From Meander Designs to a Routing Application Using a Shape Grammar to Cellular Automata Methodology

Thomas H. Speller, Jr.

Systems Engineering and Operations Research Department
Volgenau School of Engineering
George Mason University
Fairfax, VA 22030

The usefulness of a methodology that integrates shape grammar for capturing design information with cellular automata for computational output of a design solution space is demonstrated in this paper. The application domain is the ornamental artwork known as Chinese lattices or meanders, a subject of earlier interest in shape grammar studies. In this study, a specification for a Chinese lattice is used for creating a shape grammar to capture the model's rules of self-organization, which are then transcribed into cellular automata to physically generate a catalog of designs that meet the requirements of this particular meander style. Then, the study compares the use of a probabilistic (evolutionary computation) technique against complete enumeration for managing the search for unique designs. In consideration of the finding of a very large number of rule solutions for a design specification which produced only a very small number of graphically unique architectures, the question is raised as to whether a more efficient search process other than brute force enumeration can be used. Finally, the meander study is extended to a real engineering system, demonstrating the applicability of the shape grammar to cellular automata (SG $\rightarrow$ CA) methodology for finding the most efficient system architecture solutions for a comparable routing/circuit problem. System architectures addressing an underground heating specification are automatically generated and evaluated, resulting in a group of design alternatives displaying the best piping layouts for the given requirements.

1. Introduction

Meanders (see Figure 1) are decorative rectilinear or curvilinear patterns that were devised in antiquity for adornment of utilitarian objects [1–5]. The word is derived from the Meander River in ancient Greece (now Turkey) noted for its winding bends [1]. As with the original use of meanders, aesthetics is still an important feature of system architectures in today's world.

Intrigued by the meander style, Dye [6] was inspired to compose a two-volume collection on Chinese lattices entitled *A Grammar of Chinese Lattice*. Knight later studied meanders in depth as an architectural design domain [7, 8]. Knight's insight that there is a grammar capable of generating each lattice style led to the present study's adoption of meanders by which to demonstrate the application of shape grammar to cellular automata (SG → CA) [9] for an ornamental artwork, enabling computational generation of a catalog of designs in this style.

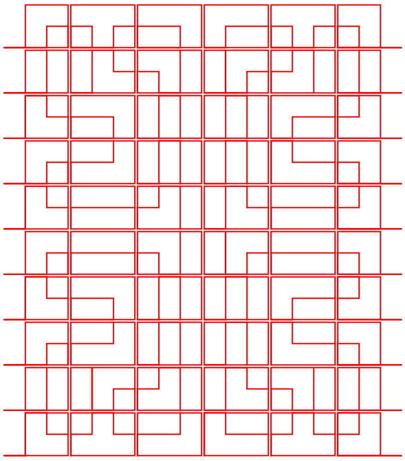


Figure 1. Example of a decorative meander from antiquity.

1.1 Shape Grammar

Since the design function is critical in system architecting, the ability to visually represent design forms and their physical rules of self-organization in a rigorous and also understandable manner is a necessity. A simple and versatile approach for achieving such computing with shapes is shape grammar, developed by Stiny and Gips [10] and based on formal languages as defined by the work of Chomsky [11]. A shape grammar is intended to express a formal system of rules for characterizing the structure of a design in a spatial language. "[D]esign is execution of a computation in a shape algebra to produce required shape information, and the rules of shape grammar specify how to carry out that computation. These rules encode knowledge of form, function, and the relationship of the two." [12, p. 238] In fact, while the past practice of shape grammar has focused on form, functions and properties of shapes can be included in the grammar [13, 14].

Shape grammar is thus an intuitive but also technically precise methodology that generates languages of design to allow visualization of the possible form and function outputs of the given rule inputs. A shape grammar consists of a vocabulary of shapes as well as markers (spatial variables) that control the positioning of shapes in the vocabulary. Shape rules created by an "architect" are applied recursively starting with the initial shape, generating designs that compose a grammatically correct language. The shape grammar utilizes such operators as shape addition, subtraction, and deletion. Furthermore, formally defined operations called transformations change the current state of the design through the operators of translation, rotation, reflection, and scaling. A production set of rules incorporating a defined sequence of operators and transformations evolves the initial condition shape step by step to a final state. Operators and transformations and their combinatoric compositions can yield tremendous diversity and also lead to changes in styles.

To date, there have been a few demonstrations of shape grammars in product design [15–17], but the hand application of shape grammars has proved a limitation to their use. Stiny has pointed out (via personal communication) that the practical use of shape grammar has been limited because of the current lack of a "robust" compiler or interpreter, leaving laborious hand manipulation or the development of custom software that may have narrow generalizability (recent attempts at shape grammar programs include [18–22]). Therefore, computational machinery that would facilitate the use of shape grammar across a variety of architectural applications would be of great value.

1.2 Cellular Automata

One promising approach for addressing the aforementioned shape grammar deficiency is through the use of cellular automata. Cellular automata, which act on sets of rules for defined neighborhood conditions, developed with the origins of computers and from the desire to model nature's self-generative capabilities, which are mainly attributable to von Neumann [23, 24, pp. 79–82], Ulam and Zuse [25], and Wiener and Rosenblueth [26]. Noting that simple rules and programs can create complex systems [27, 28], Wolfram began to use cellular automata as a model to study complexity and better understand nature. Capable of simulating complex system dynamics, a cellular automaton is described as a parallel processing computation machine with a neighborhood of finite state conditional rules that can model physics in time and space. The state of a discrete static or dynamical system at the next step function is determined by the collective action of these rules, which recursively evolve the system's state.

The cellular automaton neighborhood appears as a lattice of cells, and every rule is represented by a pattern of cells containing certain if–then conditional values (i.e., if a particular configuration of values is present in the lattice at a given step in time, then a certain value is entered in a specified cell for the next step). A cellular automaton is highly restricted; every possible local neighborhood configuration must have a rule that determines its reaction, and the range of this reaction is always the same. Consisting of a logical computation or pattern match (as to a list structure, see [29, 30]), a rule is thus applied to each empty cell based on the values in the surrounding cells of its defined neighborhood to determine the given cell's value at the next step. Because the cellular automata are expressed as a deterministic finite state system, it is possible to perform computations on a PC with single or multicore processors. Therefore, simulations can be run on a PC essentially without losing any emulation power to represent real systems operating as parallel processing systems.

The practical utilization of cellular automata has been hindered due to the difficulty of finding the proper rules to exhibit a target behavior and by the level of abstraction required to conceptualize the functioning of the cellular automata. Practitioners have expressed uncertainty as to how to apply cellular automata to modeling because they have confronted the immense difficulty of trying to find a set of rules from an astronomically large rule space [10, 31] that would generate a legitimate desired system architecture. The search for cellular automata rules has also been addressed by trial and error, genetic algorithms [32, 33], and genetic programming [34] with varying success.

Clearly, a cellular automata methodology for complex system modeling carries the advantage of parallel processing for large datasets with minimal overhead [33] and using local neighborhood interactions. Additionally, this algebraic, combinatoric, and logical approach permits the use of symbolic variables and functional operations according to specified rules. Cellular automata can be programmed (such as by means of the general programming language *Mathematica*® [30]) for generating output from an initial condition once the desired rules are defined.

1.3 Shape Grammar and Cellular Automata

Both shape grammar and cellular automata serve as clear models for bottom-up system architecting since they utilize basic elements and rules to transform and assemble them based on current state patterns in a given space (neighborhood). The opportunity thus exists for mapping a visually depicted form-function (system architecture) directly into the cellular automata, which then can generate the output in a visual-spatial format as a designer would produce. As the input methodology, the shape grammar expresses the forms and their relationships as well as any physical properties or physical laws of the form-function. The shape grammar production set of sequenced rules is then transcribed into cellular automata rules with the appropriate conditional neighborhoods defined, and the cellular automata computational machinery outputs the design space.

2. Developing a Design-Generating System to Describe a Meander Style

2.1 The Specification

The particular design specification (set of requirements) for the meander in this study (referring to Figure 2) entailed an initial condition that consisted of a {5 rows, 4 columns} matrix dimension, with the requirement that the meander enter only at the upper right matrix cell at {row 1, column 4}, traverse through every cell of the grid without reentering any cell, and exit only at the bottom left matrix cell at {5,1}. The meander could either traverse continuously through every cell in the grid, or meander "islands" would form to assure every cell had been contacted (discontinuities of the meander path were allowed). Then, symmetry reflection rules were applied to flip this {5,4} grid (which constitutes the upper left-hand quadrant of the lattice) to create the remaining three quadrants of the complete lattice {10 rows, 8 columns}.

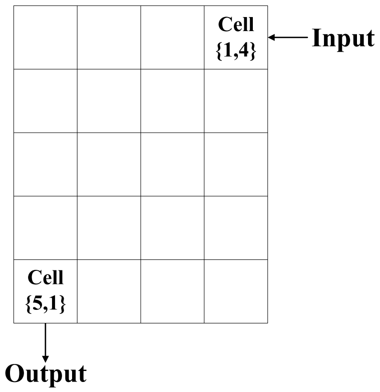


Figure 2. The specification indicating the input and output cell positions.

2.2 The Meander Shape Grammar

To derive the set of shape variables, meanders were hand-drawn against a grid background for decomposition into primitive line forms that would permit line connections from cell to cell, either straight or a 90° turn. Then, taking into account the direction of the cellular automaton processing, the local neighborhood conditions were defined. Each cell output could be determined in this case by two neighboring input cells, immediately above and immediately to the left of the cell to be computed (see Figure 3). Shape rules to generate outputs for every combination of neighborhood input shapes were enumerated. This step necessitated consideration of what boundary conditions were required per the specification and, consequently, what additional rules for the constrained outputs were needed. Marker variables were designed to serve as switches for turning on other rules in order to change or limit the functioning of the meander at the sites of the imposed boundary conditions (such as to prevent any outer cell from containing a shape that would exit the lattice). The markers also served to provide a perimeter line construction surrounding the complete lattice. Of note, the addition of these marker constraints technically expanded the {5,4} matrix in Figure 2 to {6,5} (shown in Figure 4).

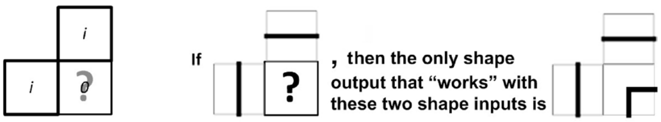
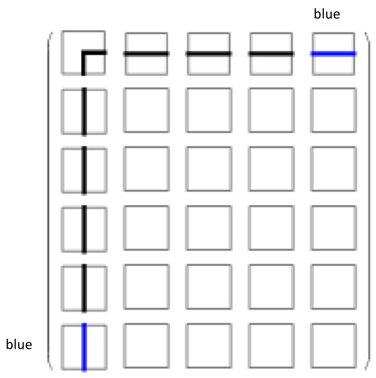

i
i
0
?
If
?
, then the only shape output that "works" with these two shape inputs is

Figure 3. The meander shape grammar neighborhood.



Graphic

$$\begin{pmatrix} \texttt{s3m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2b} \\ \texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\ \texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\ \texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\ \texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\ \texttt{s1b} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \end{pmatrix}$$

Symbolic

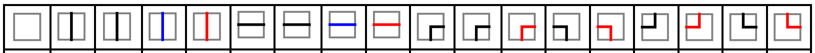
Figure 4. The initial condition in graphic and symbolic form.

The shape grammar was developed to express a meander system in a format that directly lent itself to a cellular automaton derivation. Transcribed simple rules and markers were carried over in the cellular automaton steps to route the meander primitives (the shape variables) in only physically legitimate directions within boundary constraints. However, certain situations called for a rule to impose a restriction on the meander to indicate that a neighborhood was illegitimate because it would not satisfy the specification. Neighborhoods that deviated from the specification were explicitly assigned a "halt" rule, which was intended to stop the execution of the cellular automaton from further generation. Finally, a group of rules was developed to rewrite the marker variables as a constant into their cell of origin during every application of the cellular automaton step function. Such rules served as an identity function when it was desired that certain cell values (the boundary delimiters) not be changed by the main rule set.

The shape variables required for the meander shape grammar include the following.

- 1 empty shape (i.e., blank cell in lattice not yet computed), symbolically {s0}
- 6 predetermined shape variables, symbolically {s1, s2, s3, s4, s5, s6}
- 11 shape markers, symbolically {s1m, s1b, s1r, s2m, s2b, s2r, s3m, s3r, s4r, s5r, s6r}

(The __m designates a marker that constrains the top row and left-hand boundaries. The __b represents a blue marker to bound the upper right or lower left corners of the quadrant. The __m and __b markers are used in the initial condition. The __r or red markers are rule generated, serving as boundaries for the bottom row and the right-hand side of the quadrant.)



s0	s1	s1m	s1b	s1r	s2	s2m	s2b	s2r	s3	s3m	s3r	s4	s4r	s5	s5r	s6	s6r

Shape and marker variables with their symbolic equivalents.

A rule set for a particular solution attempt is composed of 128 rules (if–then conditions) drawn from the two groups of meander rules. The formal simple relationships of form-function are symbolically expressed according to local neighborhood conditions, classified as Group I rules when the condition is invariant or as Group II rules when options exist for the condition (see Appendix A for listing). Group I contains 106 rules, each of which invariably results in just one possible cell output, so all of these rules are always in effect. Group II includes 22 rules also always in effect, but each has two alternative outcomes (note Figure 6(b) and (c)). A combinatoric table was constructed for the Group II rules to produce each possible listing of outcomes to be in effect {1 or 2} across all 22 rules. Each row in this combinatoric table would therefore consist of the same 22 rules with either the first or the second output option, and all the rows together would present every possible combination of outcomes across the 22 rule sequence. Each of the possible rule sets was then selected one at a time from the Group II rule combinatoric table and appended to the 106 Group I rules to generate another meander by the cellular automaton. The combinatoric variety of Group II rule alternatives resulted in a creative space of 4 194 304 possible meanders using $2^{22} = 4\,194\,304$ rule sets; each rule set thus generates a possible, although not necessarily unique, meander.

The initial condition configuration is shown in Figure 4 with the meander line shapes transcribed into letter-number symbols for use with the two-dimensional cellular automaton generating machine described in the following section. This initial condition is composed of the 6 row by 5 column lattice with boundary defining markers (marker shapes) and empty lattice cells (empty shapes). The postgeneration condition (shown in Figure 5) will still contain the original boundary markers, the generated meander line shapes, and additional marker shapes generated for the last row and last column. These generated markers act to further constrain the meander from traversing outside of the right-hand side and bottom of the matrix except at the specified input/output cells.

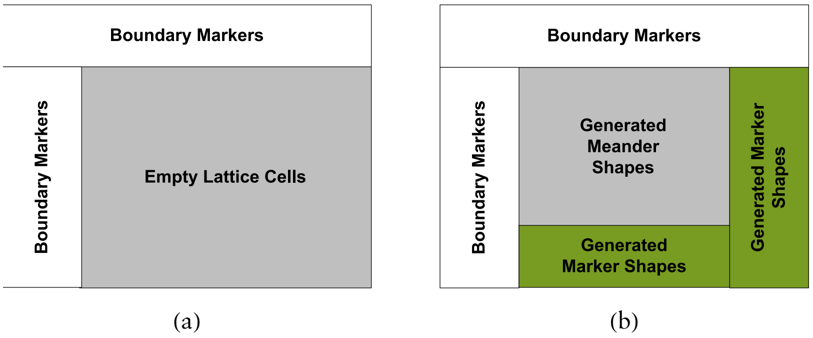


Figure 5. Lattice at (a) the initial condition and (b) after meander generation.

2.3 Applying the Cellular Automata Computational System to Generate a Set of Meanders

To enable the meander shape grammar rule format, a Moore two-dimensional neighborhood template was used [35, 36], with cells of the neighborhood made active by assigning 1 to the controlling cells and 0 to the inactive (non-influential) cells (see Figure 6(a)). Every cell in the step (s + 1) lattice has a conditional neighborhood from step (s) that determines the value of the center cell for step (s + 1). With respect to the meander construction, generation of each next step's meander shapes (at the central cell of the Moore neighborhood matrix) would be determined according to what meander shapes were already just above it as well as to its left (these are the two active or influencing neighbors). Figures 6(b) and (c) illustrate how Group I and Group II rules differ within this neighborhood.

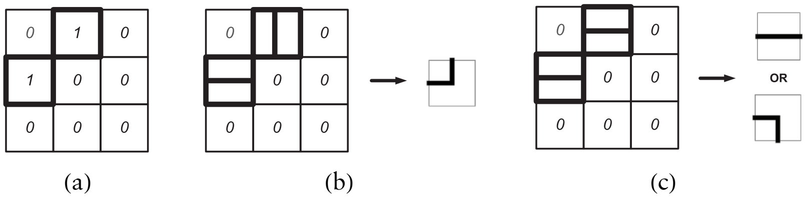
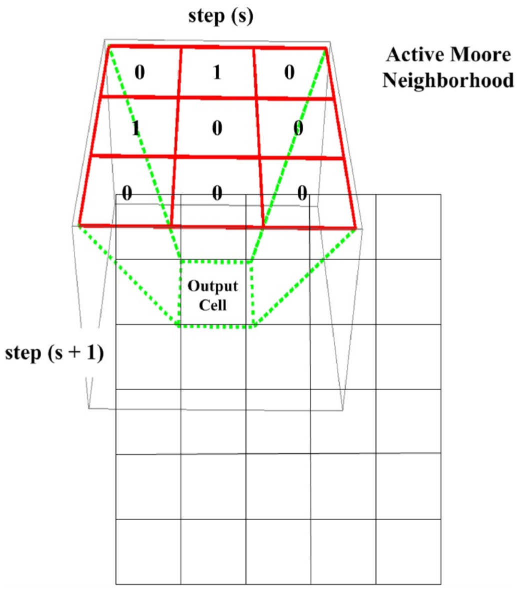


Figure 6. The Moore neighborhood rules: (a) Moore active neighborhood, (b) Group I rule resulting in a single output, and (c) Group II rule resulting in two possible outputs.

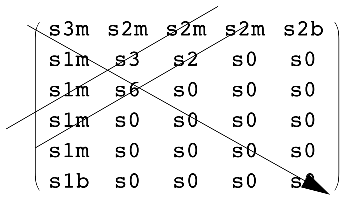
The {5,4} meander of this study was generated in eight steps, where each step applied the neighborhood rules to every cell in the lattice as a parallel processing finite state machine (Figure 7(a)). Since the cellular automata computation may not be intuitively obvious, Figure 7(b) shows the state of the matrix after step 1 using the selected rule set 3 for this example (from the initial condition shown in Figure 4). Step 1 seems to indicate that only cell {2,2} changed to s3. Actually, all cells were replaced according to their previous (s) state neighborhood conditions and with corresponding rules applied.



(a)

$$\begin{pmatrix} \texttt{s3m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2b} \\ \texttt{s1m} & \texttt{s3} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\ \texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\ \texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\ \texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\ \texttt{s1b} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \end{pmatrix}$$

(b)



$$\begin{pmatrix} \texttt{s3m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2b} \\ \texttt{s1m} & \texttt{s3} & \texttt{s2} & \texttt{s0} & \texttt{s0} \\ \texttt{s1m} & \texttt{s6} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\ \texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\ \texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\ \texttt{s1b} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \end{pmatrix}$$

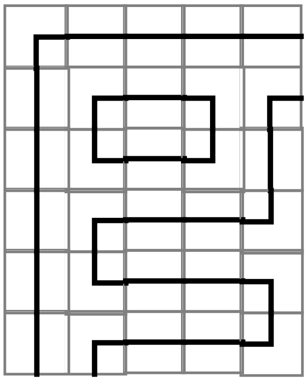
(c)

Figure 7. (a) Cellular automata as a parallel processing finite state machine, (b) step 1, and (c) step 2 and the cellular automata build direction.

Step 2 in Figure 7(c) shows that two s0 have been replaced by s6 and s2 in the matrix. Also shown is the direction of the build process by the cellular automaton function, starting at the upper left corner and progressing diagonally down the matrix in the eight successive steps. The next six steps are illustrated in Figure 8.

$\begin{pmatrix} \mathtt{s3m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2b} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s4} & \mathtt{s0} \\ \mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s0} & \mathtt{s0} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s0} & \mathtt{s0} & \mathtt{s0} \\ \mathtt{s1m} & \mathtt{s0} & \mathtt{s0} & \mathtt{s0} & \mathtt{s0} \\ \mathtt{s1b} & \mathtt{s0} & \mathtt{s0} & \mathtt{s0} & \mathtt{s0} \end{pmatrix}$	$\begin{pmatrix} \mathtt{s3m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2b} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s4} & \mathtt{s3r} \\ \mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s5} & \mathtt{s0} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s0} & \mathtt{s0} \\ \mathtt{s1m} & \mathtt{s6} & \mathtt{s0} & \mathtt{s0} & \mathtt{s0} \\ \mathtt{s1b} & \mathtt{s0} & \mathtt{s0} & \mathtt{s0} & \mathtt{s0} \end{pmatrix}$	$\begin{pmatrix} \mathtt{s3m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2b} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s4} & \mathtt{s3r} \\ \mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s5} & \mathtt{s1r} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s2} & \mathtt{s0} \\ \mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s0} & \mathtt{s0} \\ \mathtt{s1b} & \mathtt{s3r} & \mathtt{s0} & \mathtt{s0} & \mathtt{s0} \end{pmatrix}$
Step 3	**Step 4**	**Step 5**
$\begin{pmatrix} \mathtt{s3m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2b} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s4} & \mathtt{s3r} \\ \mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s5} & \mathtt{s1r} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s2} & \mathtt{s5r} \\ \mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s2} & \mathtt{s0} \\ \mathtt{s1b} & \mathtt{s3r} & \mathtt{s2r} & \mathtt{s0} & \mathtt{s0} \end{pmatrix}$	$\begin{pmatrix} \mathtt{s3m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2b} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s4} & \mathtt{s3r} \\ \mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s5} & \mathtt{s1r} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s2} & \mathtt{s5r} \\ \mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s2} & \mathtt{s4r} \\ \mathtt{s1b} & \mathtt{s3r} & \mathtt{s2r} & \mathtt{s2r} & \mathtt{s0} \end{pmatrix}$	$\begin{pmatrix} \mathtt{s3m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2b} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s4} & \mathtt{s3r} \\ \mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s5} & \mathtt{s1r} \\ \mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s2} & \mathtt{s5r} \\ \mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s2} & \mathtt{s4r} \\ \mathtt{s1b} & \mathtt{s3r} & \mathtt{s2r} & \mathtt{s2r} & \mathtt{s5r} \end{pmatrix}$
Step 6	**Step 7**	**Step 8**

(a)



(b)

Figure 8. (a) Steps 3–8 symbolically and (b) step 8 graphically.

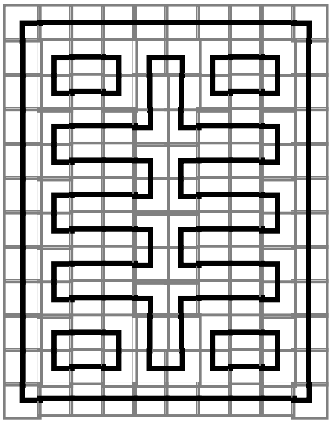
Once each {5,4} matrix was generated, any meander with a halt condition was programmatically removed, while a successful meander was compared to the file of saved meanders and added if unique. Finally, all markers were erased, the {5,4} grid of each complete unique solution (the upper left-hand quadrant of the lattice) was quadrupled by applying symmetry reflection rules, and the shape symbols were translated back to the actual meander line shapes to provide graphical output (see Figure 9).

2.4 The Solution Space of Meander System Architectures

As indicated in Figure 10, there were 841 693 rule sets that generated meander solutions satisfying the entrance/exit requirements and the requirement that a line pass through every cell. These meander designs accounted for only 20% of the total enumerated creative space of both successful and faulty meanders generated by the combinatoric table of rule sets (determined by the two possible outcomes of the 22 rules in the Group II rule set, 2^{22}). Furthermore and more startling, out of the large solution set, only 20 unique meander designs were found (shown in Figure 11). In genomic terms, a very large number of surviving genotypes expressed themselves through a very small number of phenotypes. Thus, an extremely large number of rule sets provided the same solution.

$$\begin{pmatrix}
\mathtt{s3m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2m} & \mathtt{s2b} & \mathtt{s2} & \mathtt{s2} & \mathtt{s2} & \mathtt{s2} & \mathtt{s4} \\
\mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s4} & \mathtt{s3r} & \mathtt{s4} & \mathtt{s3} & \mathtt{s2} & \mathtt{s4} & \mathtt{s1} \\
\mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s5} & \mathtt{s1r} & \mathtt{s1} & \mathtt{s6} & \mathtt{s2} & \mathtt{s5} & \mathtt{s1} \\
\mathtt{s1m} & \mathtt{s3} & \mathtt{s2} & \mathtt{s2} & \mathtt{s5r} & \mathtt{s6} & \mathtt{s2} & \mathtt{s2} & \mathtt{s4} & \mathtt{s1} \\
\mathtt{s1m} & \mathtt{s6} & \mathtt{s2} & \mathtt{s2} & \mathtt{s4r} & \mathtt{s3} & \mathtt{s2} & \mathtt{s2} & \mathtt{s5} & \mathtt{s1} \\
\mathtt{s1b} & \mathtt{s3r} & \mathtt{s2r} & \mathtt{s2r} & \mathtt{s5r} & \mathtt{s6} & \mathtt{s2} & \mathtt{s2} & \mathtt{s4} & \mathtt{s1} \\
\mathtt{s1} & \mathtt{s6} & \mathtt{s2} & \mathtt{s2} & \mathtt{s4} & \mathtt{s3} & \mathtt{s2} & \mathtt{s2} & \mathtt{s5} & \mathtt{s1} \\
\mathtt{s1} & \mathtt{s3} & \mathtt{s2} & \mathtt{s2} & \mathtt{s5} & \mathtt{s6} & \mathtt{s2} & \mathtt{s2} & \mathtt{s4} & \mathtt{s1} \\
\mathtt{s1} & \mathtt{s6} & \mathtt{s2} & \mathtt{s2} & \mathtt{s4} & \mathtt{s3} & \mathtt{s2} & \mathtt{s2} & \mathtt{s5} & \mathtt{s1} \\
\mathtt{s1} & \mathtt{s3} & \mathtt{s2} & \mathtt{s4} & \mathtt{s1} & \mathtt{s1} & \mathtt{s3} & \mathtt{s2} & \mathtt{s4} & \mathtt{s1} \\
\mathtt{s1} & \mathtt{s6} & \mathtt{s2} & \mathtt{s5} & \mathtt{s6} & \mathtt{s5} & \mathtt{s6} & \mathtt{s2} & \mathtt{s5} & \mathtt{s1} \\
\mathtt{s6} & \mathtt{s2} & \mathtt{s2} & \mathtt{s2} & \mathtt{s2} & \mathtt{s2} & \mathtt{s2} & \mathtt{s2} & \mathtt{s2} & \mathtt{s5}
\end{pmatrix}$$

(a)



(b)

Figure 9. Step 8 example: final matrix for rule set 3 (a) after reflection and (b) graphically.

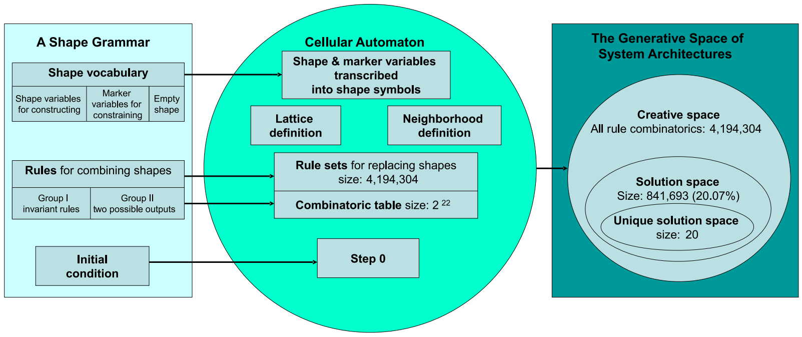

A Shape Grammar
Shape vocabulary
Shape variables for constructing
Marker variables for constraining
Empty shape
Rules for combining shapes
Group I invariant rules
Group II two possible outputs
Initial condition
Cellular Automaton
Shape & marker variables transcribed into shape symbols
Lattice definition
Neighborhood definition
Rule sets for replacing shapes size: 4,194,304
Combinatoric table size: 2 22
Step 0
The Generative Space of System Architectures
Creative space
All rule combinatorics: 4,194,304
Solution space
Size: 841,693 (20.07%)
Unique solution space
size: 20

Figure 10. Generation of a meander system architecture using the SG → CA approach to satisfy a specification.

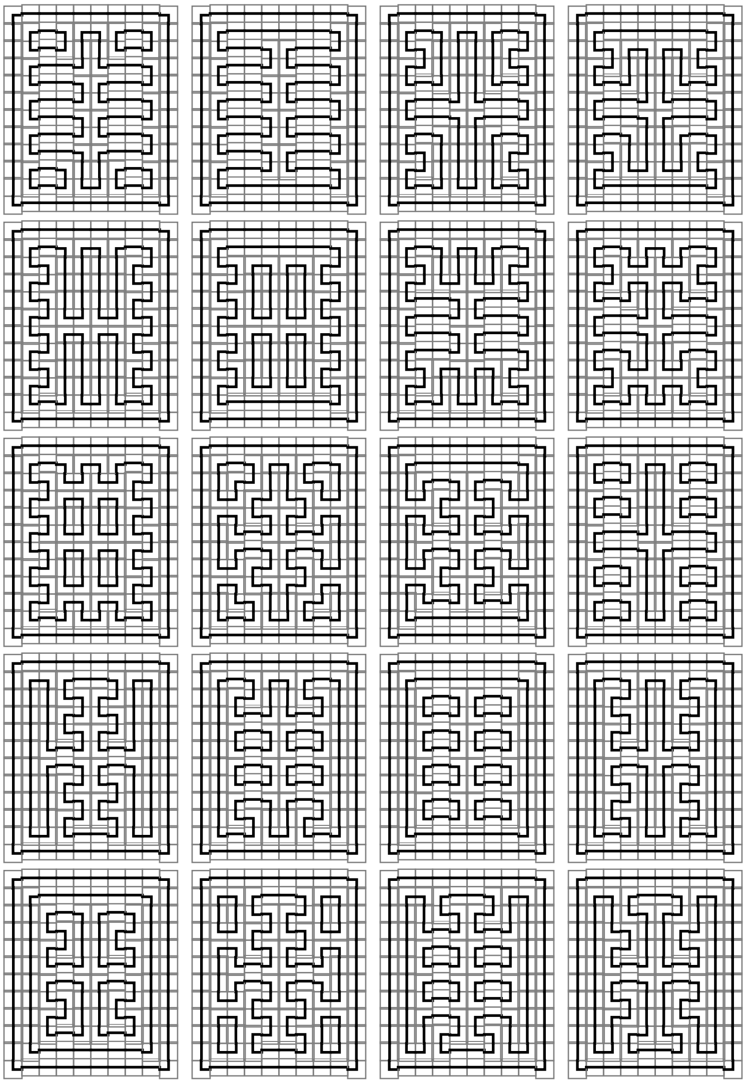


Figure 11. The 20 unique solutions satisfying the specification (depicted in lattice form).

One might conclude that taking a sampling approach to finding the meander solutions from such an enormous creative space could potentially miss finding a significant percentage of the unique solutions, thus demanding a complete enumeration of design possibilities. The question arises, however, as to how a probabilistic approach might compare to the enumerative approach just employed in more efficiently finding all the unique solutions within the meander creative space. Would an evolutionary computation technique, for example, take less time to find the unique solutions than the brute force enumeration used here?

3. Comparing Enumerative versus Probabilistic Methods for Finding All Unique Solutions in a Large Design Space of Architectures

It was observed that unique solutions numbered 1 through 12 were almost uniformly distributed throughout the creative space (see Table 1) while the remaining eight solutions were located in narrow zones. If there are only 20 unique graphic solutions, might there be a more efficient and still effective method to locate these 20 out of an enormous possibility space? Enumeration, obviously, guarantees finding all unique solutions but at a significant cost in computing power and time.

In view of the distribution of unique solutions detailed in Table 1, drawing samples of solutions from the total creative space of rule set generations to search for the unique designs calls for an approach that will either look in the "right" bin segments of the distribution or will cast widely enough to counter blindness to the actual asymmetrical frequency topology.

3.1 Probabilistic Approaches for Searching the Creative Space

To address the question about the search approach, three probabilistic trials were designed employing evolutionary computation techniques: random sampling of the creative space to search for solutions (comparable to 100% mutation), a genetic algorithm using the selection and recombination (crossover) operators only, and a genetic algorithm varying in sample size using selection, recombination (crossover), and mutation operators.

One rule set drawn from the total group of possible rule sets generating a meander solution (2^{22} combinatoric rule set versions) served as the original genome for the evolutionary computation, with genomic variation occurring due to Group II rules having variable outcomes. The halt condition that was intended to stop the execution of the cellular automaton from further generation served as the analogy for a genetic flaw in the resultant phenotype. A fitness function totaled the number of halts or genetic flaws in the complete phenotypes generated. The objective was to reduce the number of genetic flaws to zero by applying the operators of a random sample population: selection, pairing, crossover, and mutation [37, 38]. The advance knowledge that there were 20 unique solutions was used to bring the evolutionary computation to a halt once the 20 unique solutions were found. For each probabilistic trial described below, the total number of samples and the time required for the computation were collected for 10 independent runs of each algorithm. The comparative data and summaries are contained in Table 2.

Unique Solution	**Total**	**As % of Solution Space**	**{1,300}***	**{301,600}**	**{601,900}**	**{901,1200}**	**{1201,1500}**	**{1501,1800}**	**{1801,2100}**	**{2101,2400}**	**{2401,2700}**	**{2701,3000}**	**{3001,3300}**	**{3301,3600}**	**{3601,3900}**	**{3901,4200}**
1	36864	4.38%	4096	2688	1408	4096	0	4096	180	3916	5728	2464	4096	0	4096	0
2	73728	8.76%	8192	2688	5504	5368	2824	8056	316	8012	5728	10656	5548	2644	8192	0
3	33592	3.99%	2352	2336	2352	2336	2352	2336	2352	2336	3136	3888	2352	2336	2352	776
4	67192	7.98%	4688	4688	4688	4688	4688	4688	4688	4688	6256	7804	4688	4688	4688	1564
5	16792	2.00%	1184	1168	1168	1184	1168	1168	1184	1152	1568	1936	1168	1184	1168	392
6	33592	3.99%	2368	2336	2336	2368	2320	2352	2352	2320	3152	3868	2340	2364	2316	800
7	17024	2.02%	1328	1232	1024	1184	1324	1076	1048	1320	1728	1664	1204	1324	1056	512
8	67200	7.98%	4672	4700	4676	4700	4676	4696	4680	4696	6248	7828	4684	4692	4684	1568
9	33600	3.99%	2336	2364	2340	2336	2336	2360	2344	2336	3104	3924	2348	2336	2336	800
10	67072	7.97%	4608	4608	4608	4728	4744	4736	4736	4608	6144	7680	4778	4798	4760	1536
11	33280	3.95%	2560	2048	2560	2168	2440	2432	2176	2560	2560	4096	2218	2390	2560	512
12	65536	7.79%	4096	4096	4096	5368	4872	6008	4232	4096	4096	8192	5548	4692	6144	0
13	65536	7.79%	0	0	0	18928	37500	9108	0	0	0	0	0	0	0	0
14	29768	3.54%	0	0	0	2684	4484	5050	4166	0	0	0	2774	4394	5120	1096
15	29768	3.54%	0	0	0	2684	4484	5052	4164	0	0	0	2774	4394	5120	1096
16	59464	7.06%	0	0	0	4732	9374	9376	9286	0	0	0	4820	9376	9374	3126
17	29765	3.54%	0	0	0	2683	4484	5048	4167	0	0	0	2772	4396	5120	1095
18	32768	3.89%	0	0	0	4336	9488	2560	0	0	0	0	4436	9596	2352	0
19	16384	1.95%	0	0	0	0	0	0	0	0	0	0	5546	8790	2048	0
20	32768	3.89%	0	0	0	0	0	0	0	0	0	0	9642	18750	4376	0
	841693	**100.00%**	42480	34952	36760	76571	103558	80198	52071	42040	49448	64000	73736	93144	77862	14873
Percentage of Total Solutions			5.05%	4.15%	4.37%	9.10%	12.30%	9.53%	6.19%	4.99%	5.87%	7.60%	8.76%	11.07%	9.25%	1.77%

* Bin range of rule sets for generating solutions from 1 to 300,000; subsequent bins range in thousands.

Table 1. Frequency distribution for valid meander solutions and the unique graphic solutions across the creative space.

Trial Runs	1	2	3	4	5	6	7	8	9	10	**Mean**	**Standard Deviation**
Random Selection of Rule Sets												
number of rule sets sampled to find 20 unique architecture solutions	411	414	885	402	249	706	361	876	651	540	549.500	220.902
CPU time (sec)	18.079	17.766	38.688	17.984	10.922	32.219	16.703	40.11	30.25	25.219	24.794	10.063
Crossover at 50%, No Mutation												
number of rule sets sampled to find 20 unique architecture solutions	90	170	140	120	140	180	140	100	190	210	148.000	39.101
CPU time (sec)	32.265	63.891	51.672	43.813	51.921	68.079	52.36	35.75	73.422	80.266	55.344	15.870
Crossover at 50% and Mutation at 4.5%												
number of rule sets sampled to find 20 unique architecture solutions	70	80	80	70	160	100	170	90	130	180	113	43.218
CPU time (sec)	23.75	27.843	27.719	23.562	61.359	35.922	63.531	32	47.562	67.187	41.044	17.349
								Mean of means for time (sec)			40.394	
Enumerative												
all rule sets tested to find 20 unique architecture solutions											4,194,304	
CPU time											41 hours	

Table 2. Search enumeration and evolutionary computation techniques—comparison data and statistical summaries.

The first trial employed purely random sampling of the creative space. In each run, meanders were generated one at a time from the sampled rule sets and examined automatically to see if each one was a solution (no halts existed). If valid, then each solution was match tested against a building file of unique solutions to determine if it was a duplicate, and if not, it was added to the unique solution file. The algorithm generates up to 1000 samples within a run randomly selected from the total rule set (size 2^{22}) or stops upon finding the 20 unique architectural solutions, whichever occurs first. In 20 independent runs (only 10 runs are shown in Table 2), the algorithm found the 20 unique solutions each time before reaching 1000 rule set samples. This first trial is considered a baseline for comparison.

A second trial was conducted using 50% selection and 50% crossover operators with no mutation operator. An initial run sampled 10 genomes or rule sets selected randomly, and the resulting meanders were all generated and examined to record any unique solutions. Then, the meanders were rank sorted by halt (genetic flaw) count from low to high. The 50% of the initial population sample, five in this case, with the fewest halts or genetic flaws was selected for random pairing with another five randomly selected rule sets. For each pair, the genomes were crossed at the halfway point, and the phenotypes of this new sample of meanders were generated and tested for the presence of unique solutions. The top 50% (five) of this new generation group with the fewest halts were selected and paired with another five randomly selected genomes. This process continued for up to 20 generations or until 20 unique solutions were discovered, whichever occurred first. As in the case of the random sampling, each run of this evolutionary computation algorithm found all 20 unique solutions without reaching the stopping point.

The third trial also began with an initial sample size of 10 genomes (rule sets) and followed the process just described in the second study except for the inclusion of a mutation factor (50% selection, 50% crossover, 4.5% mutation operators). For each crossed genome pair, one of the positions in their rule sequences was randomly mutated (the rule's right-hand option was switched to the alternative). The algorithm was similarly set to run up to 20 iterations or find the 20 unique solutions, whichever occurred first. Once again, all unique solutions were identified in each run before the program's limit. (It should be noted that in addition to this study of sample size 10, identical solution searches were conducted with sample sizes of six and 100 with similar results to the 10.)

3.2 Comparison of the Enumerative and Evolutionary Computation Approaches for Managing the Creative Space

All of the sampling techniques were particularly useful with regard to the topology of the solution space in this study. While the output contained a very small number of unique solutions, each of these solutions could be derived from a much larger range and number of different rule sets within the solution space, thus affording a reasonable target for sampling. The evolutionary computation techniques are astonishingly quick compared to enumeration. The time to find all 20 unique solutions in a creative space of approximately 4.2 million ranged from approximately 25 to 55 seconds depending on the operator settings for the evolutionary computations. In comparison, the fully enumerative approach required approximately 41 hours of CPU time. Clearly the probabilistic approach was far more efficient in finding the unique, successful meander designs from a total creative space that produced 20.07% successful but at the same time 99.99% duplicated solutions. However, this economy comes at the cost of information, namely that of the exact number of unique solutions (which enumeration does provide) and therefore a lack of assurance regarding when to stop searching for solutions.

4. An Application of the Meander Style to an Engineering Problem

By applying various constraints to a lattice, the meanders generated in this study could be interpreted as routing or circuitry scenarios and extended to a real-life application, in this case efficiently routing an underfloor heating system for different sized and shaped rooms. The algorithm development for designing efficient piping systems is a difficult problem due to the wide variation of possible designs. A simple pattern similar to one used by a manufacturer of underfloor heating systems [39] is shown in Figure 12. Interestingly, the pattern of pipes bears a resemblance to the Chinese lattices.

Underfloor heating systems are radiant heating systems using conduction, unlike radiators, forced air, and fireplace heating systems, which are convection heating systems. The underfloor heating is generated by hot water circulating in pipes routed beneath the floor surface. The heat uniformly emanates from the bottom upward, and the physical system is “invisible,” providing an aesthetic appeal. This type of heating system has become more popular in recent years, especially in Europe. Hot water furnaces fueled by gas, oil, or electricity heat the water that is pumped through manifold systems to different zones in a dwelling.

The pipe routing problem becomes one of system architecting. Even though the example in Figure 12 looks quite simple, the actual patterns of routings can be varied and complex. The location of the pump and manifold, the input/output positions interfacing with each room or heating zone, and the routing or meander pattern within each zone become a combinatoric routing or continuous circuit layout problem. There are many possible combinations of where to locate the inputs and outputs of the piping for each zone and where to place the manifold/pump. For purposes of this study, arbitrary specifications were defined with respect to these factors, in addition to the size and configuration of the spaces to be heated. Each zone can then be thought of as a subsystem within the total underfloor heating supersystem.

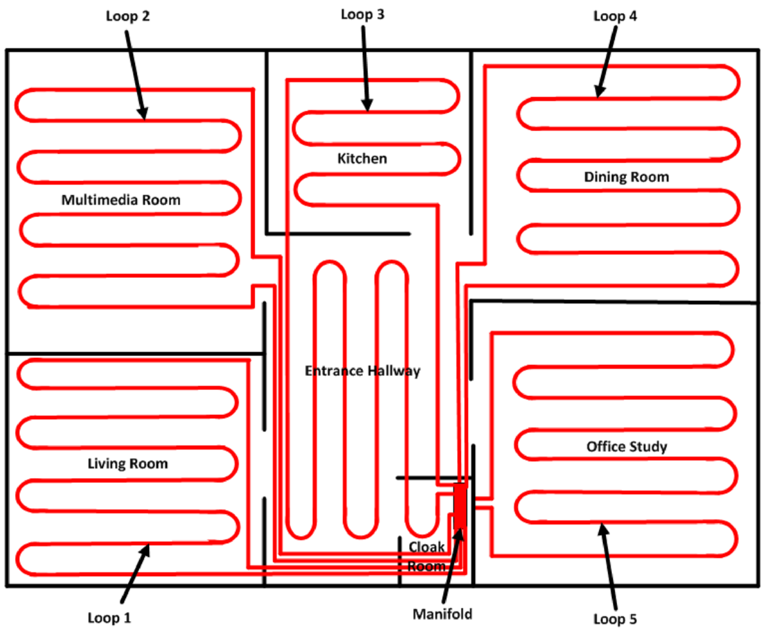


Figure 12. Example of a typical pipe layout.

The following specifications are proposed for an underfloor heating system design.

1. There are three zones as shown in Figure 13.
2. The manifold/pump may be located anywhere at the bottom of Zone 3.
3. From Zone 3, the input/output piping may connect anywhere to Zone 1, but may not connect to Zone 2 because there is a solid foundation wall between Zone 2 and Zone 3.
4. Zone 1 and Zone 2 interconnect.
5. The system architecture(s) generated must be efficient for providing uniform heat and be the lowest cost to manufacture and install.

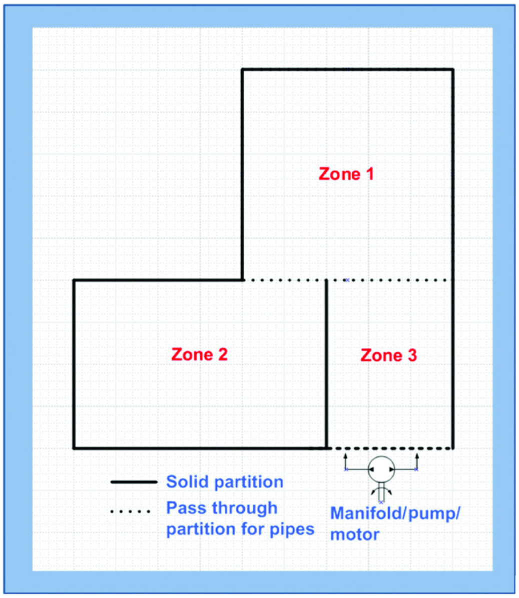


Figure 13. The room layout.

4.1 Developing the Design System to Describe the System Architecture

A shape grammar was developed to express an underfloor piping system in a format that directly lent itself to a cellular automaton derivation. Shape rules to generate outputs (the next pipe form to be used) for every combination of neighborhood input shapes (the current stage of piping configuration) were enumerated. To assure that the piping conformed to the specification, marker variables served as switches for turning on other rules in order to change or limit the choice of pipe forms at the sites of imposed boundaries. Transcribed simple rules and markers were then carried over in the cellular automaton steps to route the pipe elements in only physically legitimate directions within boundary constraints. A halt rule to stop the execution of the cellular automaton from further generation was assigned to prevent illegitimate piping configurations.

The shape grammar for the underfloor heating system was based on the meander shape grammar of the earlier study but required more marker variables for managing the zone barriers and interfaces, as well as more rules associated with these markers. A grid was superimposed over the entire floor area, which was partitioned into three room heating zones. The size of the grid cell controls the density of the pipe routing and thus the positioning of pipes to provide the most uniform heating. In addition, the pipe was required to traverse continuously through each cell in the grid, while meander “islands” that did not result in total grid connectivity would be rejected. Zone partitions were identified by marker shapes, which were later erased. As with the meander grammar, the same six basic pipe shapes were employed, but 37 shape markers and 432 rules were required for the more complex boundaries involved (see Appendix C).

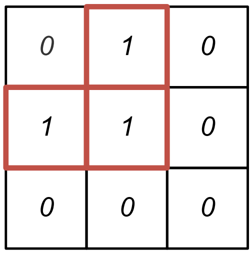


Figure 14. The local neighborhood definition.

4.2 Developing the Computational System for Generating the System Architecture

The underfloor piping Moore neighborhood as indicated in Figure 14 has three active conditional neighbors, now including the center cell that would be filled by one of the two possible initial condition empty shape variables, {s0} or {s0r}, to distinguish which shape rule output option to use. A particular solution attempt could draw from one of the same two groups of piping rules (Group I invariant rules and Group II rules having two right-hand options). Certain cells in each zone were restricted to a single rule output (Group I rules) for the initial condition because of boundary constraints. Initial condition configurations for the heating zones therefore were based on whether each cell was free or not to utilize rule set output options (whether either a Group I or Group II rule could be employed, or only a Group I rule could be used). Since Group II rules had two possible outputs, initial conditions could be varied according to a combinatoric enumeration of option choices for unrestricted cells. If a Group II rule was called for, that cell's initial condition value determined whether to use the first option {s0} or the second {s0r}. If a Group I rule was evoked in this same cell, then only its single output could be dictated. Thus, certain identical neighborhood configurations could result in either one of two possible outcomes, yielding a great combinatoric space, lattice-wide, of different configurations for possible local actions.

Zone 1 required a {6,6} dimension matrix and had an initial condition with 19 cells possessing two possible values, as shown in Figure 15. The 19 cell options led to a combinatoric space of size 2^{19}, or 524 288 different initial conditions. Both the first row and column contain marker variables and are not part of the heating zone itself.

$$\begin{pmatrix}
\texttt{s0m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2b} \\
\texttt{s1m} & \texttt{s0} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \texttt{s0} \\
\texttt{s1m} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \texttt{s0} \\
\texttt{s1m} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \texttt{s0} \\
\texttt{s1m} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \texttt{s0} \\
\texttt{s5m} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \{\texttt{s0}, \texttt{s0r}\} & \texttt{s0}
\end{pmatrix}$$

Figure 15. Initial condition options for Zone 1.

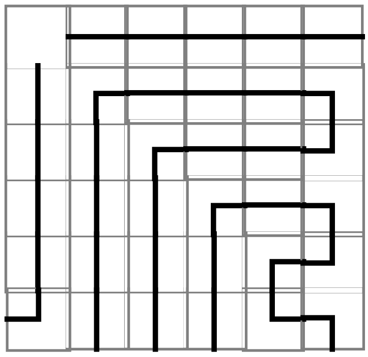
The 18th applied initial condition from all combinatoric initial conditions that could possibly be composed from Figure 15 did yield a solution, as shown in Figure 16.

$$\begin{pmatrix}
\texttt{s0m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2b} \\
\texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\
\texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\
\texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0} \\
\texttt{s1m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0r} & \texttt{s0} \\
\texttt{s5m} & \texttt{s0} & \texttt{s0} & \texttt{s0} & \texttt{s0r} & \texttt{s0}
\end{pmatrix}$$

initial condition

$$\begin{pmatrix}
\texttt{s0m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2b} \\
\texttt{s1m} & \texttt{s3} & \texttt{s2} & \texttt{s2} & \texttt{s2} & \texttt{s4r} \\
\texttt{s1m} & \texttt{s1} & \texttt{s3} & \texttt{s2} & \texttt{s2} & \texttt{s5r} \\
\texttt{s1m} & \texttt{s1} & \texttt{s1} & \texttt{s3} & \texttt{s2} & \texttt{s4r} \\
\texttt{s1m} & \texttt{s1} & \texttt{s1} & \texttt{s1} & \texttt{s3} & \texttt{s5r} \\
\texttt{s5m} & \texttt{s1n} & \texttt{s1p} & \texttt{s1} & \texttt{s6} & \texttt{s4r}
\end{pmatrix}$$

solution generated (symbolic)



solution generated (graphic)

Figure 16. The Zone 1 solution generated from the 18th initial condition possibility (with markers shown).

Zone 2 was a {5,7} dimension matrix, also with an initial condition of 19 cells possessing two possible values, as shown in Figure 17. The combinatoric size is the same as Zone 1, or 2^{19}. The graphical output of one possible solution for Zone 2 is depicted in Figure 18. Again, the first row and column serve as markers, including carrying over two cells {1,6} and {1,7} from a Zone 1 solution to show the presence of inputs/outputs.

$$\begin{pmatrix}
\texttt{s0m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2m} & \texttt{s2m} & \texttt{s1n} & \texttt{s1p} \\
\texttt{s1m} & \texttt{s0} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \texttt{s0} \\
\texttt{s1m} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \texttt{s0} \\
\texttt{s1m} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \texttt{s0} \\
\texttt{s1m} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \texttt{s0}
\end{pmatrix}$$

Figure 17. Initial condition options for Zone 2.

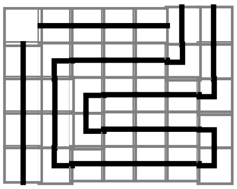


Figure 18. One of 37 solutions for Zone 2 (with markers shown).

Zone 3 was a {5,4} dimension matrix having an initial condition where eight cells possessed two possible values, shown in Figure 19. The cellular automaton generates $2^8 = 256$ matrices in six steps. The four possible solutions that resulted are shown in Figure 20. Marker variables for the first row boundary identified the input/output locations from Zone 1.

$$\begin{pmatrix}
\texttt{s0m} & \texttt{s3} & \texttt{s5} & \texttt{s1r} \\
\texttt{s1m} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \texttt{s0} \\
\texttt{s1m} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \texttt{s0} \\
\texttt{s1m} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \texttt{s0} \\
\texttt{s1m} & \{\texttt{s0, s0r}\} & \{\texttt{s0, s0r}\} & \texttt{s0}
\end{pmatrix}$$

Figure 19. Initial condition options for Zone 3.

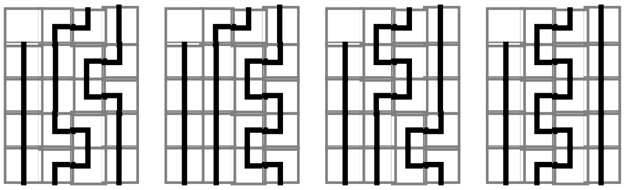


Figure 20. Zone 3 unique solutions (with markers shown).

4.3 Selecting the Most Efficient System Architectures from the Solution Space

The room heating zones were considered as subsystems or modules of the whole system to afford a divide-and-conquer strategy for computing the meanders for each zone [40]. The only interdependence of the zones is at their possible points of interconnection with the location of input/output pipes. After each zone solution was generated, a pattern-matching filter (comparable to a "fitness test") was applied to screen out solutions that would not meet the specification (i.e., would not interconnect with each other). Only unique, good solutions were stored. In a final operation, the three zones were combined so that the supersystem configuration catalog could be enumerated.

Once all the solutions for the complete underfloor heating system had been output, their individual efficiency, fitness, or measures of merit could be calculated and compared. For this study, the least action principle was invoked. Least action was interpreted as being represented by those solutions having the highest count of vertical and horizontal lines (straight pipe sections rather than bent pipe sections). The justification for this prescription is derived from three points.

- Straight pipes are easier and less costly to manufacture and install.
- Less pumping power (less energy) is required for straight sections.
- The circulation area for water and its radiant heat is considered more consistent for straight pipe configurations.

Zone 1 produced 34 solutions, Zone 2 had 37 solutions, and Zone 3 yielded four solutions. Combined, they produced a catalog of 5032 solutions. Nine randomly selected examples are shown in Figure 21. The count of straight lines as a fitness measure provided a least action group of 60 solutions (shown in Figure 22), giving stakeholders a more manageable group from which to choose a design. Computation time on a single processor PC for generating all solutions was extremely fast.

5. Discussion

This study was based on an analysis of the Chinese lattices presented in Dye's work [6]. A review of these artwork patterns raised the question as to whether the SG → CA approach could be applied to model the style of any particular Chinese lattice. It was relatively simple to visually analyze Chinese lattices to develop the vocabulary of shapes and then to construct a grammar of rules that assemble these shapes according to a meander specification. Programming the cellular automata to capture these shapes and rules to output a design catalog is not difficult with the appropriate training. While neither the use of a shape grammar or of cellular automata alone would have solved this study's presented problem, the combined SG → CA methodology was successful in this goal.

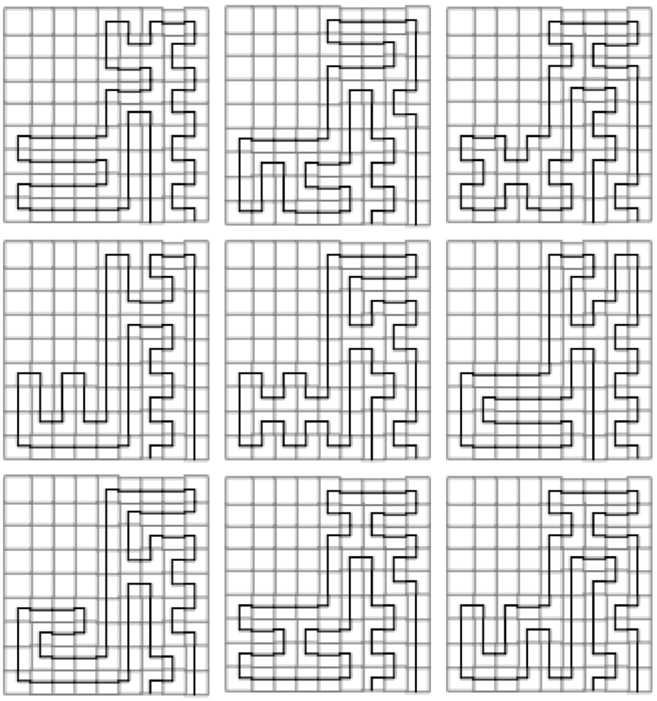


Figure 21. Nine samples taken from the 5032 piping solutions (markers erased).

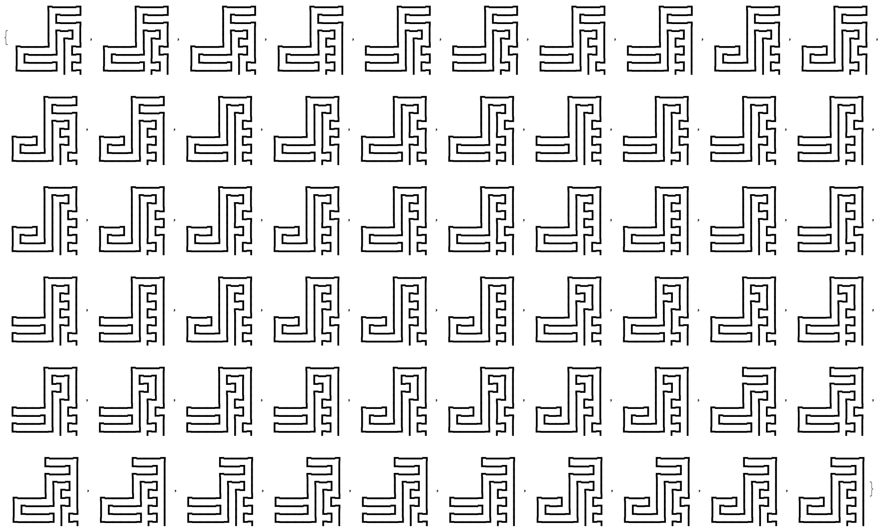


Figure 22. Least action group (grid and markers erased).

In demonstrating the problem-solving scenario with the meander design, the following observations were made. Combinatoric techniques greatly expanded the creative and solution spaces, and the use of simple or very basic primitives allowed a greater number of combinatoric opportunities to arise. Boundary constraints were critical within the rules set for ensuring the requirements for viable solutions; architectures are also very sensitive to the initial condition, which is itself a type of constraint.

Enumerative generation with the SG → CA meander design approach was able to catalog and account for all possible solutions. However, the enumerative search was relatively very slow and resulted in an overwhelming amount of useless information. On the other hand, the evolutionary computational search turned out to be quite useful with the large creative space, even with its lumpy distribution. The evolutionary computation search was extremely fast and required fewer samples than a purely random search. The most efficient search process combined the evolutionary computation features of both crossover and mutation, the latter of which introduces some randomness into the operation, thereby facilitating a wider search range.

However, there are no standard operator value settings (or best system architecture) for the different evolutionary computation methods, so a large variety of evolutionary computation designs could be applied. The evolutionary computation design and operator value settings are up to the experiential judgment and tinkering of the modeler. Furthermore, evolutionary computation is probabilistic—there is no guarantee of finding all the solutions or knowing exactly how many solutions are contained in the system architecture solution space. Therefore, there is no definitive way to determine when to halt the evolutionary computation algorithm.

The trade-off between computational efficiency (evolutionary computation) versus comprehensiveness of search results (enumerative generation) brings to mind the principle of satisfying the specification [41]. The system architect must decide if it is appropriate to use an enumerative approach such as to catalog multiple solutions, to identify all of the possible solutions (find all the needles in a haystack), or to obtain an understanding of the topology of the solution space within its rule set domain, as opposed to utilizing the very efficient sampling approaches provided by evolutionary computation methods when completeness or in-depth understanding is not of primary concern.

Finally, noticing the comparability of the meander design results to the requirements of an underground heating system, the meander SG → CA algorithm was extended to generate piping solutions for a given specification. Systems originally possessing sole aesthetic appeal were extended to the generation of systems with functional purpose in the satisfaction of a specification to solve a real-life problem. The basic form and rules of the aesthetic subject thus were generalized for application in a functional setting where flow was the design foundation in both.

In addition to the adaptation of aesthetics to function, this portion of the study demonstrated the rapid solution execution of a practical problem with a potentially large combinatoric space using the SG $\rightarrow$ CA methodology. To allow for manageable computation, the problem scenario was decomposed into separate but interfacing local zones (neighborhoods, modules), which then allowed for efficient solution development when the subsystem architectures were configured together by fit (intraconnectivity) into the complete system. Rules generated constraints that served as boundaries for channeling the creative process toward the intended purpose. For the underfloor circuit problem, the SG $\rightarrow$ CA approach highlighted the usefulness of bottom-up design analysis and synthesis for solving a problem that might have a solution space too large for human ease of management. An extension of this routing architecture methodology conceivably could be applied to other systems involving flow of information, energy, or things, such as for a factory layout with materials flow, power grid architecting, information networks, and circuit layouts.

6. Conclusion

The generation by hand of a complete catalog of all possible meander solutions to a given problem is extremely difficult in terms of time consumption and task management. Currently, no programs exist to undertake this graphic and combinatoric activity. The purpose of this paper was to show how the shapes and their rules for organization or assembly could be captured by a shape grammar, which in turn could be transcribed into symbols that would interact within cellular automata neighborhoods according to the rules. This interaction occurs across time steps, allowing neighborhood state changes to ultimately generate a complete design bottom-up. The general approach used for creating meander designs was applied to a similarly based circuit routing problem. Regardless of the search approach employed to find different legitimate meander designs, the shape grammar to cellular automata (SG $\rightarrow$ CA) methodology was able to undertake this design problem effectively and efficiently.

Using the methodology as exemplified herein, one can computationally explore a variety of design scenarios requiring bottom-up assembly of primitive elements according to a set of identifiable rules [42]. This approach has application potential for developing and testing theories, for creating a multitude of new designs for real-life situations, and for solving certain practical problems. The use of shape grammar thus can provide the system architect the assurance that design proceeds by rules embodying the relevant laws of nature or principles of physics. Harnessing cellular automata, the architect then may be able to generate a very large creative space of design candidates, selecting those that satisfy specific requirements (stability, robustness, aesthetics, cost, etc.) and thereby managing a possibly explosive design space.

Appendix

A. Meander Rule Set

This appendix lists the 128 rules: the formal simple relationships of form-function symbolically expressed according to local neighborhood conditions, composed of Group I and Group II combinations.

```
GroupI = {{s3m} → s3m, {s1m} → s1m, {s2m} → s2m,
   {s1b} → s1b, {s2b} → s2b, {s0, s0, 0} → s0, {s0, s2b, 0} → s0,
   {s0, s2m, 0} → s0, {s1, s1r, 0} → s1r, {s1, s2, 0} → s3,
   {s1, s2r, 0} → halt, {s1, s3r, 0} → s1r, {s1, s4r, 0} → s1r,
   {s1, s5, 0} → s3, {s1, s5r, 0} → halt, {s1, s6, 0} → s3,
   {s1b, s0, 0} → s0, {s1b, s6, 0} → s3r, {s1m, s0, 0} → s0,
   {s1m, s2m, 0} → s3, {s1m, s6, 0} → s3, {s1r, s1, 0} → s6r,
   {s1r, s2, 0} → halt, {s1r, s3, 0} → s6r, {s1r, s4, 0} → s6r,
   {s1r, s5, 0} → halt, {s1r, s6, 0} → halt, {s2, s1, 0} → s5,
   {s2, s1r, 0} → s5r, {s2, s2b, 0} → s2r, {s2, s2r, 0} → s4r,
   {s2, s3, 0} → s5, {s2, s3r, 0} → s5r, {s2, s4, 0} → s5,
   {s2, s4r, 0} → s5r, {s2, s5r, 0} → s4r, {s2r, s1, 0} → s5r,
   {s2r, s1r, 0} → s5r, {s2r, s2, 0} → s2r, {s2r, s3, 0} → s5r,
   {s2r, s4, 0} → s5r, {s2r, s4r, 0} → s5r, {s2r, s5, 0} → s2r,
   {s2r, s5r, 0} → halt, {s2r, s6, 0} → s2r, {s3, s1, 0} → s5,
   {s3, s1r, 0} → s5r, {s3, s2b, 0} → s2r, {s3, s2r, 0} → s4r,
   {s3, s3, 0} → s5, {s3, s3r, 0} → s5r, {s3, s4, 0} → s5,
   {s3, s4r, 0} → s5r, {s3, s5r, 0} → s4r, {s3r, s1, 0} → s5r,
   {s3r, s2, 0} → s2r, {s3r, s3, 0} → s5r, {s3r, s4, 0} → s5r,
   {s3r, s5, 0} → s2r, {s3r, s6, 0} → s2r, {s4, s1r, 0} → s1r,
   {s4, s2, 0} → s3, {s4, s2b, 0} → s3r, {s4, s2m, 0} → s3,
   {s4, s2r, 0} → halt, {s4, s3r, 0} → s1r, {s4, s4r, 0} → s1r,
   {s4, s5, 0} → s3, {s4, s5r, 0} → halt, {s4, s6, 0} → s3,
   {s5, s1r, 0} → s1r, {s5, s2, 0} → s3, {s5, s2r, 0} → halt,
   {s5, s3r, 0} → s1r, {s5, s4r, 0} → s1r, {s5, s5, 0} → s3,
   {s5, s5r, 0} → halt, {s5, s6, 0} → s3, {s5r, s1, 0} → s6r,
   {s5r, s1r, 0} → halt, {s5r, s2, 0} → halt, {s5r, s3, 0} → s6r,
   {s5r, s4, 0} → s6r, {s5r, s4r, 0} → halt, {s5r, s5, 0} → halt,
   {s5r, s5r, 0} → halt, {s5r, s6, 0} → halt, {s6, s1, 0} → s5,
   {s6, s1r, 0} → s5r, {s6, s2r, 0} → s4r, {s6, s3, 0} → s5,
   {s6, s3r, 0} → s5r, {s6, s4, 0} → halt, {s6, s4r, 0} → halt,
   {s6, s5r, 0} → s4r, {s6r, s1, 0} → s5r, {s6r, s1r, 0} → s5r,
   {s6r, s2, 0} → s2r, {s6r, s3, 0} → s5r, {s6r, s4, 0} → halt,
   {s6r, s4r, 0} → halt, {s6r, s5, 0} → s2r, {s6r, s5r, 0} → halt,
   {s6r, s6, 0} → s2r, {s1b, s1, 0} → s1r, {s1b, s3, 0} → s1r};
```

```
Length[GroupI]

106

GroupII =
  {{s1, s1, 0} → {s1, s6}, {s1, s3, 0} → {s1, s6}, {s1, s4, 0} → {s1, s6},
   {s2, s2, 0} → {s2, s4}, {s2, s5, 0} → {s2, s4}, {s2, s6, 0} → {s2, s4},
   {s3, s2, 0} → {s2, s4}, {s3, s5, 0} → {s2, s4}, {s3, s6, 0} → {s2, s4},
   {s4, s1, 0} → {s1, s6}, {s4, s3, 0} → {s1, s6}, {s4, s4, 0} → {s1, s6},
   {s5, s1, 0} → {s1, s6}, {s5, s3, 0} → {s1, s6}, {s5, s4, 0} → {s1, s6},
   {s6, s2, 0} → {s2, s4}, {s6, s5, 0} → {s2, s4}, {s6, s6, 0} → {s2, s4},
   {s1m, s3, 0} → {s6, s1}, {s1m, s1, 0} → {s6, s1},
   {s3, s2m, 0} → {s4, s2}, {s2, s2m, 0} → {s4, s2}};

Length[GroupII]

22
```

B. Symmetry Rules

```
rulesquad2 = {s1 → s1, s1m → s1, s1b → s1, s1r → s1, s2 → s2,
   s2m → s2, s2b → s2, s2r → s2, s3 → s4, s3m → s4, s3r → s4,
   s4 → s3, s4r → s3, s5 → s6, s5r → s6, s6 → s5, s6r → s5};
rulesquad3quad4 = {s1 → s1, s1m → s1, s1b → s1, s1r → s1, s2 → s2,
   s2m → s2, s2b → s2, s2r → s2, s3 → s6, s3m → s6, s3r → s6,
   s4 → s5, s4r → s5, s5 → s4, s5r → s4, s6 → s3, s6r → s3};
```

C. Pipe Routing Rule Set

This appendix lists the 432 rules: the formal simple relationships of form-function symbolically expressed according to local neighborhood conditions.

```
ruleSet = {{s1m} → s1m, {s2m} → s2m, {s5m} → s5m, {s2b} → s2b, {s0m} → s0m,
   {s1m, s2m, s0} → s3, {s3, s2m, s0} → s2, {s3, s2m, s0r} → s4,
   {s2, s2m, s0} → s2, {s2, s2m, s0r} → s4, {s4, s2m, s0} → s3,
   {s4, s2m, s0r} → s3, {s1m, s3, s0} → s1, {s1m, s3, s0r} → s6,
   {s1m, s4, s0} → s1, {s1m, s4, s0r} → s6, {s3, s2b, s0} → s4r,
   {s2, s2b, s0} → s4r, {s4, s2b, s0} → halt, {s1m, s1, s0} → s1,
   {s1m, s1, s0r} → s6, {s1m, s6, s0} → s3, {s1m, s6, s0r} → s3,
   {s5m, s1, s0} → s1n, {s5m, s1, s0r} → s6n, {s5m, s3, s0} → s1n,
   {s5m, s3, s0r} → s6n, {s5m, s6, s0} → s3n, {s5m, s6, s0r} → s3n,
   {s2, s5m, s0} → s2, {s2, s5m, s0r} → s4, {s3, s5m, s0} → s2,
   {s3, s5m, s0r} → s4, {s4, s5m, s0} → s3, {s4, s5m, s0r} → s3,
   {s1, s4r, s0} → s1r, {s2, s4r, s0} → s5r, {s3, s4r, s0} → s5r,
   {s4, s4r, s0} → s1r, {s5, s4r, s0} → s1r, {s6, s4r, s0} → halt,
   {s1, s1r, s0} → s1r, {s2, s1r, s0} → s5r, {s3, s1r, s0} → s5r,
   {s4, s1r, s0} → s1r, {s5, s1r, s0} → s1r, {s6, s1r, s0} → s5r,
   {s1, s5r, s0} → halt, {s2, s5r, s0} → s4r, {s3, s5r, s0} → s4r,
{s4, s5r, s0} → halt, {s5, s5r, s0} → halt, {s6, s5r, s0} → s4r,
{s1, s1, s0r} → s6, {s1, s1, s0} → s1, {s1, s2, s0} → s3,
{s1, s2, s0r} → s3, {s1, s3, s0r} → s6, {s1, s3, s0} → s1,
{s1, s4, s0r} → s6, {s1, s4, s0} → s1, {s1, s5, s0} → s3,
{s1, s5, s0r} → s3, {s1, s6, s0} → s3, {s1, s6, s0r} → s3,
{s2, s1, s0} → s5, {s2, s1, s0r} → s5, {s2, s2, s0r} → s4,
{s2, s2, s0} → s2, {s2, s3, s0} → s5, {s2, s3, s0r} → s5,
{s2, s4, s0} → s5, {s2, s4, s0r} → s5, {s2, s5, s0r} → s4,
{s2, s5, s0} → s2, {s2, s6, s0r} → s4, {s2, s6, s0} → s2,
{s4, s1, s0r} → s6, {s4, s1, s0} → s1, {s4, s2, s0} → s3,
{s4, s2, s0r} → s3, {s4, s3, s0r} → s6, {s4, s3, s0} → s1,
{s4, s4, s0r} → s6, {s4, s4, s0} → s1, {s4, s5, s0} → s3,
{s4, s5, s0r} → s3, {s4, s6, s0} → s3, {s4, s6, s0r} → s3,
{s6, s1, s0} → s5, {s6, s1, s0r} → s5, {s6, s2, s0r} → s4,
{s6, s2, s0} → s2, {s6, s3, s0} → s5, {s6, s3, s0r} → s5,
{s6, s4, s0} → halt, {s6, s4, s0r} → halt, {s6, s5, s0r} → s4,
{s6, s5, s0} → s2, {s6, s6, s0r} → s4, {s6, s6, s0} → s2,
{s5, s1, s0r} → s6, {s5, s1, s0} → s1, {s5, s2, s0} → s3,
{s5, s2, s0r} → s3, {s5, s3, s0r} → s6, {s5, s3, s0} → s1,
{s5, s4, s0r} → s6, {s5, s4, s0} → s1, {s5, s5, s0} → s3,
{s5, s5, s0r} → s3, {s5, s6, s0} → s3, {s5, s6, s0r} → s3,
{s3, s1, s0} → s5, {s3, s1, s0r} → s5, {s3, s2, s0r} → s4,
{s3, s2, s0} → s2, {s3, s3, s0} → s5, {s3, s3, s0r} → s5,
{s3, s4, s0} → s5, {s3, s4, s0r} → s5, {s3, s5, s0r} → s4,
{s3, s5, s0} → s2, {s3, s6, s0r} → s4, {s3, s6, s0} → s2,
{s1, s4p, s0} → s1r, {s2, s4p, s0} → s5r, {s3, s4p, s0} → s5r,
{s4, s4p, s0} → s1r, {s5, s4p, s0} → s1r, {s6, s4p, s0} → halt,
{s1, s1p, s0} → s1r, {s2, s1p, s0} → s5r, {s3, s1p, s0} → s5r,
{s4, s1p, s0} → s1r, {s5, s1p, s0} → s1r, {s6, s1p, s0} → halt,
{s1, s5p, s0} → halt, {s2, s5p, s0} → s4r, {s3, s5p, s0} → halt,
{s4, s5p, s0} → halt, {s5, s5p, s0} → halt, {s6, s5p, s0} → s4r,
{s1, s3p, s0} → s1r, {s2, s3p, s0} → s5r, {s3, s3p, s0} → s5r,
{s4, s3p, s0} → s1r, {s5, s3p, s0} → s1r, {s6, s3p, s0} → halt,
{s1, s2p, s0} → halt, {s2, s2p, s0} → s4r, {s3, s2p, s0} → halt,
{s4, s2p, s0} → halt, {s5, s2p, s0} → halt, {s6, s2p, s0} → s4r,
{s1, s6p, s0} → halt, {s2, s6p, s0} → s4r, {s3, s6p, s0} → halt,
{s4, s6p, s0} → halt, {s5, s6p, s0} → halt, {s6, s6p, s0} → s4r,
{s2, s1n, s0} → s5, {s2, s1n, s0r} → s5, {s3, s1n, s0} → s5,
{s3, s1n, s0r} → s5, {s4, s1n, s0} → s1, {s4, s1n, s0r} → s6,
{s2, s3n, s0} → s5, {s2, s3n, s0r} → s5, {s3, s3n, s0} → s5,
{s3, s3n, s0r} → s5, {s4, s3n, s0} → s1, {s4, s3n, s0r} → s6,
{s2, s6n, s0} → s2, {s2, s6n, s0r} → s4, {s3, s6n, s0} → s2,
{s3, s6n, s0r} → s4, {s4, s6n, s0} → s3, {s4, s6n, s0r} → s3,
{s1n, s1, s0} → s1p, {s1n, s1, s0r} → s6p, {s3n, s1, s0} → s5p,
{s3n, s1, s0r} → s5p, {s6n, s1, s0} → halt, {s6n, s1, s0r} → halt,
{s1n, s2, s0} → s3p, {s1n, s2, s0r} → s3p, {s3n, s2, s0} → s2p,
{s3n, s2, s0r} → s2p, {s6n, s2, s0} → s4p, {s6n, s2, s0r} → s4p,
{s1n, s3, s0} → s1p, {s1n, s3, s0r} → s6p, {s3n, s3, s0} → s5p,
{s3n, s3, s0r} → s5p, {s6n, s3, s0} → halt, {s6n, s3, s0r} → halt,
{s1n, s4, s0} → s1p, {s1n, s4, s0r} → s6p, {s3n, s4, s0} → s5p,
{s3n, s4, s0r} → s5p, {s6n, s4, s0} → halt, {s6n, s4, s0r} → halt,
{s1n, s5, s0} → s3p, {s1n, s5, s0r} → s3p, {s3n, s5, s0} → s2p,
{s3n, s5, s0r} → s2p, {s6n, s5, s0} → s4p, {s6n, s5, s0r} → s4p,
{s1n, s6, s0} → s3p, {s1n, s6, s0r} → s3p, {s3n, s6, s0} → s2p,
{s3n, s6, s0r} → s2p, {s6n, s6, s0} → s4p, {s6n, s6, s0r} → s4p,
{s1r, s1, s0} → s1, {s1r, s1, s0r} → s6, {s4r, s1, s0} → s1,
{s4r, s1, s0r} → s6, {s5r, s1, s0} → s1, {s5r, s1, s0r} → s6,
{s1r, s2, s0} → s3, {s1r, s2, s0r} → s3, {s4r, s2, s0} → s3,
{s4r, s2, s0r} → s3, {s5r, s2, s0} → s3, {s5r, s2, s0r} → s3,
{s1r, s3, s0} → s1, {s1r, s3, s0r} → s6, {s4r, s3, s0} → s1,
{s4r, s3, s0r} → s6, {s5r, s3, s0} → s1, {s5r, s3, s0r} → s6,
{s1r, s4, s0} → s1, {s1r, s4, s0r} → s6, {s4r, s4, s0} → s1,
{s4r, s4, s0r} → s6, {s5r, s4, s0} → s1, {s5r, s4, s0r} → s6,
{s1r, s5, s0} → s3, {s1r, s5, s0r} → s3, {s4r, s5, s0} → s3,
{s4r, s5, s0r} → s3, {s5r, s5, s0} → s3, {s5r, s5, s0r} → s3,
{s1r, s6, s0} → s3, {s1r, s6, s0r} → s3, {s4r, s6, s0} → s3,
{s4r, s6, s0r} → s3, {s5r, s6, s0} → s3, {s5r, s6, s0r} → s3,
{s1p, s1, s0} → s1, {s1p, s1, s0r} → s6, {s2p, s1, s0} → s5,
{s2p, s1, s0r} → s5, {s3p, s1, s0} → s5, {s3p, s1, s0r} → s5,
{s4p, s1, s0} → s1, {s4p, s1, s0r} → s6, {s5p, s1, s0} → s1,
{s5p, s1, s0r} → s6, {s6p, s1, s0} → s5, {s6p, s1, s0r} → s5,
{s1p, s2, s0} → s3, {s1p, s2, s0r} → s3, {s2p, s2, s0} → s2,
{s2p, s2, s0r} → s4, {s3p, s2, s0} → s2, {s3p, s2, s0r} → s4,
{s4p, s2, s0} → s3, {s4p, s2, s0r} → s3, {s5p, s2, s0} → s3,
{s5p, s2, s0r} → s3, {s6p, s2, s0} → s2, {s6p, s2, s0r} → s4,
{s1p, s3, s0} → s1, {s1p, s3, s0r} → s6, {s2p, s3, s0} → s5,
{s2p, s3, s0r} → s5, {s3p, s3, s0} → s5, {s3p, s3, s0r} → s5,
{s4p, s3, s0} → s1, {s4p, s3, s0r} → s6, {s5p, s3, s0} → s1,
{s5p, s3, s0r} → s6, {s6p, s3, s0} → s5, {s6p, s3, s0r} → s5,
{s1p, s4, s0} → s1, {s1p, s4, s0r} → s6, {s2p, s4, s0} → s5,
{s2p, s4, s0r} → s5, {s3p, s4, s0} → s5, {s3p, s4, s0r} → s5,
{s4p, s4, s0} → s1, {s4p, s4, s0r} → s6, {s5p, s4, s0} → s1,
{s5p, s4, s0r} → s6, {s6p, s4, s0} → s5, {s6p, s4, s0r} → s5,
{s1p, s5, s0} → s3, {s1p, s5, s0r} → s3, {s2p, s5, s0} → s2,
{s2p, s5, s0r} → s4, {s3p, s5, s0} → s2, {s3p, s5, s0r} → s4,
{s4p, s5, s0} → s3, {s4p, s5, s0r} → s3, {s5p, s5, s0} → s3,
{s5p, s5, s0r} → s3, {s6p, s5, s0} → s2, {s6p, s5, s0r} → s4,
{s1p, s6, s0} → s3, {s1p, s6, s0r} → s3, {s2p, s6, s0} → s2,
{s2p, s6, s0r} → s4, {s3p, s6, s0} → s2, {s3p, s6, s0r} → s4,
{s4p, s6, s0} → s3, {s4p, s6, s0r} → s3, {s5p, s6, s0} → s3,
{s5p, s6, s0r} → s3, {s6p, s6, s0} → s2, {s6p, s6, s0r} → s4,
{s1} → s1, {s2} → s2, {s3} → s3, {s4} → s4, {s5} → s5, {s6} → s6,
{s1r} → s1r, {s2r} → s2r, {s3r} → s3r, {s4r} → s4r, {s5r} → s5r,
{s6r} → s6r, {s1p} → s1p, {s2p} → s2p, {s3p} → s3p, {s4p} → s4p,
{s5p} → s5p, {s6p} → s6p, {s1n} → s1n, {s2n} → s2n, {s3n} → s3n,
{s4n} → s4n, {s5n} → s5n, {s6n} → s6n, {s0, s0, s0} → s0,
{s0, s0r, s0} → s0, {s0r, s0, s0} → s0, {s0, s0, s0r} → s0r,
{s0, s0r, s0r} → s0r, {s0r, s0r, s0} → s0, {s0r, s0, s0r} → s0r,
{s0r, s0r, s0r} → s0r, {s0, s2m, s0} → s0, {s0, s2m, s0r} → s0r,
{s0r, s2m, s0} → s0, {s0r, s2m, s0r} → s0r, {s0, s2b, s0} → s0,
{s0r, s2b, s0} → s0, {s1m, s0, s0} → s0, {s1m, s0, s0r} → s0r,
{s1m, s0r, s0} → s0, {s1m, s0r, s0r} → s0r, {s5m, s0, s0} → s0,
{s5m, s0, s0r} → s0r, {s5m, s0r, s0} → s0, {s5m, s0r, s0r} → s0r,
{s0, s5m, s0} → s0, {s0, s5m, s0r} → s0r, {s0r, s5m, s0} → s0,
{s0r, s5m, s0r} → s0r, {s0, s1n, s0} → s0, {s0, s1n, s0r} → s0r,
{s0r, s1n, s0} → s0, {s0r, s1n, s0r} → s0r, {s0, s3n, s0} → s0,
{s0, s3n, s0r} → s0r, {s0r, s3n, s0} → s0, {s0r, s3n, s0r} → s0r,
{s1r, s0, s0} → s0, {s1r, s0, s0r} → s0r, {s1r, s0r, s0} → s0,
{s1r, s0r, s0r} → s0r, {s4r, s0, s0r} → s0r, {s4r, s0r, s0} → s0,
{s4r, s0r, s0r} → s0r, {s5r, s0, s0} → s0, {s5r, s0, s0r} → s0r,
{s5r, s0r, s0} → s0, {s5r, s0r, s0r} → s0r, {s0, s1, s0} → s0,
{s0, s2, s0} → s0, {s0, s3, s0} → s0, {s0, s4, s0} → s0,
{s0, s5, s0} → s0, {s0, s6, s0} → s0, {s0, s1, s0r} → s0r,
{s0, s2, s0r} → s0r, {s0, s3, s0r} → s0r, {s0, s4, s0r} → s0r,
{s0, s5, s0r} → s0r, {s0, s6, s0r} → s0r, {s0r, s1, s0} → s0,
{s0r, s2, s0} → s0, {s0r, s3, s0} → s0, {s0r, s4, s0} → s0,
{s0r, s5, s0} → s0, {s0r, s6, s0} → s0, {s0r, s1, s0r} → s0r,
{s0r, s2, s0r} → s0r, {s0r, s3, s0r} → s0r, {s0r, s4, s0r} → s0r,
{s0r, s5, s0r} → s0r, {s0r, s6, s0r} → s0r, {s0, s1p, s0} → s0,
{s0r, s1p, s0} → s0, {s0, s2p, s0} → s0, {s0r, s2p, s0} → s0,
{s0, s3p, s0} → s0, {s0r, s3p, s0} → s0, {s0, s4p, s0} → s0,
{s0r, s4p, s0} → s0, {s0, s5p, s0} → s0, {s0r, s5p, s0} → s0,
{s0, s6p, s0} → s0, {s0r, s6p, s0} → s0, {s0, s1r, s0} → s0,
{s0r, s1r, s0} → s0, {s0, s4r, s0} → s0, {s0r, s4r, s0} → s0};
```

References

[1] Xenophon, *Xenophon: Anabasis Books I-VII* (trans. C. L. Brownson), Cambridge, MA: Harvard University Press, 1980.

[2] O. Jones, *The Grammar of Ornament*, London: Studio Editions, 1986.

[3] W. H. Goodyear, *The Grammar of the Lotus*, New York: Sampson Low and Co., 1891.

[4] C. J. Herringham, "The Snake Pattern in Ireland, the Mediterranean and China," *The Burlington Magazine for Connoisseurs*, **13**(63), 1908 pp. 132–137.

[5] C. J. Herringham, "Notes on Oriental Carpet Patterns-VI. Meander and Key Patterns," *The Burlington Magazine for Connoisseurs*, **15**(74), 1909 pp. 98–104.

[6] D. S. Dye, *A Grammar of Chinese Lattice*, Vol. 2, Cambridge, MA: Harvard University Press, 1937.

[7] T. W. Knight, "Transformations of the Meander Motif on Greek Geometric Pottery," *Design Computing*, **1**, 1986, pp. 29–67.

[8] T. W. Knight, *Transformations in Design: A Formal Approach to Stylistic Change and Innovation in the Visual Arts*, Cambridge: Cambridge University Press, 1994.

[9] T. H. Speller, Jr., "An Algorithmic Approach to System Architecting Using Shape Grammar-Cellular Automata," Ph.D. thesis, Engineering Systems Division, School of Engineering, Massachusetts Institute of Technology, Cambridge, MA, 2008.

[10] G. Stiny and J. Gips, "Shape Grammars and the Generative Specification of Painting and Sculpture," in *Information Processing (IFIP71)*, Ljubljana, Yugoslavia (C. V. Freiman, ed.), Amsterdam: North Holland, 1972 pp. 1460–1465.

[11] N. Chomsky, *Syntactic Structures*, The Hague: Mouton & Co., 1957.

[12] W. J. Mitchell, *The Logic of Architecture: Design, Computation, and Cognition*, Cambridge, MA: MIT Press, 1990.

[13] G. Stiny, "Weights," *Environment and Planning B: Planning and Design*, **19**(4), 1992 pp. 413–430. doi:10.1068/b190413.

[14] G. Stiny, *Shape: Talking about Seeing and Doing*, Cambridge, MA: MIT Press, 2006.

[15] M. Agarwal, J. Cagan, and G. Stiny, "A Micro Language: Generating MEMS Resonators by Using a Coupled Form-Function Shape Grammar," *Environment and Planning B: Planning and Design*, **27**(4), 1999 pp. 615–626. doi:10.1068/b2619.

[16] M. Agarwal and J. Cagan, "A Blend of Different Tastes: The Language of Coffeemakers," *Environment and Planning B: Planning and Design*, **25**(2), 1998 pp. 205–226. doi:10.1068/b250205.

[17] K. Shea and J. Cagan, "The Design of Novel Roof Trusses with Shape Annealing: Assessing the Ability of a Computational Method in Aiding Structural Designers with Varying Design Intent," *Design Studies*, **20**(1), 1999 pp. 3–23. doi:10.1016/S0142-694X(98)00019-2.

[18] Y. Wang, "3D Architecture Form Synthesizer," M.S. thesis, Department of Architecture, Massachusetts Institute of Technology, Cambridge, MA, 1999. http://dspace.mit.edu/handle/1721.1/51561.

[19] M. Tapia, "From Shape to Style. Shape Grammars: Issues in Representation and Computation," Ph.D. thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1996.

[20] M. Tapia, "A Visual Implementation of a Shape Grammar System," *Environment and Planning B: Planning and Design*, **26**(1), 1999 pp. 59–73. doi:10.1068/b260059.

[21] P. Testa, U.-M. O'Reilly, M. Kangas, and A. Kilian, "MoSS: Morphogenetic Surface Structure: A Software Tool for Design Exploration," in *Proceedings of Greenwich 2000 Digital Creativity Symposium,* Greenwich, England: University of Greenwich, 2000. http://designexplorer.net/newscreens/moss/moss.pdf.

[22] M. McGill, "A Visual Approach for Exploring Computational Design," S.M. thesis, Department of Architecture, Massachusetts Institute of Technology, Cambridge, MA, 2001. http://dspace.mit.edu/handle/1721.1/68806.

[23] J. von Neumann, *Theory of Self-Reproducing Automata* (A. W. Burks, ed.), Urbana, IL: University of Illinois Press, 1966.

[24] J. von Neumann, *The Computer and the Brain*, New Haven, CT: Yale University Press, 1958.

[25] A. Ilachinski, *Cellular Automata: A Discrete Universe*, River Edge, NJ: World Scientific, 2001.

[26] N. Wiener and A. Rosenblueth, "The Mathematical Formulation of the Problem of Conduction of Impulses in a Network of Connected Excitable Elements, Specifically in Cardiac Muscle," *Archivos del Instituto Cardioligia de Mexico*, **16**(3), 1946 pp. 205–265.

[27] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., 2002.

[28] S. Wolfram, "Statistical Mechanics of Cellular Automata," *Reviews of Modern Physics*, **55**(3), 1983 pp. 601–644. http://www.stephenwolfram.com/publications/articles/ca/83-statistical.

[29] H. Abelson, G. J. Sussman, and J. Sussman, *Structure and Interpretation of Computer Programs*, 2nd ed., Cambridge, MA: MIT Press, 1996.

[30] *Mathematica*, Release Version 8.0, Champaign, IL: Wolfram Research, Inc., 2011.

[31] N. Ganguly et al., *A Survey on Cellular Automata*, technical report GSD+03, Centre for High Performance Computing, Dresden University of Technology, Dresden, Germany, 2003. http://www.cs.unibo.it/bison/publications/CAsurvey.pdf.

[32] P. Hajela and B. Kim, "GA Based Learning in Cellular Automata Models for Structural Analysis," in *Proceedings of the 3rd World Congress on Structural and Multidisciplinary Optimization (WCSMO99)*, Buffalo, NY, 1999.

[33] P. Hajela and B. Kim, "On the Use of Energy Minimization for CA Based Analysis in Elasticity," *Structural and Multidisciplinary Optimization*, **23**(1), 2000 pp. 24–33. doi:10.1007/s00158-001-0162-2.

[34] J. R. Koza et al., *Genetic Programming III: Darwinian Invention and Problem Solving*, San Francisco, CA: Morgan Kaufmann Publishers, 1999.

[35] E. F. Moore, "Machine Models of Self-Reproduction," *Proceedings of the 14th Symposium in Applied Mathematics* (R. E. Bellman, ed.), **14**, New York: American Mathematical Society, 1962 pp. 17–33.

[36] E. F. Moore, "Machine Models of Self-Reproduction," *Proceedings of the Symposium on Mathematical Problems in the Biological Sciences*, New York: American Mathematical Society, 1961.

[37] J. Holland, "Genetic Algorithms and the Optimal Allocations of Trial," *SIAM Journal of Computing*, **2**(2), 1973 pp. 88–105.

[38] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addison-Wesley, 1989.

[39] Underfloor Heating Systems. "Underfloor Heating Pipe Layout." (Apr 4, 2012) http://www.underfloorheatingsystems.co.uk/underfloor-heating-design/pipe-layout.

[40] G. Schlosser and G. P. Wagner, eds., *Modularity in Development and Evolution*, Chicago: University of Chicago Press, 2004.

[41] H. A. Simon, *The Sciences of the Artificial*, Cambridge, MA: MIT Press, 1996.

[42] T. H. Speller, Jr., "The Use of Shape Grammar-Cellular Automata for Modeling Molecular Dynamics," *Journal of Computational and Theoretical Nanoscience*, **6**(10), 2009 pp. 2179–2193(15). doi:10.1166/jctn.2009.1271.