USING GENERATIVE REPRESENTATIONS FOR STRUCTURAL DESIGN

Yu Zhang, Alan Kwan, John Miles
Cardiff School of Engineering, The Parade, Cardiff University, UK

ABSTRACT: Work in recent years has shown that topological reasoning with search algorithms using traditional representations such as parameters, ground structures, voxels, etc is very limiting. Each type of representation is only to be suitable for a limited number of topologies. This is restrictive because there are many problems where the topology of the solution is unknown except in the most general terms or there are competing topologies which are suitable for solving a given problem. Hence, at best, choosing a representation technique can be difficult and at worst it can restrict the search so that a full examination of the problem is not possible. Also, as the available computational power increases and the technology of search algorithms is enhanced, the topologies being reasoned about become ever more complex and so the representations within the algorithms can become cumbersome. A possible solution to these difficulties is the use of generative geometries where the object is represented by a set of rules which describe how to create the object. These can, when correctly implemented, give a compact representation and one which can be handled within typical search algorithms like for example genetic algorithms. This paper looks at the use of L-systems. They are being applied to beam design problems although this paper focuses on the representation. As will be shown in the paper, although the representation has some attractions, there are also some difficulties with the implementation and especially with enforcing constraints. The paper describes work which is in progress rather than a completed project.

KEYWORDS: generative representation, evolutionary computation, structures, search algorithms.

1 INTRODUCTION

Design is a complex task involving searching for solutions while applying constraints. Design is also, typically, a multi-disciplinary activity involving many participants and disciplines. Although the participants try to understand the objectives and constraints of the other disciplines, given the high level of specialisation that occurs, it is inevitable that they will not be totally successful. Hence the various participants will set their objectives and constraints given a less than perfect understanding of the given problem. Coupled to this is the shear complexity of design spaces. It can easily be demonstrated that design spaces, for anything more than relatively trivial problems, contain at least millions of feasible solutions and often contain billions. The situation is further complicated by the fact that the design space is not static but evolves as the design process progresses and the objectives and constraints are refined.

Given the above problems, designers have virtually no chance of locating areas of "good" solutions within the design space. Claims are made by designers that they can achieve this by using heuristics based on experience but even a basic level of understanding of statistics shows that the probability of this occurring is extremely low, less than the chance of winning most national lotteries.

If the design process is to be improved, then the only feasible option would seem to be some sort of computational support. It has already been shown that "black box" approaches are not feasible (e.g. see Rafael & Smith (2003)) and so it has to be some form of collaboration between the designer and the computational support. Currently the most promising approach is the use of evolutionary algorithms to search through the design space and find areas of high performance (Parmee, 2001). These algorithms have the ability to cope with the complexity of design search spaces and additionally are able to find areas of good solutions by sampling a small fraction of the total space.

If evolutionary algorithms are to be totally successfully as design tools, then they have to be able to reason about the full complexity of whatever problem is being considered. That means that they have to cope with the multi-disciplinary aspects, the constraints and the complexity. The literature abounds with examples of evolutionary algorithms being applied to multi-disciplinary design problems but in all cases, the problems that are solved only consider part of the challenge. Take for example the design of typical framed buildings such as an office block. Three design teams have provided examples of the use of evolutionary design systems to solve this problem, these being Rafiq(1999), Khajehpour & Grierson (1999) & Sisk et al (1999). Probably the most complete of these, in terms of domain coverage, is the work of Khajehpour and Grierson, but their consideration of the architectural aspects is minimal and there is no consideration of fire engineering needs. Also for all three of the above examples, the topology of the building is fixed. The methods developed only apply to buildings with a rectangular floor plate. If truly comprehensive software tools to support designers are to be developed, then a fundamental aspect of such systems will be their topological reasoning ability.

The subject of topological reasoning was discussed by Zhang et al (2006) who showed that the current approaches used with evolutionary algorithms all have their limitations. The use of parameters effectively restricts the search to the shapes that can be described with the chosen parameter set and other techniques such as voxels and computational geometry based methods are only suitable for certain classes of problems. Therefore new approaches are needed. This paper looks at a potentially interesting approach which is the use of generative representations where the representation is defined as the method of describing the topology within the algorithm.

Generative representations are relatively new. A good example is the work of Hornby (2003) who uses generative representations to solve a number of problems including the design of tables. Examples from the construction industry are rare probably the best example being Kicinger et al (2005) who use cellular automata to design bracing systems for tall buildings.

In this paper, the use of L-systems, as advocated by Hornby (2003), is investigated. The aim of the work is to use the representation to search for good solutions to beam design problems but at present the work is focussing on the representation. The paper therefore concentrates on this latter issue and looks at some of the lessons that have been learned to date. The research is work in progress rather than a completed project but nevertheless, the findings are of interest and will be of use to others who are contemplating using generative representations. The representation is linked to a Genetic Algorithm (GA) with the latter being used as the search engine.

2 PERCEIVED ADVANTAGES OF GENERATIVE REPRESENTATIONS

The traditional representation used with evolutionary algorithms is the binary string. Binary representations can be employed in a variety of ways, for example a Boolean "on – off" form is used with voxel representations (e.g. Griffiths & Miles, 2003) and many others use binary representations of numbers. Other examples use real number representations (e.g. Sisk et al, 1999). In the latter example the length of the genome was allowed to vary as the solution changed.

The above examples are both of static representations where the genome directly represents some salient feature of the problem being solved. For a generative representation, the genome only represents the problem indirectly because it consists of a set of rules for building the solution.

This indirect representation can result in a very compact genome being capable of representing a complex solution. It also means that just by changing one part of the genome, quite substantial changes in the solution are possible (Hornby, 2003). However, this could also be a disadvantage because the search may lack stability.

3 L-SYSTEMS & TURTLE GRAPHICS

L-Systems were derived by Lindemayer (Przemyslaw et al, 1990) to provide functions that will generate plant like shapes. They have since been adopted as a sort of shape grammar by a number of people and used to look at design problems (e.g. Coates (1997), Hornby (2003)). In the original L-Systems, plant like objects are created by successively replacing parts of a simple object by using a set of rewriting rules.

The process of rewriting involves working through a sequence of symbols and replacing each symbol with another symbol. This is achieved through a set of rules which specifies exactly how the rewriting process should occur. By undertaking this in an iterative manner starting from an initial symbol, complex strings can be created from a simple representation. Hornby (2003) gives the following example:

$$a: \rightarrow ab$$

$$b: \rightarrow ba$$

If one starts with the symbol a, the following strings are produced :

a

ab

abba

abbabaab

...

If L-systems are linked to turtle graphics (Abelson & diSessa, 1981), then they can be used for producing shapes. An example of turtle graphics is given below in fig.1

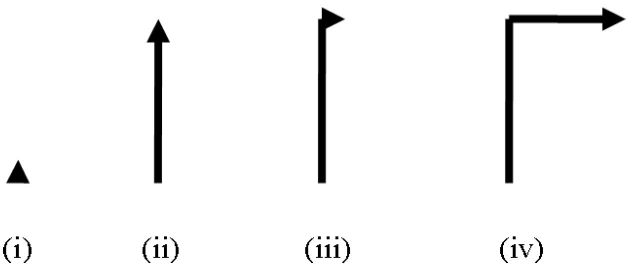

Figure 1. Turtle Graphics.

In this example the turtle is seen in its initial state in fig.1(i). The turtle's X axis is defined as being the direction in which it is facing and in this case, so far we the viewer is concerned, it is pointing upwards. If it is given the command **f(1)**: the turtle moves forward one step (fig.1(ii)). It is then given the command **c(1),** which means rotate clockwise by 90^o (fig.1(iii)), and another forward command **f(1)** then causes the turtle to move one step as shown in fig.1(iv). The full commands for turtle graphics coupled to a voxel representation are given in table 1.

For two-dimensional representation, the set of commands above can be simplified as is shown in table 2.

To use the graphics commands with L-Systems, rules of the following form are employed,

$$A :\rightarrow F(n)[L(n)A]R(n)F(n)A$$

which produces the following sequence of strings,

1. A
2. F(n)[L(n)A]R(n)F(n)A
3. F(n)[L(n)F(n)[L(n)A]R(n)F(n)A]R(n)F(n)F(n)[L(n)A]R(n)F(n)A

Table 1. Turtle graphics commands for voxel structures (Hornby, 2003).

Command	Description
[]	Push/ pop state to stack
f(n)	Move forward in the turtle's X direction by n units
b(n)	Move backwards in the turtle's X direction by n units
u(n)	Rotate heading n x 90° about the turtle's Z axis
d (n)	Rotate heading n x -90° about the turtle's Z axis
l(n)	Rotate heading n x 90° about the turtle's Y axis
r(n)	Rotate heading n x -90° about the turtle's Y axis
c(n)	Rotate heading n x 90° about the turtle's X axis
cc(n)	Rotate heading n x -90° about the turtle's X axis

Table 2. Simplified turtle graphics commands.

Command	Description
[]	Push/ pop state to stack
F(n)	Move forward by n units
L(n)	Rotate the turtle's heading n x δ° to the left
R(n)	Rotate the turtle's heading n x δ° to the left

4 PREVIOUS WORK

There is not a lot of previous work on the coupled use of L-systems and turtle graphics to create some form of structure. The most significant contribution has been made by Hornby (2003), who used the approach to develop table like structures. Hornby made a fundamental addition to the grammar of L-Systems by introducing a repeat operator which in concept is very much like a *For – Next* loop used in many computer languages. For example, {block}(n) repeats the enclosed block n times. This coupled with the use of parametric L-Systems greatly enhances the functionality of the approach from that which is in the simple example given above. An example of such an L-System is given below.

$$P0(n_0): n_0 \geq 1 \rightarrow F(n_0)[P1(n_0 \times 2)]L(1)F(2)R(1)P0(n_0 - 1)$$

$$P1(n_0): n_0 \geq 1 \rightarrow \{R(1)[F(n_0)]\}(2)$$

This L-System has two production rules, namely P0 and P1. A production rule consists of three components: a predecessor (e.g. $P1(n_0)$), a condition (e.g. $n_0 \geq 1$) and a successor (e.g. $\{R(1)[F(n_0)]\}(2)$). Normally, an L-System contains more than two production rules, each of which contains multiple condition-successor pairs. Fig.2 shows an example of Hornby's table design which is produced from an L-System of fifteen production rules. Each of the rules contains two or three condition-successor pairs.

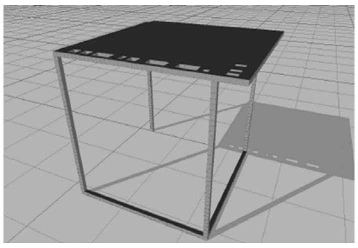

Figure 2. A table evolved using generative representation.

5 CURRENT WORK

For the current work, L-Systems are used as generative representations with the aim being to initially design 2D beam cross sections and later to move onto 3D shapes. The work is still in progress and because of complications which have arisen when the authors have tried to implement the representation, very little progress has been made on the beam design. Therefore this is not discussed in this paper which instead looks at some of the implementation issues that occur when using L-Systems with GAs. The discussion below focuses on problems that have been found with this form of representation.

6 POPULATION INITIALISATION

Each individual within the GA's population is considered as a parametric L-system with fixed number of production rules each with a fixed number of condition-successor pairs. Generating an individual involves filling in a blank template for such an L-system. Conditions are created by comparing a random parameter against a constant value. Successors are created by joining together a fixed number of blocks of commands and productions, which may be enclosed by push/pop symbols ("[" and "]"), or replication symbols ("{" and "}"). An example of such production rules is shown below.

P0(n0,n1):

n1>15 → {F(2)}(n0){P4(n1-1,n1+2)L(2)}(n1)P6(n1-1,2-n1)L(n1)

n0>6 → {R(3)}(n1)P6(n1-2,n0+3)[P12(n1-1,3-n0)L(n1)][L(1)]L(2)

n1>0 → R(n0)F(1){R(2)L(3)}(n0){R(2)L(n0)}(3)L(n1)

An individual also contains information like the starting condition, that is, the initial values of n0 and n1, and the number of iterations of the rewriting that are to be allowed. After an L-system is generated, it is evaluated to see if its fitness is above a certain value. Currently, only those that score above a preset threshold are accepted into the initial population. The process stops when the population is full.

7 EVALUATING AN INDIVIDUAL

Individuals are stored in the program and processed by the GA operators as L-systems rather than the structures they represent. However, in order to evaluate the fitness of an individual, it has to be interpreted into a structure (i.e. a phenotype). This introduces a possibly significant difference between this type of generative representation and the more traditional forms. The representation is a set of rules rather than a string describing the salient features of the problem being solved. The fitness is assessed using the phenotype but a small change in the genotype can produce a massively different phenotype so the typical relationship between selective pressure, expressed in concepts such as the schema theory (Goldberg, 1989), and the genotype appears to be weaker with this type of representation. How strong this relationship actually is and its influence on selective pressure, is something which the authors intend to investigate in the near future.

After a preset number of rewriting iterations (the limit is imposed because the expanded form of the genotype can be huge) by removing the production symbols (e.g. P0(n1,n2), P1(n1,n2)...), a string that only contains the construction commands including the push/pop and replication symbols is achieved. These commands direct the movement of the turtle, which produces a series of connected line segments. Mapping those line segments into the predefined grid and filling in each of the voxels that those line segments pass through, results in a continuum which is ready to be evaluated in conjunction with external FEA modules. Fig.3 shows a two-dimensional example of this interpretation with the line segments on the left.

The resolution of the grid affects the shape that is created. Fig.4 shows coarser and finer grids.

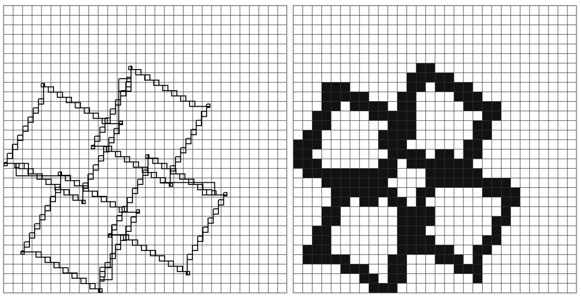
Figure 3. 2D example of L-system to continuum interpretation.

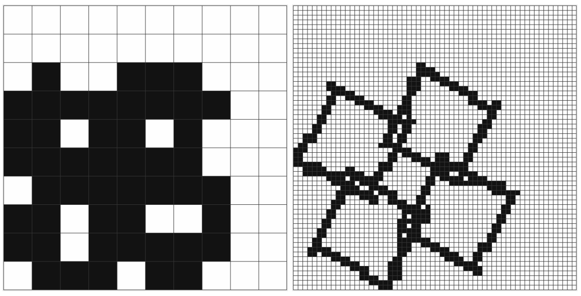
Figure 4. Coarser and finer grids.

8 GENETIC OPERATORS

For the generative representation that is discussed in this paper, genetic operators such as crossover and mutation do not act upon the structures but the L-systems that define those structures, that is, the production rules. There are several ways to mutate an individual as well as to apply crossover. Supposing that P0 is to be mutated,

P0(n0,n1):

n0>1 → F(3)[R(n0)F(n1)]P2(n0-1,n1)L(1){F(1)}(2)

some of the possible mutations are:

1. mutate a parameter:

P0(n0,n1):

n0>1 → F(3)[R(**2**)F(n1)] P2(n0-1,n1)L(1){F(1)}(2)

2. delete a symbol:

P0(n0,n1):

n0>1 → F(3)[R(n0)F(n1)]P2(n0-1,n1){F(1)}(2)

3. insert a symbol:

P0(n0,n1):

n0>1 → F(3)[R(n0)F(n1)**L(2)**]P2(n0-1,n1)L(1){F(1)}(2)

4. replace a symbol:

P0(n0,n1):

n0>1 → F(3)[R(n0)F(n1)]P2(n0-1,n1)**R(n0)**{F(1)}(2)

For the crossover, entire condition-successor pairs of selected production rules or part of each can be swapped between the parent individuals to generate new individuals. Supposing that P1 from parent 1 and P2 from parent 2 are selected for crossover,

Parent 1:

P1(n0,n1):

n1>1 → [F(1)R(1)]F(n0)P3(n0-1,n1+1)

Parent 2:

P2(n0,n1):

n0>2 → {P(3,n1)F(2)}(2)L(1)F(n1)

possible results for crossover are:

1. replace the entire condition-successor pair:

Child 1:

P1(n0,n1): n0>2 → {P(3,n1)F(2)}(2)L(1)F(n1)

Child 2:

P2(n0,n1):

n1>1 → [F(1)R(1)]F(n0)P3(n0-1,n1+1)

2. replace part of the condition-successor pair:

Child 1:

P1(n0,n1): n1>1 → [F(1)R(1)] L(1)F(n1)P3(n0-1,n1+1)

Child 2:

P2(n0,n1): n0>2 → {P(3,n1)F(2)}(2) F(n0)

Understanding the effect of mutation and crossover is an important part of understanding the representation. Work is being done on this topic at the moment and comprehensive results are yet to be determined. However, what can be said is that the impact of both crossover and mutation in topological terms is very similar. Also, for both operators, the impacts on the form of the phenotype range from negligible to highly significant depending on the position of the change and what is changed.

9 OVERLAPPING PROBLEMS

For turtle graphics, the overlapping of line segments is allowed. This is useful for producing branches rather than creating a single path from the start to the end. However, too much overlapping may cause problems. Fig.5 shows an example for a 2D case shown in line form (i.e. not converted into voxels). Clearly, the magnified part of the solution shown on the right of Fig.5 contains far more information than is needed just to create the shape. It is yet to be understood whether the duplication of information is good or bad in terms of the evolutionary process but it does cause problems when creating the phenotypes as their computational size can be very large.

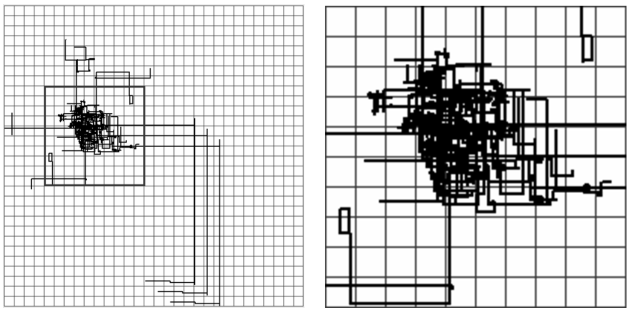
Figure 5. Excessive overlapping.

10 SIZE PROBLEMS

Most structural design problems have restrictions upon the physical size of the solution. Unlike representations using parameters or voxels, using an L-system as a generative representation results in a lack of control over the size of the structure it defines. Even with a small number of rewriting iterations, the resulting structures from randomly generated individuals can often exceed any predefined boundary. One thing that the authors have tested in their research is forcing the algorithm to only accept individuals from random initialisation, mutation and crossover operations that fit within pre-defined geometrical boundaries. However, experiments suggest that, by doing this, the efficiency of the algorithm is significantly decreased because of the loss of genetic information from the non-compliant solutions. It also appears that the restriction imposed on the search leads to premature convergence on sub-optimal solutions. In order the solve this problem, the idea of a "soft boundary" is being tested, in which illegal individuals are allowed within the population but are penalised in terms of their fitness according to how much they exceed the "real boundary". In effect this is a constraint violation problem and the difficulties that have been experienced have been typical of search problems where non-compliant individuals are deleted from the population. At the time of writing, experiments are being carried out to test the application of the "soft boundary" concept.

11 CONCLUSIONS

Generative representations offer the attractive possibility that one can create very complex topologies using a set of relatively simple and compact rules. The work described in this paper, is the start of an investigation into the use of L-systems for the representation of structural elements. At present the work is in its infancy but already some significant findings have been made :-

- The relationship between the assessment of the fitness of the phenotype and the selective pressure on the genotype is much weaker than usual because the relationship between the genotype is a set of rules rather than being a more direct representation.
- The impact of crossover and mutation is relatively similar.
- The impact of the changes induced by crossover and mutation ranges from negligible to highly significant.
- It is difficult to control the geometrical size of the shapes that are created.
- There can be a very significant degree of repetition within an individual in terms of the rules covering the same space many times. This can result in phenotypes which are computationally very large.

Further work is taking place to identify solutions to some of the problems that have been found to date and to investigate more fully the suitability of L-systems for solving topological reasoning problems in structural design.

REFERENCES

Abelson H & diSessa A, 1981. Turtle Geometry: The computer as a medium for exploring maths, MIT Press Cambridge USA, 477pp.

Coates P, 1997. Using genetic programming & L-Systems to explore 3D design worlds, In Junge R (ed), Proc of CAAD futures, Munich, Germany.

Griffiths, D.R. & Miles, J.C., 2003. Determining the optimal cross-section of beams, Advanced Engineering Informatics, 17, 59-76.

Goldberg, D,1989. Genetic algorithms in search, optimization and machine learning, New-York: Addison-Wesley, 300pp.

Hornby, G..,2003. Generative Representations for Evolutionary Design Automation, PhD thesis, Department of Computer Sci., Brandeis University, 242pp.

Khajehpour, S. & Grierson, D.1999. Filtering of Pareto-Optimal trade-off surfaces for building conceptual design, in Topping B.H.V. & Kumar, B. (eds) Optimzation & control in Civil & Structural Engineering, Civil-Comp Press, Edinburgh UK, 63-70.

Kicinger R, Arciszewski T & De Jong K.,2005, Evolutionary computation & structural design: A survey of the state of the art, Computers & Structures, 83, 1943-1978.

Parmee, I, 2001. Evolutionary and adaptive computing in Engineering design, Springer-Verlag, London, 286pp.

Przemyslaw, P., Lindenmayer A & Hanan J, 1990. The algorithmic beauty of plants, Springer-Verlag, New York, 228pp.

Rafael, B & Smith I, 2003. Fundamentals of Computer-Aided Engineering, Wiley, 306pp.

Rafiq, M., Bugmann, G. & Easterbrook, D., 1999. Building concept generation using genetic algorithms integrated with neural networks, In Borkowski, A. (Ed.) AI in Structural Eng.: IT for Design, Manufacturing, Maintenance and Monitoring. Proceedings of the 6th EG-SEA-AI Workshop, Wierzba, Poland, 165-173.

Sisk, G., Moore C. & Miles, J., 1999. A decision support system for the conceptual design of building structures using a genetic algorithm. in Borkowski A. (ed) AI in Engineering, Wydawnictwa Naukowo-Techniczne Warsaw,175-188.

Zhang L, Wang K, Shaw D, Miles J, Parmee I & Kwan A, 2006. Representation and its impact on topological search in Evolutionary Computation, Proc Joint Int. Conf on Comp & Decision Making in Civ & Building Eng (eds) Rivard H, Miresco E & Melhem H, Montreal Canada , 2359-2368.