that $t(x) = 1$ (true) for $43 < x_3 < 54$, where x_3 is a component of x, and represents, say, the age of a medical patient. In general, a classifier's condition will refer to more than one of the input components, usually all of them. If a classifier's condition is satisfied, i.e. its $t(x) = 1$, then that classifier joins the match set and influences the system's action decision. In a sense, the match set consists of classifiers in the population that *recognize* the current input.

Among the classifiers—the condition-action rules—of the match set will be some that advocate one of the possible actions, some that advocate another of the actions, and so forth. Besides advocating an action, a classifier will also contain a *prediction* of the amount of payoff which, speaking loosely, "it thinks" will be received if the system takes that action. How can the LCS decide which action to take? Clearly, it should pick the action that is likely to receive the highest payoff, but with all the classifiers making (in general) different predictions, how can it decide? The technique adopted is to compute, for each action, an average of the predictions of the classifiers advocating that action—and then choose the action with the largest average. The prediction average is in fact weighted by another classifier quantity, its *fitness*, which will be described later but is intended to reflect the reliability of the classifier's prediction.

The LCS takes the action with the largest average prediction, and in response the environment returns some amount of payoff. If it is in a learning mode, the LCS will use this payoff, P, to alter the predictions of the responsible classifiers, namely those advocating the chosen action; they form what is called the *action set*. In this adjustment, each action set classifier's prediction p is changed mathematically to bring it slightly closer to P, with the aim of increasing its accuracy. Besides its prediction, each classifier maintains an estimate ε of the error of its predictions. Like p, ε is adjusted on each learning encounter with the environment by moving ε slightly closer to the current absolute error $|p - P|$. Finally, a quantity called the classifier's *fitness* is adjusted by moving it closer to an inverse function of ε, which can be regarded as measuring the accuracy of the classifier. The result of these adjustments will hopefully be to improve the classifier's prediction and to derive a measure—the fitness—that indicates its accuracy.

The adaptivity of the LCS is not, however, limited to adjusting classifier predictions. At a deeper level, the system treats the classifiers as an evolving population in which accurate—i.e. high fitness—classifiers are reproduced over less accurate ones and the "offspring" are modified by genetic operators such as mutation and crossover. In this way, the population of classifiers gradually changes over time, that is, it adapts *structurally*. Evolution of the population is the key to high performance since the accuracy of predictions depends closely on the classifier conditions, which are changed by evolution.

Evolution takes place in the background as the system is interacting with its environment. Each time an action set is formed, there is finite chance that a genetic algorithm will occur in the set. Specifically, two classifiers are selected from the set with probabilities proportional to their fitnesses. The two are copied and the copies (offspring) may, with certain probabilities, be mutated and recombined ("crossed"). Mutation means changing, slightly, some quantity or aspect of the classifier condition; the action may also be changed to one of the other actions. Crossover means exchanging parts of the two classifiers. Then the offspring are inserted into the population and two classifiers are deleted to keep the population at a constant size. The new classifiers, in effect, compete with their parents, which are still (with high probability) in the population.

The effect of classifier evolution is to modify their conditions so as to increase the overall prediction accuracy of the population. This occurs because fitness is based on accuracy. In addition, however, the evolution leads to an increase in what can be called the "accurate generality" of the population. That is, classifier conditions evolve to be as general as possible without sacrificing accuracy. Here, general means maximizing the number of input vectors that the condition matches. The increase in generality results in the population needing fewer distinct classifiers to cover all inputs, which means (if identical classifiers are merged) that populations are smaller, and also that the knowledge contained in the population is more visible to humans—which is important in many applications. The specific mechanism by which generality increases is a major, if subtle, side-effect of the overall evolution.

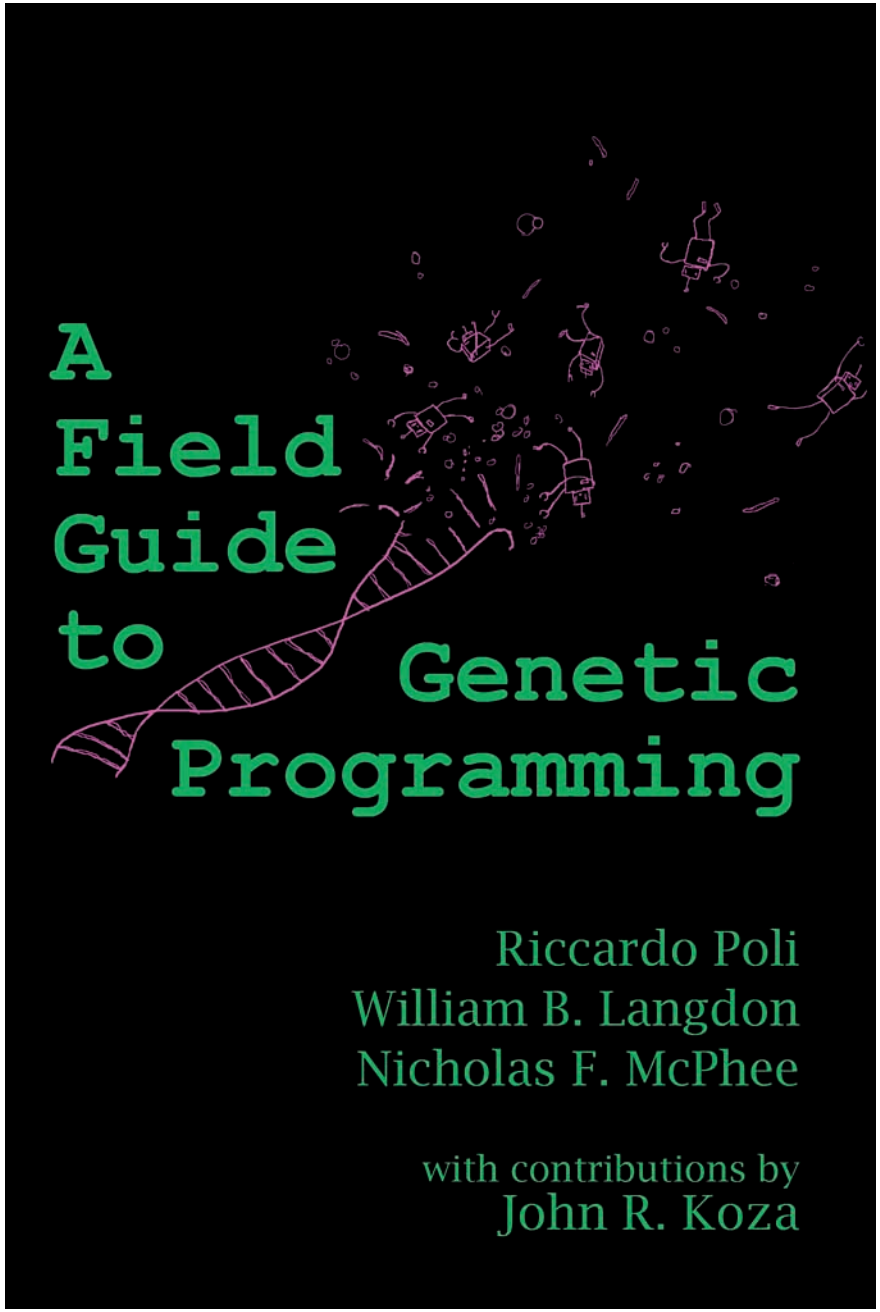
Summarizing, a learning classifier system is a broadly-applicable adaptive system that learns from external reinforcement and through an internal structural evolution derived from that reinforcement. In addition to adaptively increasing its performance, the LCS develops knowledge in the form of rules that respond to different aspects of the environment and capture environmental regularities through the generality of their conditions.

Many important aspects of LCS were omitted in the above presentation, including among others: use in sequential (multi-step) tasks, modifications for non-Markov (locally ambiguous) environments, learning in the presence of noise, incorporation of continuous-valued actions, learning of relational concepts, learning of hyper-heuristics, and use for on-line function approximation and clustering. An LCS appears to be a widely applicable cognitive/agent model that can act as a framework for a diversity of learning investigations and practical applications.

About the author

Stewart W. Wilson does research and consults on learning classifier systems. After receiving S.B. in physics and Ph.D. in electrical engineering degrees from MIT, he worked for some years with Edwin H. Land, inventor of the Polaroid camera, on an interactive teaching machine concept. About 1981 he chanced on John Holland's early writings on classifier systems and since that time has focused on them. He believes that the value of classifier systems as mental/agent models increases every year, together with their real-world applicability.

Homepage: http://www.prediction-dynamics.com
Email: wilson@prediction-dynamics.com



Free Book: A Field Guide To Genetic Programming

A Field Guide To Genetic Programming
Riccardo Poli, William B. Langdon, Nicholas F. McPhee (with contributions from John R. Koza)
ISBN 978-1-4092-0073-4, 250 pages, 6" x 9", softcover, (download free pdf!)

Within 24 hours of its sell out launch at EuroGP-2008 the free PDF of this 250 page book on genetic programming was down loaded more than 820 times from lulu.com

Printed copies are also available (for GBP 7.20 plus postage etc.) from lulu.com.

Visit www.gp-field-guide.org.uk for more information.

Dissertation Corner

Handling Constrained Optimization Problems and Using Constructive Induction to Improve Representation Spaces in Learnable Evolution Model

Doctoral Thesis by Janusz Wojtusiak

The learnable evolution model (LEM) is an evolutionary optimization method which uses machine learning to guide the evolution process (Michalski, 2000). At each step of evolution a machine learning program is applied to induce hypotheses why some candidate solutions perform better and others perform worse. These hypotheses are then instantiated in order to produce new candidate solutions.

This dissertation investigates two closely related problems in the learnable evolution model: the automatic improvement of representation spaces using constructive induction, and the handling of constraints in optimization problems. The former includes an investigation of different aspects of representation space transformations in the context of optimization problems, the development of algorithms that perform these transformations, and algorithms for creating new candidate solutions (via instantiation) in the improved representation spaces. Handling specific types of constraints is closely related to the instantiation task in the modified representation spaces; therefore, the same methods can be used for solving both problems. Moreover, transformations of representation spaces may help in handling constraints of other types, that is, constraints that cannot be handled directly during the instantiation process.

The most important contributions of this dissertation include:

- Classification of constraints into four classes based on the difficulty of handling them in the learnable evolution model. The most important distinction is made between instantiable and general constraints. This distinction is made by the presence of an efficient method for solving them in the instantiation process.
- Design and implementation of methods for handling instantiable constraints given in the form of ordered conditions [ATTR rel EXPR]. Although this type of constraint is very limited and few real world optimization problems may have constraints in this form, they are important for instantiation of conditions with constructed attributes. This is because the algorithm for constructing new attributes can be constrained to create only attributes in this form.
- Design and implementation of three methods for handling general constraints in the learnable evolution model. The methods are specifically designed to work with the learnable evolution model and are based on trimming rules hypothesized from high performing candidate solutions, approximation of the feasible area using machine learning, and using infeasible solutions as a contrast set for learning.
- Design and implementation of methods for automatically improving representation spaces in LEM. Two methods based on data-driven constructive induction are discussed in this dissertation. One of the methods constructs new attributes only in the instantiable form mentioned above, and the other constructs more general attributes.
- Design of methods for instantiating in the modified spaces. The methods are based on the fact that conditions that include constructed attributes can be treated as constraints.

The developed algorithms are implemented in the LEM3 (Wojtusiak and Michalski, 2006) and AQ21 (Wojtusiak et al., 2006) systems and tested on a set of constrained and non-constrained benchmark optimization problems. Additionally, two real world applications are presented in this dissertation. The first one concerns optimization of parameters of complex systems, and the second concerns finding the best discretization of numeric attributes.

Bibliography

[1] Michalski, R. S., "LEARNABLE EVOLUTION MODEL Evolutionary Processes Guided by Machine Learning," Machine Learning, 38, pp. 9-40, 2000.

[2] Wojtusiak, J. and Michalski, R. S., "The LEM3 Implementation of Learnable Evolution Model and Its Testing on Complex Function Optimization Problems," Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2006, Seattle, WA, July 8-12, 2006.

[3] Wojtusiak, J., Michalski, R. S., Kaufman, K. and Pietrzykowski, J., "The AQ21 Natural Induction Program for Pattern Discovery: Initial Version and its Novel Features," Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence, Washington D.C., November 13-15, 2006.

Janusz Wojtusiak received his M.Sc. in Computer Science from Jagiellonian University and Ph.D. in Computational Sciences and Informatics (concentration in Computational Intelligence and Knowledge Mining) from George Mason University. Janusz is as post-doctoral researcher at the George Mason University Department of Health Administration and Policy. He also serves as a director of the GMU Machine Learning and Inference Laboratory. Janusz's research interests include health informatics, machine learning, evolutionary computation, knowledge mining, and related fields. In particular, his work concerns applications of the above in health care research.

Email: jwojt@mli.gmu.edu
Homepage: www.mli.gmu.edu/jwojt

New Issues of Journals

Genetic Programming and Evolvable Machines 9(1) (www)

- **Editorial introduction**, Wolfgang Banzhaf, pp 1-2 (pdf)
- **Acknowledgment**, pp 3-4 (pdf)
- **Environmental effects on the coevolution of pursuit and evasion strategies** Joc Cing Tay, Cheun Hou Tng and Chee Siong Chan, pp 5-37 (pdf)
- **Use of genetic programming to diagnose venous thromboembolism in the emergency department** Milo Engoren and Jeffrey A. Kline, pp 39-51 (pdf)
- **Sporadic model building for efficiency enhancement of the hierarchical BOA**, Martin Pelikan, Kumara Sastry and David E. Goldberg, pp 53-84 (pdf)
- **A genetic algorithm for discrete tomography reconstruction**, Cesare Valenti, pp 85-96 (pdf)

Genetic Programming and Evolvable Machines 9(2) (www)

- **Theoretical foundations of evolutionary computation**, Xiaodong Li, Wenjian Luo and Xin Yao, pp 107-108 (pdf)
- **Quotients of Markov chains and asymptotic properties of the stationary distribution of the Markov chain associated to an evolutionary algorithm**, Boris Mitavskiy, Jonathan E. Rowe, Alden Wright and Lothar M. Schmitt, pp 109-123 (pdf)
- **Detecting the epistatic structure of generalized embedded landscape**, Shude Zhou, Robert B. Heckendorn and Zengqi Sun, pp 125-155 (pdf)
- **Evolutionary dynamics for the spatial Moran process**, P. A. Whigham and Grant Dick, pp 157-170 (pdf)

Swarm Intelligence 1(2) (www)

- **Moving targets: collective decisions and flexible choices in house-hunting ants**, Nigel R. Franks, James W. Hooper, Mike Gumn, Tamsyn H. Bridger, James A. R. Marshall, Roderich Groß and Anna Dornhaus, pp 81-94 (pdf)
- **Ant-based and swarm-based clustering**, Julia Handl and Bernd Meyer, pp 95-113 (pdf)
- **An ant colony optimization approach to flexible protein–ligand docking**, Oliver Korb, Thomas Stützle and Thomas E. Exner, pp 115-134 (pdf)
- **Ant colony optimization for real-world vehicle routing problems: From theory to applications**, A. E. Rizzoli, R. Montemanni, E. Lucibello and L. M. Gambardella, pp 135-151, (pdf)

Calls and Calendar

May 2008

Genetic Programming Theory and Practice 2008

May 15-17 (Thur-Sat), 2008 Ann Arbor Michigan USA

Homepage: WWW

GPTP is a small, one-track, invitation-only workshop devoted to the integration of theory and practice. In particular, it focuses on how theory can inform practice and what practice reveals about theory. Past workshops have invited speakers to discuss theoretical work and its value to practitioners of the art, and to review problems and observations from practice that challenge existing theory.

June 2008

2008 IEEE World Congress on Computational Intelligence

June 1-6, 2008, Hong Kong

Homepage: WWW

The 2008 IEEE World Congress on Computational Intelligence (WCCI 2008) will be held at the Hong Kong Convention and Exhibition Centre during June 1-6, 2008. Sponsored by the IEEE Computational Intelligence Society, co-sponsored by the International Neural Network Society, Evolutionary Programming Society and the Institution of Engineering and Technology, WCCI 2008 is composed of the 2008 International Joint Conference on Neural Networks (IJCNN 2008), the 2008 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2008) and the 2008 IEEE Congress on Evolutionary Computation (CEC 2008). WCCI 2008 will be the fifth milestone in this series with a glorious history from WCCI 1994 in Orlando, WCCI 1998 in Anchorage, WCCI 2002 in Honolulu, to WCCI 2006 in Vancouver.

July 2008

GECCO 2008 - Genetic and Evolutionary Computation Conference

July 12-16, 2008, Atlanta, Georgia, USA

Homepage: http://www.sigevo.org/gecco-2008

Conference Program

July 12	Pre-conference free workshops and tutorials
July 13	Pre-conference free workshops and tutorials; in the evening, opening reception
July 14	Presentations: reviewed papers, late breaking papers, Evolutionary Computation in Practice, competitions
July 15	Presentations: reviewed papers, late breaking papers, Evolutionary Computation in Practice, competitions; in the evening, poster session and reception
July 15	Poster Session and reception
July 16	Presentations: reviewed papers, late breaking papers, Evolutionary Computation in Practice, competitions

September 2008

PPSN 2008 - Parallel Problem Solving from Nature

September 13-17, 2008, Dortmund, Germany

Homepage: http://www.ppsn2008.org/

Call for paper: download

Deadline April 14, 2008

PPSN X will showcase a wide range of topics in Natural Computing including, but not restricted to: Evolutionary Computation, Quantum Computation, Molecular Computation, Neural Computation, Artificial Life, Swarm Intelligence, Artificial Ant Systems, Artificial Immune Systems, Self-Organizing Systems, Emergent Behaviors, and Applications to Real-World Problems.

Paper Presentation

Following the now well-established tradition of PPSN conferences, all accepted papers will be presented during small poster sessions of about 16 papers. Each session will contain papers from a wide variety of topics, and will begin by a plenary quick overview of all papers in that session by a major researcher in the field. Past experiences have shown that such presentation format led to more interactions between participants and to a deeper understanding of the papers. All accepted papers will be published in the Proceedings.

Paper Submission

Researchers are invited to submit original work in the field of natural computing as papers of not more than 10 pages. Authors are encouraged to submit their papers in LaTeX. Papers must be submitted in Springer Verlag's LNCS style through the conference homepage, here.

ICES 2008 - 8th International Conference of Evolvable Systems: From Biology to Hardware

September 21-24, 2008. Prague, Czech Republic
Homepage: http://www.fit.vutbr.cz/events/ices2008

The 8th International Conference of Evolvable Systems (ICSE 2008) which will be held in Prague, September 21-24, 2008. Topics to be covered include, but are not limited to: Evolutionary hardware design Evolutionary circuit diagnostics and testing, Self-reconfiguring/repairing and fault tolerant systems, co-evolution of hybrid systems, generative and developmental approaches, embryonic hardware, hardware/software co-evolution, intrinsic and extrinsic evolution, real-world applications of evolvable hardware, on-line hardware evolution, MEMS and nanotechnology in evolvable hardware, evolutionary robotics, formal models for bio-inspired hardware systems adaptive computing, novel devices/testbeds/tools for evolvable hardware.

Sixth International Conference on Ant Colony Optimization and Swarm Intelligence

September 22–24, 2008. Brussels, Belgium
Homepage: http://iridia.ulb.ac.be/ants2008/

Swarm intelligence is a relatively new discipline that deals with the study of self-organizing processes both in nature and in artificial systems. Researchers in ethology and animal behavior have proposed many models to explain interesting aspects of social insect behavior such as self-organization and shape-formation. Recently, algorithms inspired by these models have been proposed to solve difficult computational problems.

An example of a particularly successful research direction in swarm intelligence is **ant colony optimization**, the main focus of which is on discrete optimization problems. Ant colony optimization has been applied successfully to a large number of difficult discrete optimization problems including the traveling salesman problem, the quadratic assignment problem, scheduling, vehicle routing, etc., as well as to routing in telecommunication networks. Another interesting approach is that of **particle swarm optimization**, that focuses on continuous optimization problems. Here too, a number of successful applications can be found in the recent literature. [...]

ANTS 2008 will give researchers in swarm intelligence the opportunity to meet, to present their latest research, and to discuss current developments and applications.

The three-day conference will be held in Brussels, Belgium, on September 22–24, 2008. Tutorial sessions will be held in the mornings before the conference program.

Further Information

Up-to-date information will be published on the web site http://iridia.ulb.ac.be/ants2008/. For information about local arrangements, registration forms, etc., please refer to the above-mentioned web site or contact the local organizers at the address below.

Conference Address

ANTS 2008
IRIDIA CP 194/6 — Tel +32-2-6502729
Université Libre de Bruxelles — Fax +32-2-6502715
Av. F. D. Roosevelt 50 — http://iridia.ulb.ac.be/ants2008
1050 Bruxelles, Belgium — email: ants@iridia.ulb.ac.be

May 2009

2009 IEEE Congress on Evolutionary Computation (CEC 2009)

May 18-21, 2009, Trondheim, NORWAY

Homepage: www.cec-2009.org

Deadline December 1, 2007

The 2009 IEEE Congress on Evolutionary Computation (CEC 2009) will be at the Nova Conference Centre and Cinema, Trondheim, Norway during May 18-21, 2009. Sponsored by the IEEE Computational Intelligence Society, co-sponsored by the Evolutionary Programming Society and the Institution of Engineering and Technology, CEC 2009 continues the successful sequence of World-class events going back to 1999.

CEC 2009 will feature a world-class conference that will bring together researchers and practitioners in the field of evolutionary computation and computational intelligence from all around the globe. Technical exchanges within the research community will encompass keynote speeches, special sessions, tutorials, panel discussions as well as poster presentations. On top of these, participants will be treated to a series of social functions, receptions and networking sessions, which will serve as a vital channel to establish new connections and foster everlasting friendship among fellow researchers.

The annual IEEE Congress on Evolutionary Computation (CEC) is one of the leading events in the area of evolutionary computation. CEC covers all topics in evolutionary computation, including, but not limited to:

- Ant colony optimization
- Artificial immune systems
- Artificial life
- Autonomous mental & behaviour development
- Bioinformatics & bioengineering
- Coevolution & collective behaviour
- Cognitive systems & applications
- Combinatorial & numerical optimization
- Computational finance & economics
- Constraint & uncertainty handling
- Estimation of distribution algorithms
- Evolutionary data mining
- Evolutionary design
- Evolutionary games
- Evolvable hardware & software
- Evolutionary intelligent agents
- Evolutionary learning systems
- Evolving neural networks & fuzzy systems
- Molecular & quantum computing
- Particle swarm intelligence
- Representation & operators

Researchers are invited to contribute high-quality papers to CEC 2009. All papers are to be submitted electronically through the Congress website by November 1, 2008. All submitted papers will be refereed by experts in the fields based on the criteria of originality, significance, quality, and clarity. In addition, we are looking for high quality proposals for Special Sessions and Tutorials for the Congress. More details on all of all of these are below.

Call for Contributed Papers

Prospective authors are invited to contribute high-quality papers to CEC2009. All papers are to be submitted electronically through the Congress website. For general inquiries, please contact General Chair Andy Tyrrell at amt@ohm.york.ac.uk. For program inquiries, contact Program Chair Pauline Haddow at Pauline.Haddow@idi.ntnu.no.

Call for Special Sessions

CEC 2009 Program Committee solicits proposals for special sessions within the technical scopes of the congress. Special sessions, to be organised by internationally recognised experts, aim to bring together researchers in special focused topics. Papers submitted for special sessions are to be peer-reviewed with the same criteria used for the contributed papers. Researchers interested in organising special sessions are invited to submit formal proposals to the Special Session Chair Jon

Timmis at jt517@ohm.york.ac.uk. A special session proposal should include the session title, a brief description of the scope and motivation, names, contact information and brief CV of the organisers.

Call for Tutorials

CEC 2009 will also feature pre-congress tutorials covering fundamental and advanced computational intelligence topics. A tutorial proposal should include title, outline, expected enrollment and presenter biography. Tutorials are expected to run for 2 hours each. Researchers interested in organising tutorials are invited to submit formal proposals to the Tutorial Chair Stephen Smith at sls@ohm.york.ac.uk.

Important Dates:

- Special Session proposals: September 1, 2008
- Paper submissions: November 1, 2008
- Tutorial proposals: December 1, 2008
- Notification of acceptance: January 16, 2009
- Final paper submission: February 16, 2009

More information can be found at: www.cec-2009.org. For general inquiries, please contact General Chair Andy Tyrrell at amt@ohm.york.ac.uk.

About the Newsletter

SIGEVOlution is the newsletter of SIGEVO, the ACM Special Interest Group on Genetic and Evolutionary Computation.

To join SIGEVO, please follow this link [WWW]

Contributing to SIGEVOlution

We solicit contributions in the following categories:

Art: Are you working with Evolutionary Art? We are always looking for nice evolutionary art for the cover page of the newsletter.

Short surveys and position papers: We invite short surveys and position papers in EC and EC related areas. We are also interested in applications of EC technologies that have solved interesting and important problems.

Software: Are you are a developer of an EC software and you wish to tell us about it? Then, send us a short summary or a short tutorial of your software.

Lost Gems: Did you read an interesting EC paper that, in your opinion, did not receive enough attention or should be rediscovered? Then send us a page about it.

Dissertations: We invite short summaries, around a page, of theses in EC-related areas that have been recently discussed and are available online.

Meetings Reports: Did you participate to an interesting EC-related event? Would you be willing to tell us about it? Then, send us a short summary, around half a page, about the event.

Forthcoming Events: If you have an EC event you wish to announce, this is the place.

News and Announcements: Is there anything you wish to announce? This is the place.

Letters: If you want to ask or to say something to SIGEVO members, please write us a letter!

Suggestions: If you have a suggestion about how to improve the newsletter, please send us an email.

Contributions will be reviewed by members of the newsletter board.

We accept contributions in LaTeX, MS Word, and plain text.

Enquiries about submissions and contributions can be emailed to editor@sigevolution.org.

All the issues of SIGEVOlution are also available online at www.sigevolution.org.

Notice to Contributing Authors to SIG Newsletters

By submitting your article for distribution in the Special Interest Group publication, you hereby grant to ACM the following non-exclusive, perpetual, worldwide rights:

- to publish in print on condition of acceptance by the editor
- to digitize and post your article in the electronic version of this publication
- to include the article in the ACM Digital Library
- to allow users to copy and distribute the article for noncommercial, educational or research purposes

However, as a contributing author, you retain copyright to your article and ACM will make every effort to refer requests for commercial use directly to you.