View metadata, citation and similar papers at core.ac.uk brought to you by CORE
provided by Servicio de Coordinación de Bibliotecas de la Universidad Politécnica de Madrid

Self-Organizing Architectural design based on Morphogenetic Programming

Nuria Gómez Blas, Luis F. de Mingo, Miguel A. Muriel

Abstract: *In this paper, we present our research into self-organizing building algorithms. This idea of self-organization of animal/plants behaviour interests researchers to explore the mechanisms required for this emergent phenomena and try to apply them in other domains. We were able to implement a typical construction algorithm in a 3D simulation environment and reproduce the results of previous research in the area. LSystems, morphogenetic programming and wasp nest building are explained in order to understand self-organizing models. We proposed Grammatical swarm as a good tool to optimize building structures.*

Keywords: *LSystems, Morphogenetic Programming, Swarm Computing, Particle Swarm Optimization.*

ACM Classification Keywords*: F.1.1 Theory of Computation - Models of Computation, I.2.6 Artificial Intelligence - Learning.*

Conference topic*: Information Modelling, Information Systems, Applied Program Systems.*

MSC*: 68Q32 Computational learning theory, 68T05 Learning and adaptive systems.*

Introduction

Many species of animals/plants exhibit simple individual actions but in groups are able to demonstrate quite complex emergent behaviours. This idea of self-organization of animal behaviour interests researchers to explore the mechanisms required for this emergent phenomena and try to apply them in other domains (such as computation). In this work, we look at lsystems, morphogenetic programming and nest building in social wasp species and use a computer model of this behaviour in order to build articial architectures. We propose the use of grammatical swarm to optimize generated structures.

LSystems

A Lsystem is a rule like description of a 3d form. It contains descriptions of parts and how they should be assembled together. A simulation program reads a Lsystem description and processes it into a 3d form, see figure 1, which can then be outputted in several formats. The description is applied to itself a number of times (recursion levels) so fractal and recursive forms are very easy to describe in a Lsystem. That's why they are used a lot for plants, trees and natural looking organic forms. By increasing the recursion level the form slowly grows and becomes more complex.

LSystems were created by biologist Lindenmayer [Lindenmayer,1990] as a method to simulate the growth of plants. But they really are an implementation of Chomsky's generative grammars. An LSystem is a set of terminal and non terminal symbols and some rules that define how non terminal symbols generate strings of new symbols. LSystems are also called recursive string substitution systems. LSystems are very useful to simulate some natural growing processes, like fungi, plants, or inorganic forms like crystals, or natural patterns. Since LSystems are basically recursive processes, they are good examples of self similarity, and are often considered a kind of fractals. The von Coch curve, a well known fractal object, can be produced easily with LSystems. LSystems are a very interesting tool for generative artists, in fact they let explore form within natural processes, they can also add scientific knowledge to artworks (grammars are linguistic models of natural processes), if well implemented they can offer an interactive laboratory for the investigation of natural and artificial forms [Lindenmayer,1990; Rozenberg,1992]. They can be

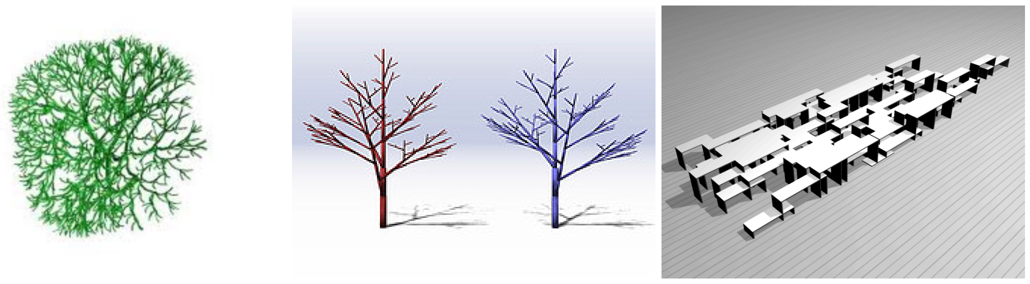


Figure 1: LSystems generated using LParser. [Lapré]

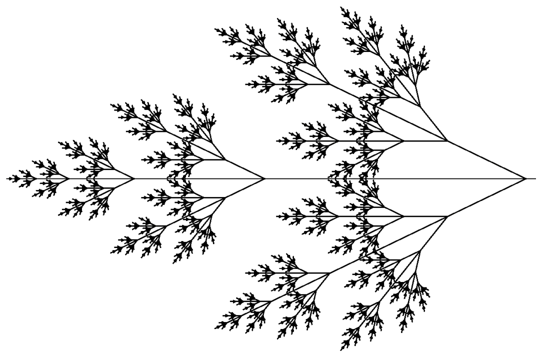
applied to sculpture, painting, music and architecture. The good of LSystems is that even with simple rules can be reached a great complexity. In fact, LSystems are quite simple to implement, but the complexity of the growing process challenges all the artistic skills we may have. So if the use of LSystems is straightforward, the good use of LSystems is very complex and needs a lot of experimentation.

A generic implementation of LSystem is designed upon the following architecture:

- The language: A set of terminal and non terminal symbols. Non terminal symbols are symbols that may generate new strings of symbols of the language. Terminal symbols always remain the same, usually they are used to scale, rotate and move the elements of the system, see figure 2. In fact non terminal symbols usually are interpreted as 2D or 3D objects, like Logo's turtle graphic commands. But they could represent words or sounds as well.
- The generative grammar:
  - Axiom: a string of symbols of the language set used as a starting point to the substitution process.
  - Rules: a rule is a string of any symbol of the language set that will replace a non terminal symbol. It is possible to create rules for every non Terminal symbol, random rules, context sensitive rules, etc.
  - The substitution process: This is a recursive process that apply the rules to every non Terminal symbol of the axiom, thus generating a new and larger string. This new string will be used as axiom for a new substitution process. This process can be repeated many times, generating bigger and bigger strings.
  - The LSystem string: Is the final string, ready to be parsed.
- The parser: A parser is the module that will read the LSystem string token by token (char by char) and perform the right action, depending of the meaning of every symbol. Seymour Papert invented "Turtle graphics" as a system for translating a sequence of symbols into the motions of an automaton (the "turtle") on a graphics display. So "F" could mean go forward for some units in the 2D or 3D space, "+" could mean turn clockwise by sdome degrees, "a" could mean draw a line, etc. You have to learn the vocabulary of the specific LSystem implementation you are using.
- Editors: In the web there are many LSystems free editors, like LParser, Fractint or GDesign. While they are similar in their basic architecture, they can be and perform differently. GDesign is the only one with a visual editor and real time 2D / 3D display.

Morphogenetic programming

Morphogenesis is the biological process that causes an organism to develop its shape. It is one of three fundamental aspects of developmental biology along with the control of cell growth and cellular differentiation. The process controls the organized spatial distribution of cells during the embryonic development of an organism. Morphogenesis



```
variables : X F
constants : + - [ ]
start   : X
rules   : (X -> F[+X][-X]FX), (F -> FF)
angle   : 25.7
```

Figure 2: Obtained figure using above grammar in a Lindenmayer system.

can take place also in a mature organism, in cell culture or inside tumor cell masses. Morphogenesis also describes the development of unicellular life forms that do not have an embryonic stage in their life cycle, or describes the evolution of a body structure within a taxonomic group. Morphogenetic responses may be induced in organisms by hormones, by environmental chemicals ranging from substances produced by other organisms to toxic chemicals or radionuclides released as pollutants, and other plants, or by mechanical stresses induced by spatial patterning of the cells.

Morphogenetic programming is the computational model of morphogenesis concept applied to architectural design, see figures 3, 4 and 5. Genetic algorithms, cellular automata, DNA models, neural networks, etc. take part on morphogenetic programming. On the other hand, swarm computing models could be considered macro models –they operate at individual level, no at molecular level-.

Achitectural behavior modeling based on cellular automata has been widely studied in recent years. Many of those models are based on the typical orthogonal mesh, the archetypical grid. The Cellular Automata orthogonal lattice is commonly used as the most neutral -thus general- geometrical base. It turns variables which are spatially extensive into their density-intensity equivalents and this immediately means that comparisons can be made [Batty,1999]. The geometric configuration of the spatial units used to represent the spatial data can have a profound effect, explaining why using spatial systems which neutralize the effect of configuration remove any bias caused by convoluted or distorting geometries.

The regular grid has other advantages, as the additional ability to work in layers without additional efforts to make different ones fit into the same system, and the significant work which proves its success even with highly refined Cellular Automata rules involving cultural and human factors [Portugali,1997].

For such reasons current models are centering their interest in orthogonal isotropic shape-constrained meshes (the regular grid); they are essentially analytic. Nevertheless, most of them function with a certain degree of deviation from classic Cellular Automata [Zhongwei,2003], redefining cell space, neighborhood, lattice and time concepts, which is necessary for some degree of flexibility.

Wasp nest building

Theory proposed by Thorpe states that wasps store an internal blueprint of the type of nest structure that they build. While building the nest, the wasps compare their blueprint with the environment in order to decide which action to perform. Strong experimental evidence against the existence of such blueprint were later demonstrated. The experiments were conducted by modifying a nest structure during construction and observing the effect on the nest building behaviour. It was observed that the wasps were not able to finish the nest as per the blueprint. The experiments of Smith provided an insight onto the nest building strategy of social wasps by showing that cues in the environment seemed to be the driving force in the construction, see figure 6.

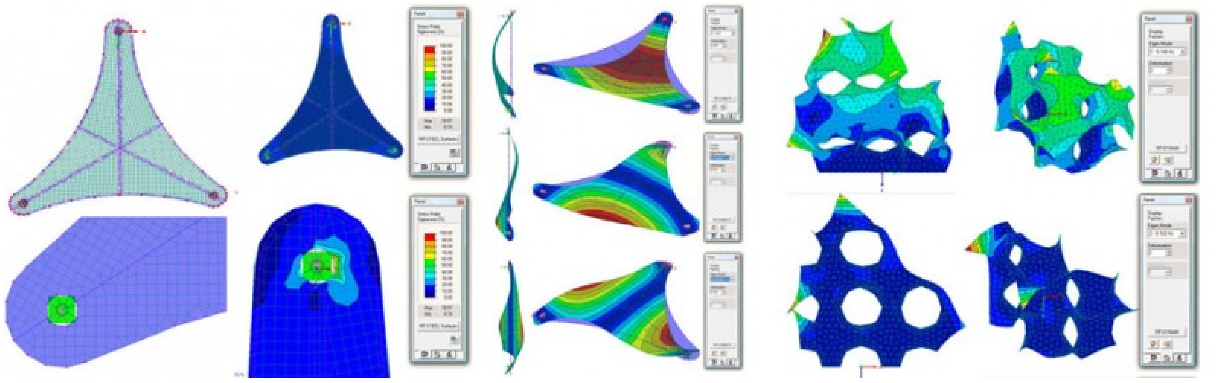


Figure 3: Minimal complexity prototype Houston 2011. [Tenu,2010]

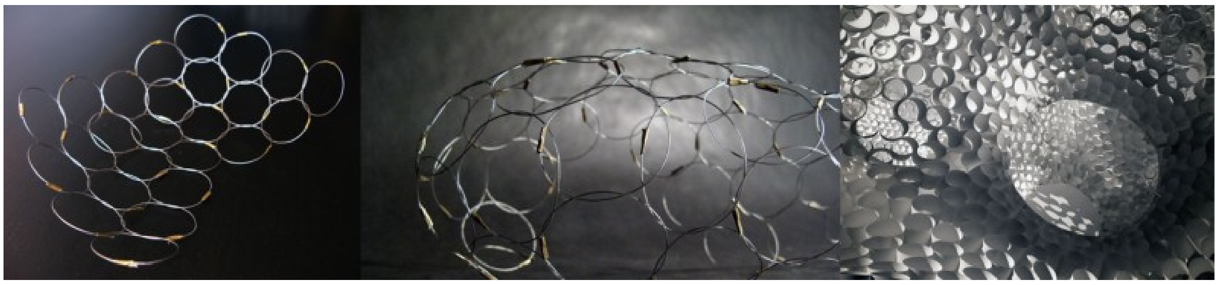


Figure 4: Minimal surfaces as architectural prototypes 2009. [Tenu,2010]

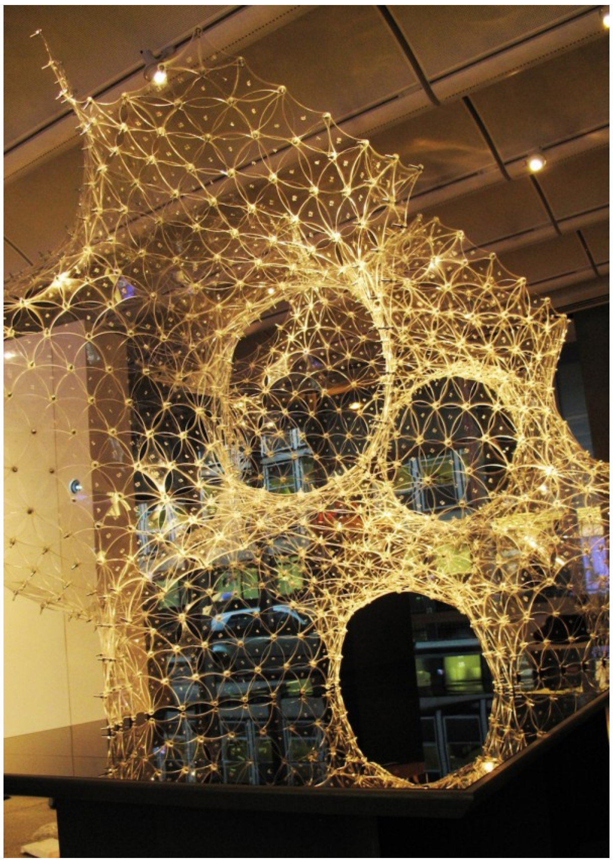


Figure 5: Self-organizing systems. Minimal complexity prototype London 2010. [Tenu,2010]

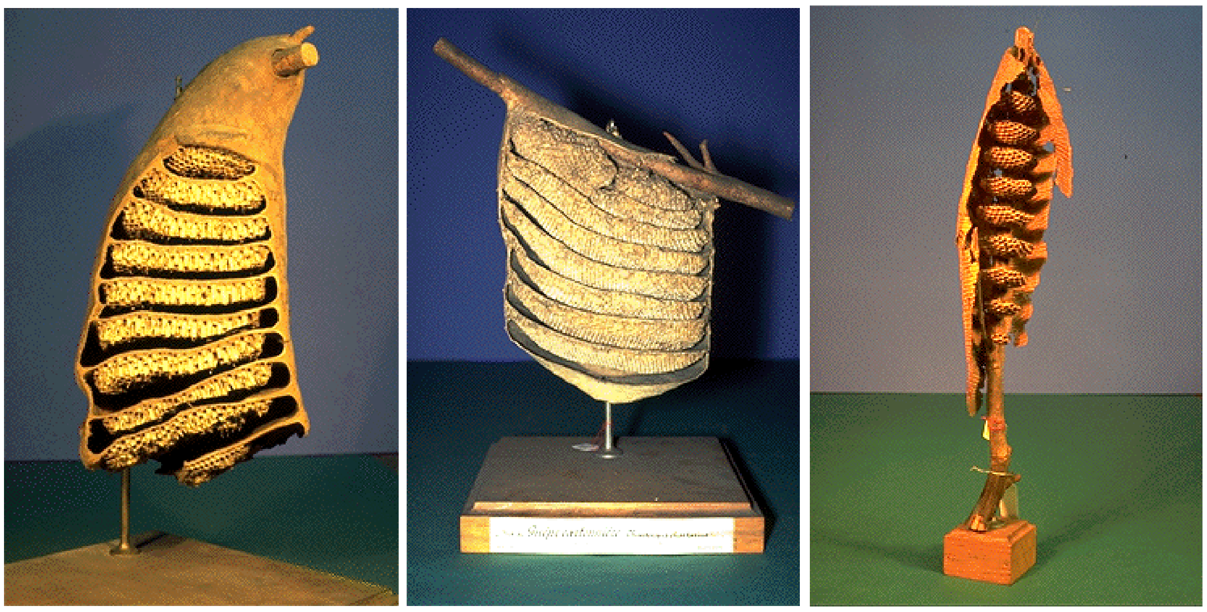


Figure 6: Examples of complex social wasp nests

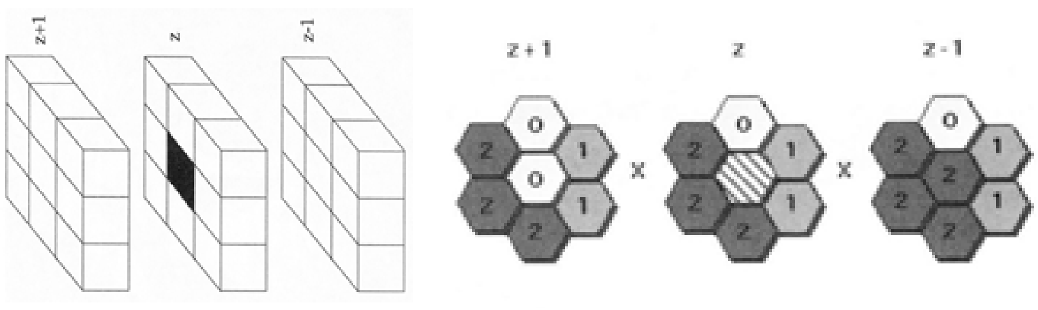
Stigmergy is defined as indirect communicating through the environment by leaving signs in the environment that could be picked up by others. Two forms of stigmergy have been identified: quantitative and qualitative. In quantitative or continuous stigmergy, the stimulus in the environment does not change, however, the amount of the stimulus can differ and evoke different responses to the stimulus. This model can be used to explain ant foraging behaviours and termite nest building.

Using the stigmergy model of wasp nest building, researchers have come up with a class of algorithms that can perform construction of artficial architectures. The algorithms use a swarm of agents that move randomly and independently in 3D space and try to match their stimulus-response systems with the local environment. The simulation space is usually a discrete cubic or hexagonal lattice hence the name lattice swarms, see figure 7. The elementary building blocks of the simulation are cubic or hexagonal bricks of different types. If an agent matches its local neighbourhood to a rule in the rule system, it deposits a brick of specified type at its current location in the lattice.

Several methods can be used for rule application. Rules can be matched deterministically or stochastically with a predefined probability. Only one set of rules can be matched or several sets of rules can be used either in seasonal manner or in a hierarchy of rule sets. The algorithms can be either coordinated or uncoordinated. In coordinated algorithms, several runs of the algorithm on the same rule system will yield architectures with common features. The architectures resulting from uncoordinated algorithms may not be similar. A high-level description of the construction algorithm is available in listing 1.

Grammatical Swarm

Grammatical Swarm (GS) relates Particle Swarm algorithm to a Grammatical Evolution (GE); genotype-phenotype mapping to generate programs in an arbitrary language. Grammatical Evolution (GE) is an evolutionary algorithm that can evolve computer programs in any language [O'Neill,2001; O'Neill,2003], and can be considered a form of grammar-based genetic programming. Rather than representing the programs as parse trees, as in GP [Koza,1999; Koza,2003], a linear genome representation is used. A genotype-phenotype mapping is employed such that each individual's variable length binary string, contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. The grammar allows the generation of programs in an arbitrary language that are guaranteed to be syntactically correct, and as such it is used as a generative grammar, as opposed



a) Squared lattice b) Hexagonal lattice

Figure 7: Lattices in wasp nest building

Algorithm 1 Wasp nest building basic algorithm

```
 1: /* Initialization */
 2: Construct lookup table
 3: Put one initial brick at predefined site
 4: **for** $k = 1$ to $m$ **do**
 5:     Assign agent $k$ a random unoccupied site
 6: **end for**
 7: **for** $t = 1$ to $t_{max}$ **do**
 8:     **for** $k = 1$ to $m$ **do**
 9:         Sense local configuration
10:         **if** Local configuration is in lookup table **then**
11:             Deposit brick specified by lookup table
12:             Draw new brick
13:         **else**
14:             Do not deposit brick
15:         **end if**
16:         Move to randomly selected, unoccupied, neighboring site
17:     **end for**
18: **end for**
```

to the classical use of grammars in compilers to check syntactic correctness of sentences. The user can tailor the grammar to produce solutions that are purely syntactically constrained, or they may incorporate domain knowledge by biasing the grammar to produce very specific forms of sentences. BNF is a notation that represents a language in the form of production rules.

Algorithm 2 PSO Hybrid Algorithm

```
1: Creation of the population of particles $\{x^i\}$ in the interval
2: for Each particle $i$ in the swarm do
3:   if $(f(x) > f(p))$ then
4:     for $(d = 1$ until $D)$ do
5:       $p_{id} = x_{id}$
6:     end for
7:   end if
8:   $g = i$
9:   for $j \in J$ do
10:    if $(f(p_j) > f(p_g))$ then
11:      $(g = j)$
12:    end if
13:  end for
14:  for $d = 1$ until $D$ do
15:    $v_{id}(t) = v_{id}(t-1) + c_1\epsilon_1(p_{id} - x_{id}(t-1)) + c_2\epsilon_2(g_d - x_{id}(t-1))$
16:  end for
17:  $nmut = rand(1, M)$
18:  for $j = 1$ until $nmut$ do
19:    $k = rand(1, M)$
20:    $x_{ik} = p_{ik}$
21:  end for
22:  $nmut = rand(1, M)$
23:  for $j = 1$ until $nmut$ do
24:    $k = rand(1, M)$
25:    $x_{ik} = g_k$
26:  end for
27: end for
```

The equations for the particle swarm algorithm, see algorithm 2, are updated by adding new constraints to velocity and location dimension values, such us $vmax$ (bounded to 255), and dimension which is bounded to the range [0; 255] (denoted as $cmin$ and $cmax$, respectively). Note that this is a continuous swarm algorithm with real-valued particle vectors. The standard GE mapping function is adopted, with the real-values in the particle vectors being rounded up or down to the nearest integer value for the mapping process. In the current implementation of GS,

fixed-length vectors are used, which implies that it is possible for a variable number of dimensions to be used during the program construction genotypephenotype mapping process. A vector's elements (values) may be used more than once if wrapping occurs, and it is also possible that not all dimensions are used during the mapping process. (This can happen whenever a program is generated before reaching the end of the vector). In this latter case, the extra dimension values are simply ignored and are considered as in trons that may be switched on in subsequent iterations.

Let us suppose the following BNF grammar:

```
<expr> :: = <expr><op><expr>
          | <var>
<op> :: = +
        | -
        | *
<var> :: = x
         | y
```

And the following genotype:

14	8	27	254	5	17	12

In the example individual (see figure 8), the left-most `<expr>` in `<expr> <op> <expr>` is mapped by reading the next codon integer value 240 and used in $240\%2 = 0$ to become another `<expr> <op> <expr>`. The developing program now looks like `<expr> <op> <expr> <op> <expr>`. Continuing to read subsequent codons and always mapping the left-most non-terminal the individual finally generates the expression `y * x - x - x + x`, leaving a number of unused codons at the end of the individual, which are deemed to be introns and simply ignored.

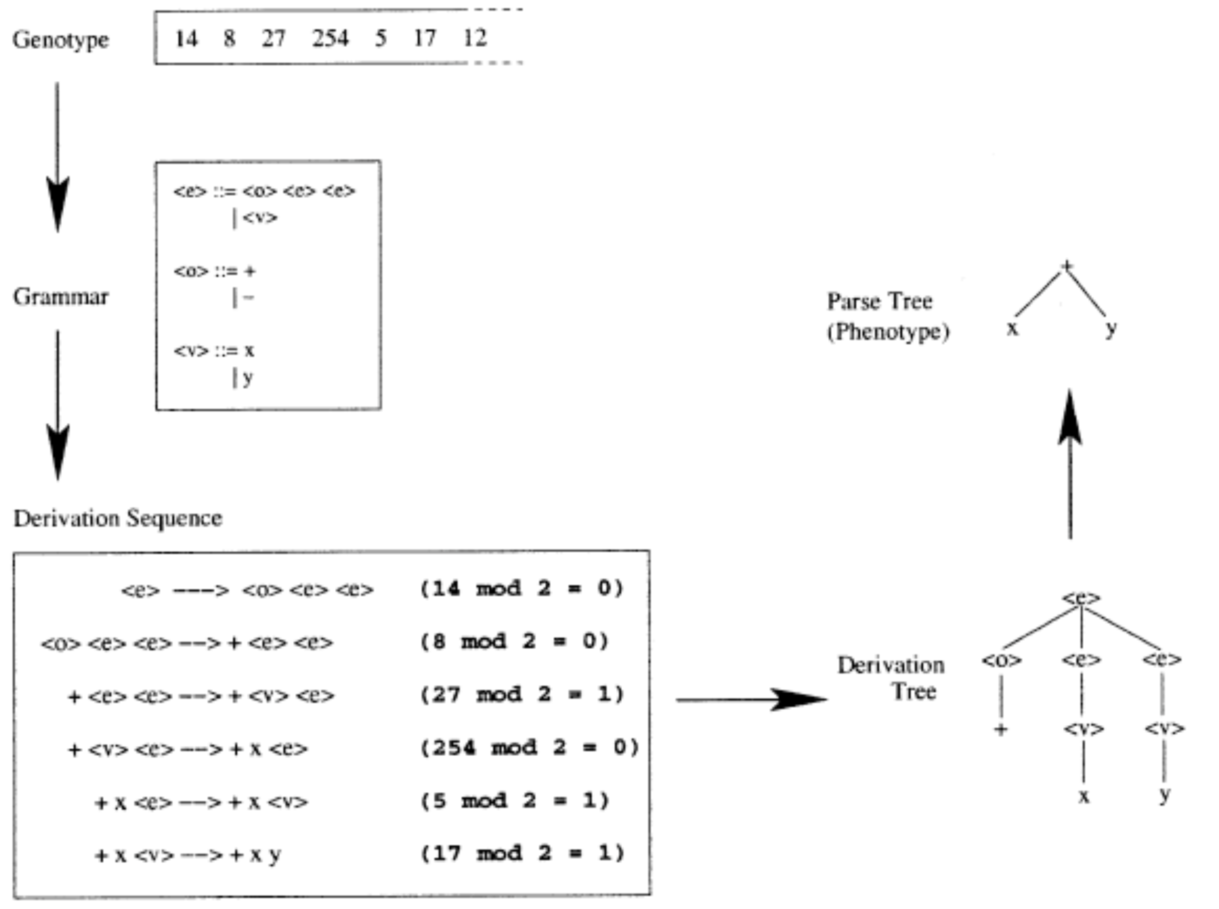

Genotype
14 8 27 254 5 17 12
Grammar
<e> ::= <o> <e> <e>
| <v>
<o> ::= +
| -
<v> ::= x
| y
Derivation Sequence
<e> ---> <o> <e> <e> (14 mod 2 = 0)
<o> <e> <e> ---> + <e> <e> (8 mod 2 = 0)
+ <e> <e> ---> + <v> <e> (27 mod 2 = 1)
+ <v> <e> ---> + x <e> (254 mod 2 = 0)
+ x <e> ---> + x <v> (5 mod 2 = 1)
+ x <v> ---> + x y (17 mod 2 = 1)
Derivation Tree
<e>
<o> <e> <e>
+ <v> <v>
x y
Parse Tree (Phenotype)
+
x y

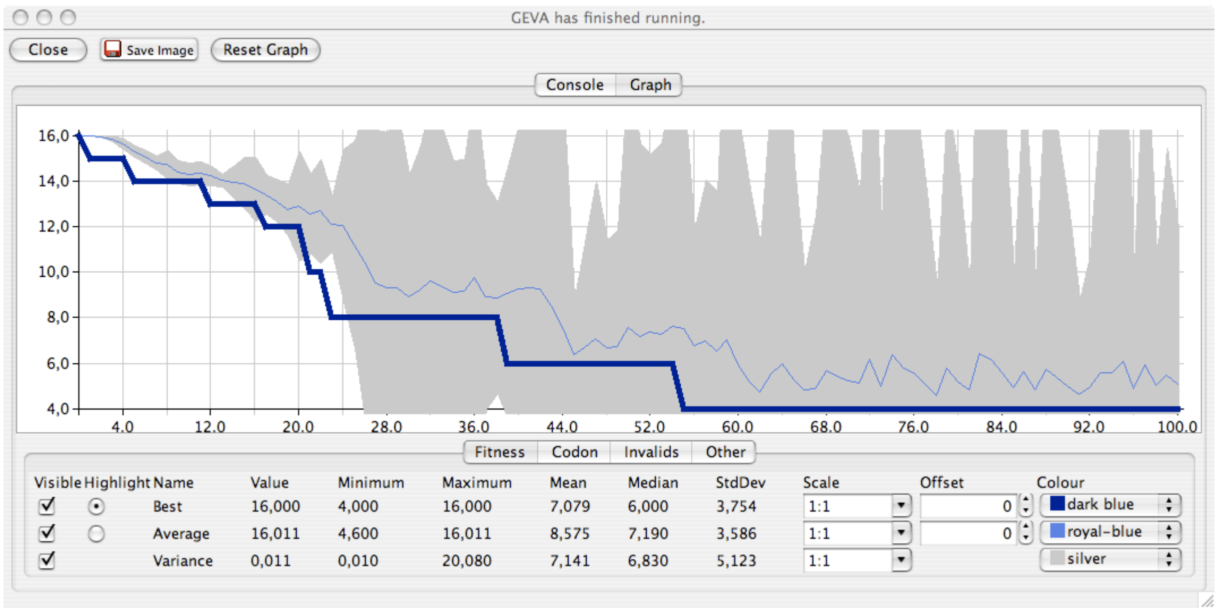
Figure 8: Grammatical Swarm concepts.

This is the classic benchmark problem in which evolution attempts to find the five input even-parity boolean function [Geva]. The grammar adopted here is:

```
<prog> ::= <expr>
<expr> ::= <expr> <op> <expr>
         | ( <expr> <op> <expr> )
         | <var> | <pre-op> ( <var> )
<pre-op> ::= not
<op> ::= "|" | & | ^
<var> ::= d0 | d1 | d2 | d3 | d4
```

The result is given by the best individual, see transcript bellow. Figure 9 shows a graphic with the best, average and variance of the swarm population. This figure has been obtained using the *GEVA* simulator [Geva].

```
( not ( d1 ) | d2 ^ d4 ) &
not ( d3 ) ^ ( not ( d1 ) &
( not ( d2 ) &
( not ( d2 ) |
( d1 ^ not ( d3 ) ^ not ( d1 ) ^
( not ( d1 ) ^ ( d0 | not ( d4 ) ) ) ) ) ^ d4 )
 ^ not ( d0 ) ) ^ d1
```



Visible	Highlight	Name	Value	Minimum	Maximum	Mean	Median	StdDev	Scale	Offset	Colour
✓	●	Best	16,000	4,000	16,000	7,079	6,000	3,754	1:1	0	dark blue
✓	○	Average	16,011	4,600	16,011	8,575	7,190	3,586	1:1	0	royal-blue
✓		Variance	0,011	0,010	20,080	7,141	6,830	5,123	1:1		silver

Figure 9: Results of *even-5-parity* problem simulated with GEVA.

Wasp nest building and Grammatical swarm

This part describes the process to follow in order to obtain a structure that minimize a certain function, according to all previous self-organizing tools. Steps needed are the following:

1. Define a grammar to generate patterns in a wasp nest building algotihm:

   Used grammar to generate self-organizing models (squared lattice, see figure 7, in case an hexagonal lattice is desired just generate 7 `<block>` in rules `<z+1>, <z>, <z-1>` instead 9), where 1 means there is a block and -1 means there is no block:

   $\langle$*building_patterns*$\rangle$ ::= $\langle$*pattern*$\rangle$ ':' $\langle$*building_patterns*$\rangle$
   | $\langle$*pattern*$\rangle$

```
⟨pattern⟩   ::= ⟨z-1⟩ ⟨z⟩ ⟨z+1⟩
⟨z-1⟩       ::= ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩
⟨z⟩         ::= ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨wasp⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩
⟨z+1⟩       ::= ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩ ⟨block⟩
⟨block⟩     ::= '-1'
              | '1'
⟨wasp⟩      ::= '0'
```

2. Define a fitness function for a given generated structure.

3. Apply the grammatical swarm algorithm to generate patterns:

   These patterns are the rules of the wasp nest building algorithm. Fitness function is computed once the structure is finished.

Acknowledgements

The research was supported by the Spanish Research Agency projects TRA2010-15645 and TEC2010-21303-C04-02.

Bibliography

[Tenu,2010] Tenu, V.: Minimal Surfaces as Self-Organizing Systems. ACADIA Conference, Cooper Union in New York in October 2010. http://www.vladtenu.com

[Lapré] Laurens Lapré. http://laurenslapre.nl/index.html

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

[Rozenberg,1992] Rozenberg G., Salomaa A.; Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology, ISBN 978-3-540-55320-5. Springer-Verlag. 1992.

[O'Neill,2001] O'Neill M., Ryan C.; Grammatical Evolution, IEEE Trans. Evolutionary Computation, Vol. 5, No.4, 2001.

[O'Neill,2003] O'Neill M., Ryan C., Keijzer M., Cattolico M.; Crossover in Grammatical Evolution, Genetic Programming and Evolvable Machines, Vol. 4 No. 1. Kluwer Academic Publishers, 2003.

[Koza,1999] Koza J.R., Andre D., Bennett III F.H., Keane M.; Genetic Programming 3: Darwinian Invention and Problem Solving, Morgan Kaufmann, 1999.

[Koza,2003] Koza J.R., Keane M.,Streeter M.J., Mydlowec W., Yu J., Lanza G.; Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers, 2003.

[Geva] http://www.grammatical-evolution.org/

[Batty,1999] M. Batty, Y. Xie, Z. Sun. Modeling urban dynamics through GIS-based CA. In: Computers, Environment and Urban Systems. Elsevier, 1999.

[Zhongwei,2003] Sun Zhongwei. Simulating urban growth using cellular automata. Intl. Inst. for Geoinformation science and Earth observation. The Netherlands, 2003.

[Portugali,1997] J. Portugali, I. Benenson. Individuals' cultural code and residential self-organization in the city space. In: Proceedings of GeoComputation 97 and SIRC 97. University of Otago, New Zealand, 1997.

Authors' Information

Luis Fernando de Mingo López *- Dept. Organización y Estructura de la Información, Escuela Univesitaria de Informática, Universidad Politécnica de Madrid, Crta. de Valencia km. 7, 28031 Madrid, Spain; e-mail: lfmingo@eui.upm.es*
Major Fields of Scientific Research: Artificial Intelligence, Social Intelligence

Nuria Gómez Blas *- Dept. Organización y Estructura de la Información, Escuela Univesitaria de Informática, Universidad Politécnica de Madrid, Crta. de Valencia km. 7, 28031 Madrid, Spain; e-mail: ngomez@eui.upm.es*
Major Fields of Scientific Research: Bio-inspired Algorithms, Natural Computing

Miguel A. Muriel *- Dept. Tecnología Fotónica, ETSI Telecomunicación, Universidad Politécnica de Madrid, Avenida Complutense 30, Ciudad Universitaria, 28040 Madrid, Spain; e-mail: m.muriel@upm.es*
Major Fields of Scientific Research: Theoretical Computer Science, Microwave Photonics