LEARNING GENETIC REPRESENTATIONS AS ALTERNATIVE TO HAND-CODED SHAPE GRAMMARS

THORSTEN SCHNIER AND JOHN S. GERO
Key Centre of Design Computing
Department of Architectural and Design Science
University of Sydney NSW 2006 Australia
Email: {thorsten,john}@arch.su.edu.au

[1]

Abstract. Shape grammars have been used to analyze and describe designs, and to create new designs that are similar in style to the designs the grammar is based on. The grammars are created by hand, involving a large amount of research about the designs and the design process. This paper proposes a different approach, where a system is given design examples, and in a bottom-up process learns stylistic features of the examples. This is achieved by using an evolutionary system that is able to change the representation it is using. With the creation of a more and more complex evolved representation, the search space of the evolutionary process is transformed so that the search for new designs is biased towards designs similar to the design examples.

1. Introduction

Shape grammars (Stiny, 1980) have been introduced as a method for formal descriptions of designs. Shape grammars consist of an alphabet of shapes, a starting shape, and shape rules that define spatial relations between shapes.

The power of shape grammars to analyze and describe designs has been shown in a variety of design areas, from architectural design (examples include Palladian Villas, Frank Lloyd Wright Houses, Wren's City Church designs and Japanese tearooms (Stiny and Mitchell, 1978; Knight, 1981; Koning and Eizenberg, 1981; Buelinckx, 1993)), over garden landscaping (Stiny and Mitchell, 1980; Knight, 1990) to de Stijl style paintings (Knight, 1989).

In all these examples, however, the translation from a set of designs into a shape grammar (and the reverse) has been done by hand. Very few attempts have

[1] To appear in *Artificial Intelligence in Design '96* edited by John S. Gero and Fay Sudweeks, Kluwer, Dordrecht, 1996

been made to automatize the process. Chase (1989) showed how the automatic generation of shapes from (given) shape grammars can be realized. Mackenzie (1989) described a system that is able to produce grammars from example designs, if the designs are described in terms of their basic components and their topology. The transformations used in that paper are not unique, many different grammars are possible for any given set of designs. The system uses a 'utility' function to distinguish between good and bad representations. This demonstrates a general problem: to create a 'sensible' shape grammar, a large amount of high level knowledge is required. Intentions of the designer, logical units in the design, logical stages in the design process are all represented in the grammars.

The purpose of this paper is to describe an alternative, more computationally oriented view. In the spirit of artificial life research, it uses a bottom-up approach, where complex shapes are created by assembling smaller sub-parts.

2. Evolving coding and Frank Lloyd Wright houses

In Gero and Schnier (1995), we described an evolutionary system which produces problem solutions that are based on example designs. In evolutionary systems, the results of a search process are very much influenced by the representation of the problem space in the coding. In usual implementations, this can pose a serious problem, because the representation might bias the results too much into certain directions. The system described in Gero and Schnier (1995), on the other hand, makes use of this by intentionally biasing the solution space towards a set of potentially interesting solutions. It does this in a two stage process. In the first step, the system is given a set of example designs. The goal of the evolutionary process in this step is to create individuals that resemble the example designs as closely as possible. To do this, the fitness function measures what and how much of the example designs are described by the individuals. The coding of the individuals is chosen to be very low-level, using very simple 'basic' genes.

While the individuals are evolved, they are at the same time analyzed, and successful combinations of low level genes are identified. For every such gene combination, a new gene is created (an 'evolved' gene) and introduced into the population. In the course of the evolution, the evolved genes which are produced aggregate more and more basic and lower-level evolved genes, encoding more and more complex aspects of the example designs. The coding itself, therefore, contains information about the example applications. Any evolutionary system using this coding is biased towards solutions similar to the design examples. This is used in the second step, where a conventional evolutionary system creates solutions to a design problem, using both original basic genes and the evolved genes. The system therefore produces solutions that incorporate aspects of the example designs, but are adapted to the new design requirements. Figure 1 illustrates the idea: beginning with a basic representation, the system creates an evolved cod-

ing based on a set of design examples. This evolved representation is then used to create new designs that show design features from the examples.

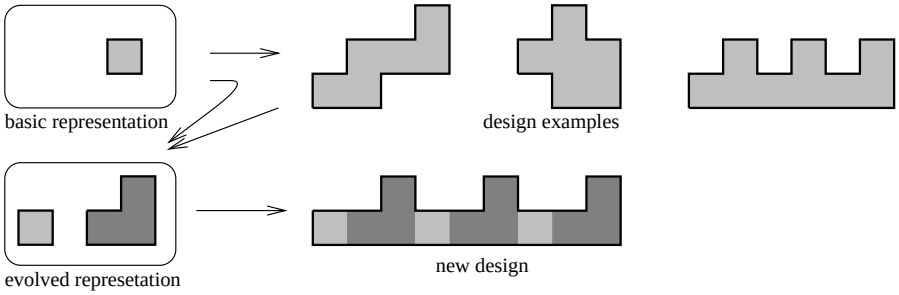


Figure 1. Use of evolved representation to capture and use typical features of a set of example designs.

2.1. INDIVIDUAL STYLE AND THE STYLE OF FRANK LLOYD WRIGHT' S PRAIRIE HOUSES

In Chan (1995), the author explained that the distinct features of a design are produced by both common features and common procedures used by the architect. Describing a style as a function of how it is generated therefore requires a deep understanding of the design process, usually supported by comments from the designer. Shape grammars usually take this approach, they represent both the common procedures (in the rules and the sequences of rules that are possible), and the common features (in the shapes manipulated by the rules). However, Chan also noted that "common features present in an architects work are indeed used by viewers to categorize the architect's style. . . . a style is said to be the function of common features". This means that, even without knowledge about the design process, it is possible to infer important aspects of a style common to a set of designs.

Chan (1992) has analyzed the style of Frank Lloyd Wright's prairie houses more closely. Some of the aspects that are of interest for the work presented here are:

1. Floorplans are always based on a grid, the grid size depends on the project.
2. The fireplace is at the center of the composition, all spaces extend from there.
3. One major shape in the floorplan is long and narrow, much of the house is only one room in depth.
4. The prairie houses have similar topological arrangements.
5. The first design step after developing an abstract of the space, is to create a geometric pattern (based on the grid).
6. The next design step integrates the functional requirements.

7. The elevation follows directly from application of an 'elevation grammar' to the plan.

Item 1 allows us to use a basic coding that is based on unit length horizontal and vertical lines. The next two items are aspects that we wish our system to pick up from the example designs, together with some of the topological arrangements from item 4. Items 5 and 6 describe the first steps in the design process, the plans used here represent in their level of detail approximately the ones produced after these steps.

The elevations are not considered in this work. As item 7 states, they are a result of the development of a floorplan. Learning features in elevations without having a given floorplan does therefore not seem sensible.

2.1.1. *Shape rules for Frank Lloyd Wright houses*

The work in this paper is based on the analysis of Frank Lloyd Wright's Prairie Houses by Koning and Eizenberg (1981). Based on the layouts of 11 prairie houses, Koning and Eizenberg develop a shape grammar that can be used to construct 10 of these houses, as well as many others that show a similar style. Their work is a typical example of the top-down analysis described above. The design using the rules is separated into 24 different steps. Roughly, the following phases can be distinguished: starting with the fireplace, a basic composition is created (18 rules). This composition is further elaborated by adding corners and porches, and detailing the interior layout (16 rules). More exterior details are added (22 rules), and the design is extended into the third dimension (12 rules). The roof is established (19 rules), together with some more details (4 rules). With the 8 rules to manipulate labels, 99 rules are necessary to create the ten different layouts.

The focus of this paper is the designs that are created by the first 34 rules: 2-dimensional layouts, with a developed basic layout, organized into function zones, and some detailing.

2.2. SEMANTICS

An important aspect of the designs we are looking at is the distinction between different function zones. The layouts have zones representing living space, service space and porches. Of central importance is the location of the fireplace. In the shape grammar used in Koning and Eizenberg (1981), the zones for service and living space are established around the fireplace with the first rules, and detailed at the end of the first 38 rules. At the same time, porches are added.

2.2.1. *Semantics in basic coding*

As described, both shapes and functional organization can be important aspects of a style of a set of designs. A system like the one described in Gero and Schnier (1995) that uses only four primitives to describe outlines (line, step, right turn, left

turn) would therefore not be sufficient. To capture information about the functional organization, the evolving coding has to be able to integrate information about the semantics of the shapes. To do this, the basic coding has to be changed, so that semantic information can be attached to the outlines.

This is done by changing the set of primitives coded by the basic genes. The elements to change direction (left turn, right turn) remain unchanged. But instead of having only either a drawn line or a step ahead, the changed basic coding now includes a set of lines of different types, with the step ahead represented as a line of type 'invisible'. The number of types is not restricted, the number of types used depends on the application.

The line types in the the basic coding are interpreted to represent the different semantics or functions of the rooms. The fitness functions used in both steps of the evolution reflect this. In the first step, to score for a certain part of a design, any individual produced has to fit the design not only in line types, but also in shape. This also means that the number of line types in the basic coding can be higher than the number used in any specific example. Individuals exhibiting unused line types don't score any fitness, the evolving genes therefore do not incorporate any combinations of basic genes that produce these line types. In the second phase, the way the results are interpreted depends on the way the line types are used. If the line types specify the function of an enclosed room, the function of the room is defined by the line type that has the majority. If the line type encodes a detail in the outline, e.g. a certain wall type or a window, then the result can be used directly without further interpretation. Both functions can be mixed, as seen in the example used here.

The line types can be represented by different colours, the coding then has some similarities to colour grammars (see e.g. Knight, 1994). In colour grammars, however, the colours don't have any semantics attached, and colours can be mixed.

3. Learning evolved genes

3.1. GENETIC ENGINEERING

One foundation for evolving representations is genetic engineering. It is derived from genetic engineering notions related to human intervention in the genetics of natural organisms. If a group of similar organisms can be seperated into two sets distinguished by a difference in one particular attribute, then a comparison of the genetic codes of the organisms in the two groups can reveal what genes or gene groups are responsible for the difference. This knowledege can be used to modify that attribute, and introduce it into or eliminate it from organisms by manipulating its genetic material (see Gero and Schnier (1995) for a more detailled discussion).

A useful notion related to genetic engineering is the definition of 'genotype' and 'phenotype'. The genotype is the set of genetic instructions that make up the

genetic code, while the phenotype is the structure that is produced as the result of the interpretation of the genotype (Langton, 1988).

3.2. EVOLVING REPRESENTATION

The starting point in the development of a system that creates and makes use of an evolving coding is a standard evolutionary system. The first step is to create a population of randomly created individuals. The coding of these individuals, the 'basic' genes, is chosen to be very low-level, putting as little domain knowledge into the coding as possible, and making sure not to exclude any interesting part of the search space. The individuals are then subjected to the standard evolutionary cycle of replication with errors and survival of the fittest. But at the same time, an additional operation screens the population, identifying particularly successful combinations of genes. For every such gene combination, a new, 'evolved' gene is created that represents this combination, and is introduced into the population. Figure 2 shows pseudo-code describing the algorithm. More detailed explanations to some of the steps can be found in the following sections.

```
begin
   create-population
   while not end-criterion
         select 2 parents
         create offspring
         if offspring not already in population
           then add offspring to population
                register offspring for shared fitness fi
         if n new individuals produced
           then select best gene combination
                create evolved gene
                add evolved gene
                replace all occurences of gene combination with evolved gene fi
         if m new individuals produced
           then recalculate weights
                recalculate all fitnesses fi
   end
end
```

Figure 2. Pseudocode for an evolutionary system used to produce an evolved representation.

Since the goal during the development of a representation is variety and not optimization, all offspring created in the variation function are kept if they:

- are not empty, i.e. they draw at least one segment.
- match (as described below) the design case at least at one position.

– no other individual already in the population has a genotype that codes for the same phenotype as the new individual. If such an individual exists, then the individual with the shorter genotype is kept. The use of evolved genes is hereby encouraged, since evolved genes usually lead to shorter genotypes. This is the only instance where an individual in the population can be replaced.

As a result, there is no step in the pseudo-code where individuals are deleted.

In the first few cycles, the evolved genes will be composed from basic genes, but in later cycles most evolved genes will represent combinations of other, lower-level evolved genes, or combinations of those with basic genes. This growing hierarchy of representations gives rise to a more and more complex and abstract coding, which is increasingly adapted to the application. In other words, the process gradually collects application-specific knowledge and codes it into the representation, rather than being coded into it by the user in the first place.

What does this mean in terms of search space? The length of the genotype is only restricted by the size of the computer memory. The search space is extremely large with respect to the number of states that can be evaluated in a limited computation time, and can therefore be seen as having infinite size. However, since the alphabet used for genotypes is finite at any state during the evolution, the set of possible designs that can be defined by a genotype of a certain length is limited. The search space can therefore be illustrated by a number of concentric circles, each defining the space of designs that can be defined by a genotype of a certain length. The inner circle contains the genotypes of length one, i.e. the basic building blocks. The further away a design (or part of design) is from the centre, the more difficult it is to find by means of generate and search. Every time an evolved gene is created, the structure of the search space is changed. The state of the new gene in the search space is moved into the centre, all design states in the next circle that can be derived from that state are moved into the second circle, and so on. Figure 3 illustrates this: the original search space is illustrated in Figure 3(a), with the four basic building blocks in the centre. The building blocks code for vectors of one unit length with the directions up, down, left and right. The arrow points to the starting point of the next element drawn (if any). The second circle shows all designs that can be derived from genes of length two (i.e., using two building blocks). The other circles give some examples of designs using genotypes of length three, four and five. If now the two closed shapes in the fourth circle are identified as particularly successful and an evolved gene is introduced for them, then the search space changes as shown in Figure 3(b). The squares now become basic building blocks, and the shapes on the fifth circle that are derived from the squares, can now be found in the second circle. The more evolved genes a design state involves, the more it is moved towards the centre. For example, the shape with the four squares that is now on the fifth circle (i.e. can be constructed

from genotypes of length five) would have been on the fourteenth circle before[2]. Since the introduction of a new gene increases the size of the alphabet of the coding, the size of the circles also grows.

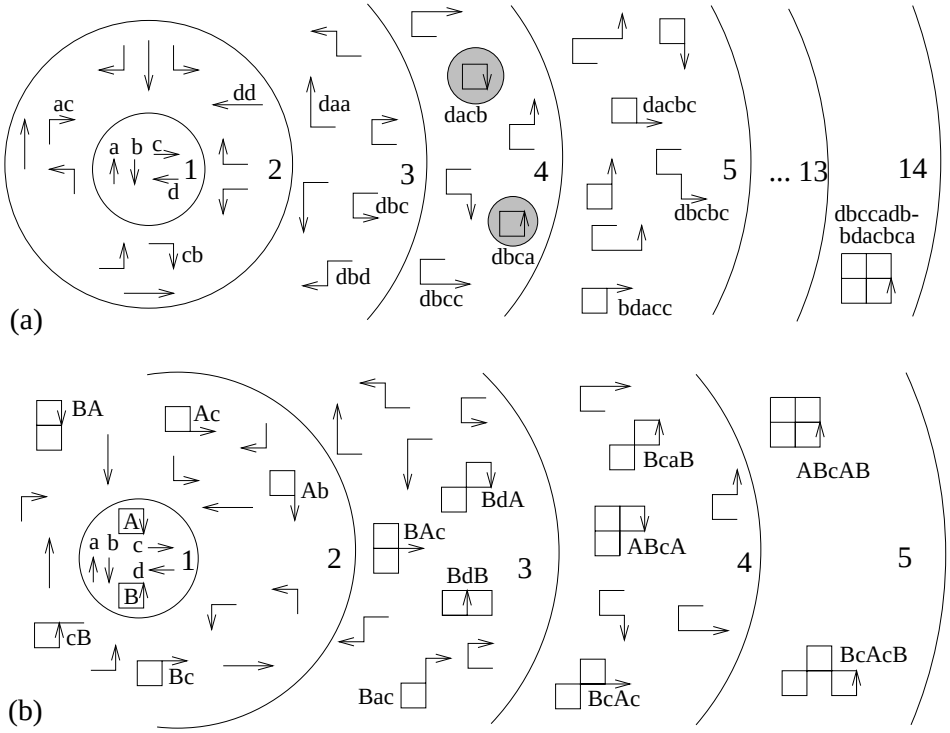


Figure 3. Example of an evolving representation: (a) original representation and (b) representation with evolved genes. Some of the corresponding genotypes are give, capital letters denote evolved genes. The transformation from phenotype to genotype is not always unique, e.g. the genotypes 'ABc' and 'BAc' produce the same phenotype. Arc segments indicate that only part of the space is shown.

The introduction of evolved genes obviously changes the probability that a gene sequence maps onto a useful feature. The number of different genes that can be used in a genotype expands, but at the same time the length of the genotype that is necessary to describe a feature shrinks, effectively reducing the size of the search space. For example, the floorplan shown in Figure 10 (a) was produced by a genotype of length eight, using two basic and 326 evolved genes. Expressed in basic genes only, the genotype has a length of 445. The space of designs that can be coded by genes of length 445 using two basic genes is about 110 orders of magnitude larger than the space of genes of length eight using 328 genes.

[2] fourteen lines, the shape cannot be drawn without drawing two lines twice

4. Learning genetic representations of floorplans

4.1. EXAMPLE FLOORPLANS

In the examples of the Frank Lloyd Wright prairie houses, lines that enclose living spaces use a different line type from lines that enclose service spaces or porches, and the fire place has a type of its own. Since only the main floor is considered here, no bedroom zones occur in the designs.

From the eleven Prairie Houses analyzed by Koning and Eizenberg (1981), four have been selected as examples for the evolving coding: the Henderson house, the Thomas house, the Martin house and the Baker house (Koning and Eizenberg, 1981). Since the basic coding only allows horizontal or vertical lines, the diagonal lines at the wings of the Henderson house have been changed into a stepped shape. Figure 4 shows the floorplans used. As a comparison, Figure 5 shows the plans of the Thomas house as given in Koning and Eizenberg (1981).

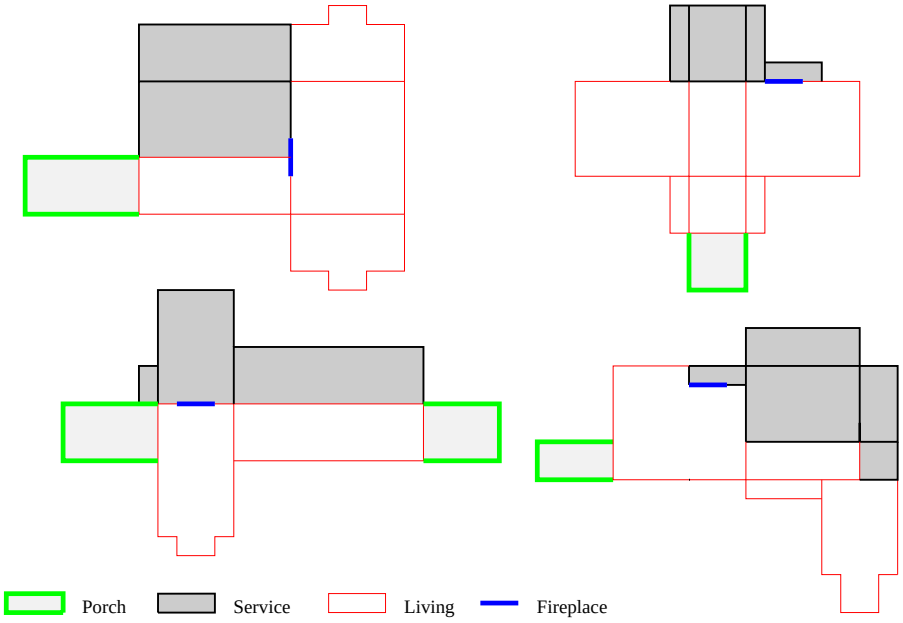


Figure 4. Frank Lloyd Wright Houses used to create the evolved coding: Henderson house (top left), Martin house (top right), Baker house (bottom left), Thomas house (bottom right).

4.2. BASIC CODING

The basic coding has to allow for lines with a variable number of line types. The coding presented in Gero (1994) used four basic genes, each coding for a different basic element: a left turn, a right turn, a line ahead, and a step ahead. One possibility for including line types is to increase the number of different basic genes,

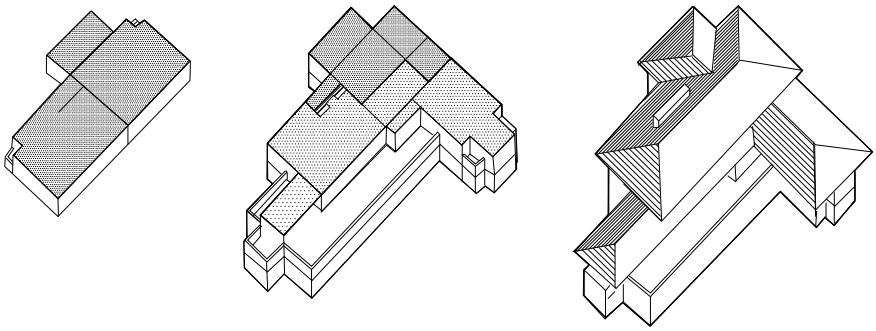


Figure 5. Thomas house defocused and reduced to four function zones, bedroom level, main floor level, and external form (Koning & Eizenberg 1981).

so that one basic gene is used for every additional type. If the number of types increases, or if additional basic elements like diagonal lines are introduced, the number of different basic genes used grows.

Another possibility is to keep the number of different basic genes constant, and use two or more successive basic genes to code for the elements. The first basic gene on a genotype would select the type of primitve used, in this case either turn or line. The following one or more basic genes then code for an attribute value. If the first gene coded for a turn, the attribute represents either of the two values 'left' or 'right'. For lines, the attribute defines the type of the line, steps ahead are treated as a line of type 'invisible'. The number of successive basic genes needed to code for the line types varies depending on the number of line types and on the number of basic genes used.

The second coding has the advantage that it is easily extensible, for example to introduce curved lines, only a new type of primitives would be added. This type could have one or more additional attributes. Similarly, for diagonal lines, one could change the number of values the turn can represent to eight. At the same time the coding remains very simple, this is one of the goals in the design of the basic coding.

A major difference from the first coding is that the meaning of basic genes is not totally position independent anymore.

In the example presented here, the second approach was chosen. Two basic genes (values 0 and 1) are used, the basic coding (without evolved genes) is therefore a binary coding. The first basic gene selects the type of the primitve:

0 A line. The attribute can take five value (five line types, including the 'step ahead', requiring three basic genes. The eight possible values are taken modulo five, three line types are therefore produced by two different values.

1 A turn. The following basic genes distinguishes between left and rigth turn.

The basic coding is shown in Figure 6.

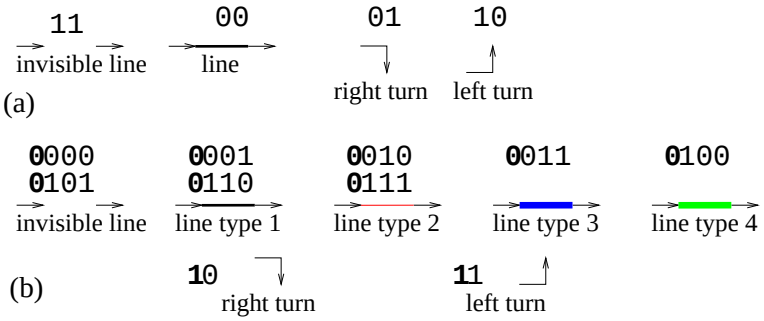


Figure 6. Basic coding, (a): original coding, (b) coding expanded to allow for different line types

4.3. CREATING EVOLVED GENES

To create evolved genes, the algorithm shown in Figure 2 is used. A special fitness function is used that measures what percentage of the examples is represented by an individual, while at the same time preventing convergence.

4.3.1. *Fitness Calculation*

The fitness calculation for a new individual during the evolution of the representation stage involves the following steps.

1. Transform the genotype of the individual into a phenotype, i.e. line-drawing.
2. Find all positions where the phenotype 'matches' the design case. A match is declared if and only if for all line segments in the phenotype there is a corresponding line segment in the design case (but not necessarily the other way round).
3. At all matching positions, draw the phenotype as a partial drawing.
4. Create the sum of the weights associated with all line segments in the design case that are represented in the partial drawing (see below for a description how the weights are calculated).
5. This sum is the current fitness value for the individual. Whenever the weights for the segments in the design case are recalculated, the fitness values of all individuals in the population have to be updated.

As an example, Figure 7(a) shows a design case with associated weights, and the shapes produced by two different individuals. Both individuals can be applied at two different positions, resulting in fitnesses of 24 and 18.

This fitness alone would lead to a convergence of individuals that describe only some aspects of the design case. To prevent this, another analogy from evolution in nature is used. The different aspects of the design case are seen as a resource (for example food) that has to be shared between all individuals using it.

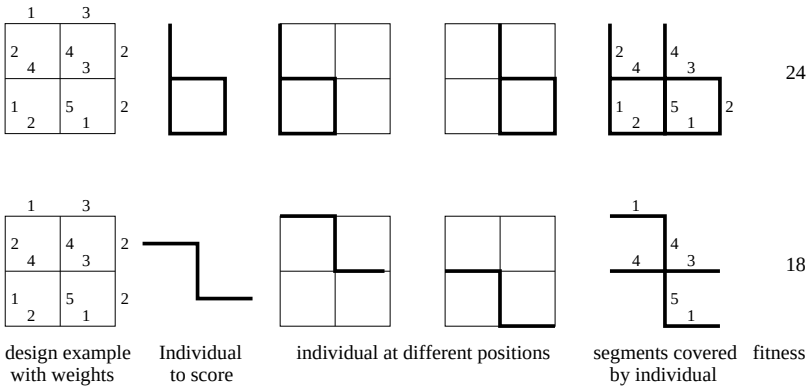


Figure 7. Example of fitness calculation for two different individuals.

Individuals therefore get high rewards if they describe aspects of the case that are covered only by few other individuals (evolutionary niches), and only little additional fitness for aspects that are described by many individuals.

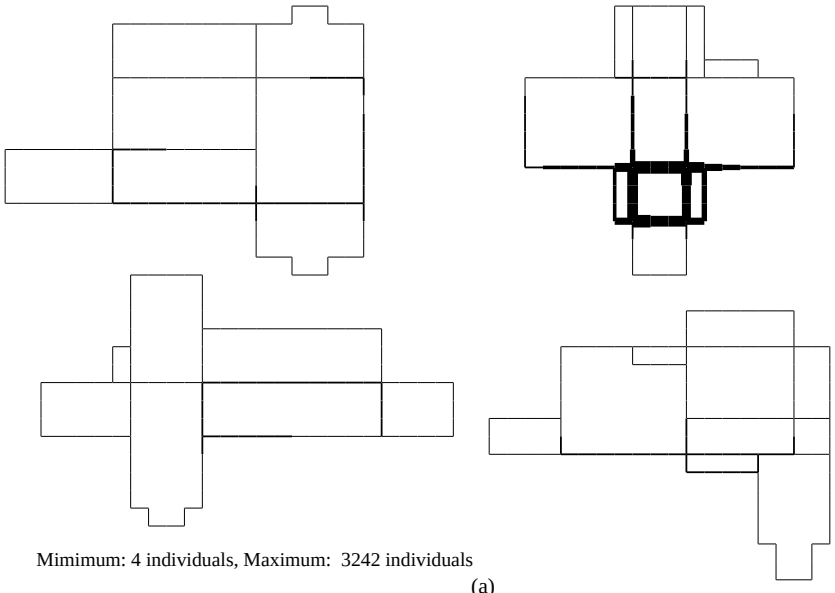
To create the 'niching' effect, every line segment in the design case has a weight associated with it (the values in Figure 7). This weight is calculated regularly as a fixed value divided by the number of individuals in the population that can be used to 'stamp' that segment. If for example only two individuals code for a 'stamp' that can be used for a certain part in the design case, both get 50% of the constant value as fitness for that part. If 20 individuals do so, each one gets only a fitness of 5%. This effectively prevents convergence towards only some features in the design case (e.g. only horizontal lines). The effect can be seen in Figure 8: without sharing, the evolving genes develop mainly in a very small region, fitness sharing leads to a much better distribution of evolved individuals.

4.3.2. *Results*

Figure 9 shows examples of evolved genes created from the four example designs. Shown are some of the last evolved genes created from the examples. Clearly visible are the shapes of rooms, and the different line types, associated with the different functions. Two of the evolved genes shown (310 and 318) have the fireplace as part of the line-drawing they code for (in this case from the Henderson house).

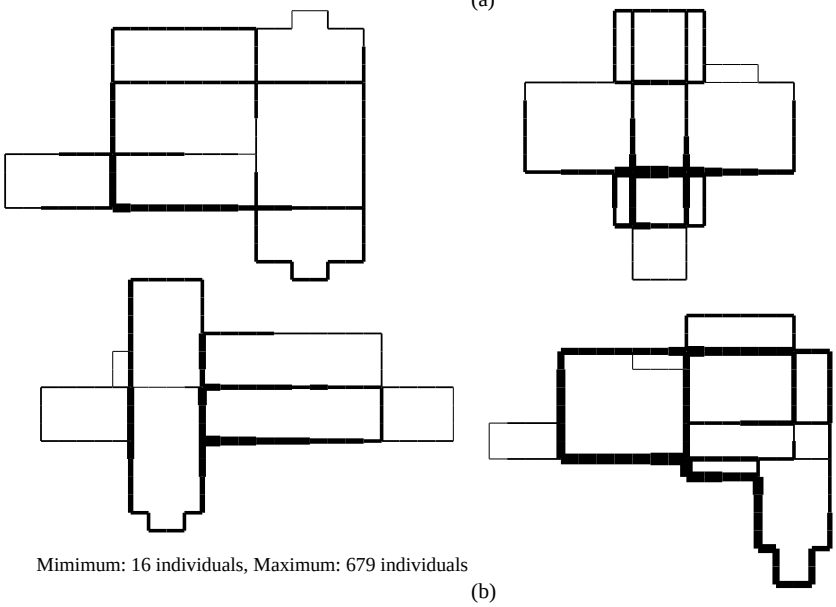
4.4. CREATING NEW FLOORPLANS USING THE EVOLVED REPRESENTATION

In the second phase, the representation evolved from the example cases is used to create new designs that show similarities in style to the example cases. For this, a standard evolutionary system is used, with the set of basic and evolved genes used in the coding.



Mimimum: 4 individuals, Maximum: 3242 individuals

(a)



Mimimum: 16 individuals, Maximum: 679 individuals

(b)

Figure 8. Distribution of individuals in population, (a) without fitness sharing, (b) with fitness sharing. Thicker lines represent more individuals covering that segment.

4.4.1. *Fitnesses*

The evolved coding, as exemplified in Figure 9, captures information about shape and function of parts of the example designs. However, the way these parts are assembled to create new designs is only influenced by the fitness function that

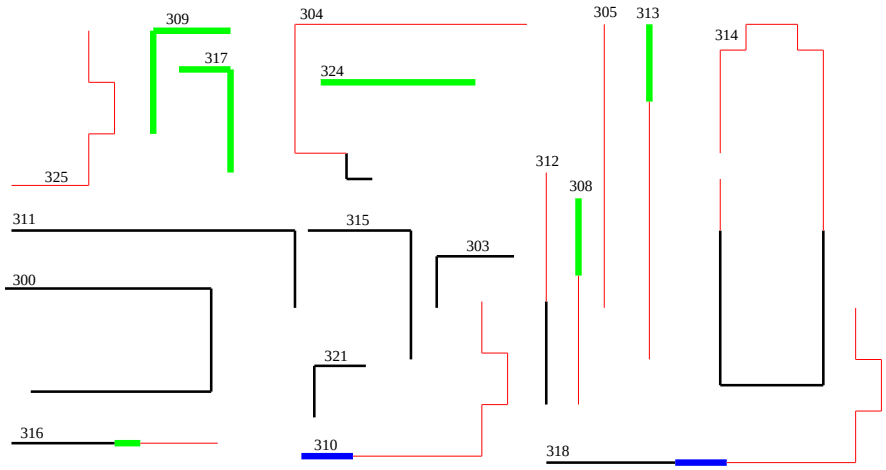


Figure 9. Examples of evolved genes created from the example designs shown in Figure 4. The labels refer to the numerical sequence in which the genes evolved.

evaluates new designs. This means that any gene that codes for a room of a certain type, for example, has no influence on what other room is next to it; and there is nothing preventing a design from using two evolved genes that each include a fireplace. As a result, topological constraints are not automatically satisfied by using the evolved coding. If designs created by the evolutionary system are to fulfil topological constraints, they have to be included in the fitness function (see Section 5 for how we plan to integrate more topology information into the evolved coding).

Frank Lloyd Wright's prairie houses follow a number of topological constraints, and they all have to be made part of the fitness. For the results presented here, fourteen different aspects influence the fitness. The following list shows the fitnesses used:

- One porch, size between 9 and 12 units
- porch connected to living area, and not connected to service area
- two to four rooms in the service area, total size between 45 and 60 units
- two to four rooms in the living area, total size between 55 and 70 units
- only one service and one living area, i.e. all rooms of that type are connected
- one fireplace, two units length, between living and service area
- no 'dead ends', i.e. lines that do not enclose any room.

4.4.2. *Pareto optimization*

For a human designer, it is relatively easy to find designs that fulfill all the fitness conditions. For a standard evolutionary system, the fitnesses have to be integrated into one fitness. This could for example be done by calculating a value between

0 and 1 for every individual fitness, and adding or multiplying them into a single fitness value. Unfortunately, by integrating all fitnesses into one value, the information about what fitness conditions are fulfilled and what conditions are not is lost to the system. As a result, the system ends up converging towards a population that is good in some respects while individuals good in different aspects are lost. Even after a very high number of individuals have been produced, the system is not able to find satisfying solutions.

A better way to handle a high number of individuals is therefore to utilize 'Pareto optimization' (see for example Radford and Gero, 1988). In a Pareto optimization process, only a partial ranking between two individuals can be established. If two individuals are compared, one individual is better than the other (dominates it) only if it is better or equal in all fitness criteria and better in at least one criterion. The comparison therefore often ends up in a draw. To select individuals that are used to produce offspring in the genetic operations, two individuals are picked randomly from the population and compared to a randomly picked reference subset (10% of the population). If one of the individuals is dominated by one of the reference individuals while the other is not, the second individual is selected as the parent. Otherwise, neither of the individuals is preferred.

This selection alone is not sufficient to prevent all individuals clustering as a small subset of possible, good solutions. As an additional measure to prevent convergence, 'niching' is used (Horn and Nafpliotis, 1993). Here, candidate individuals are compared with a number of other individuals in the population. For every individual, the distance between the fitness values is calculated. The number of individuals with a distance smaller than a threshold value is called the niche-count. In niching Pareto optimization, in order to select between two individuals that either both dominate the reference set or are both dominated by at least one individual in the reference set, the individual is chosen that has a smaller niche-count.

If a newly created individual dominates another individual in the population, it replaces it. If not, and the new individual is dominated by at least one other individual in the population, it is rejected. The third possibility is that the new individual populates a new part of the Pareto optimal front, and is therefore neither dominated nor dominates another individual. In this case, the individual has to be added to the population without replacing another individual, leading to a growing population. As an example, in one of the runs presented below, the population grew from 500 to 1581 individuals.

4.4.3. *Results*

Two runs where done using a set of 326 individuals created from the floorplans in Figure 4.

Run 1 ran for 60.677 loops, each loop creating two offspring individuals. The initial population was 1.000 individuals, the final population consisted of 1.652 individuals. From some 120.000 produced and tested individuals, 14.359 indi-

viduals where good enough to be introduced into the population.

Run 2 ran for 99.207 loops, the population grew from 500 to 1.581 individuals. Of the nearly 200.000 individuals produced, 12.502 made it into the population. Again, all 326 evolved genes where used.

The first result from Run 1 (Figure 10(a)) has a perfect fitness. The fitness function does not check if the fireplaces are straight, therefore a corner-fireplace could result. Since none of the floorplans in the example drawing have a corner fireplace, this feature cannot have been part of the evolved coding. It therefore must be coded in basic genes. The second-best result from Run 1 (Figure 10(b)) has a penalty due to one segment of 'dead end' close to the fireplace, but fulfills all other fitness criteria.

Both results of Run 2 (Figure 10(c) and (d)) have perfect fitnesses. Again, the system has taken advantage of a small weakness in the fitness function, that allowed it to put the porch inside the living space.

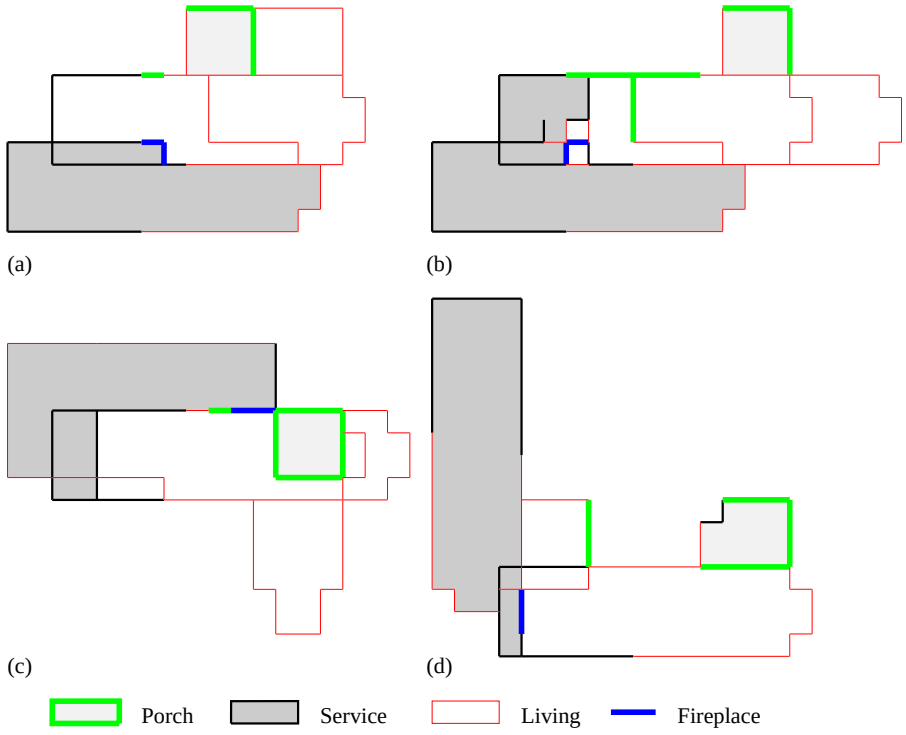


Figure 10. Floorplans created using the evolved genes from the example designs shown in Figure 4, (a) and (b) initial population 1000 individuals, (c) and (d) initial population 500 individuals.

Figure 11 shows how one of the results (the second of Run 1, see Figure 10(b)) can be extended into three dimensions by a graphic artist. The resulting house is obviously quite similar to the Thomas house (see Figure 5).

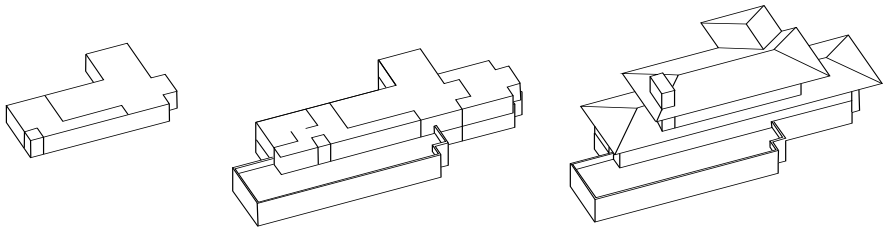


Figure 11. Floorplan from Figure 11 (b) manually extended into three dimensions; shown are bedroom level, main floor level, and roof view.

5. Discussion

5.1. FRANK LLOYD WRIGHT PRAIRIE HOUSE PLANS

As described above, no aspects of the topology are coded in the evolved coding. This results in the fact that some aspects of the example design that the system could have learned have to be added as fitness. It also shows in the results: shapes that have been outer walls in the original drawings are used in the inside (e.g. the stepped line separating the right two parts of the living room in Figure 10(b), or the outer walls of the porch in Figure 10(a)).

5.1.1. *Possible improvement: more line types*

One way to improve the 'knowledge-content' of the evolved coding is to use a higher number of line types. Different line types could be used depending on the functions of both of the rooms a wall separates, and again different line types for outer walls. This way, knowledge that for example three out of four sides of the porch are outer walls, and the fourth wall is a wall to a living space, could be integrated into the evolved coding.

An example of the Thomas house drawn with this enhanced coding is shown in Figure 12.

To realize this, no changes other than changing the parameter for the number of line types used and modifying the example drawings are required in the first step. In the second step, the fitness function would have to be added that checks if lines are used in a correct context.

A system using this coding would avoid problems like the two problems with the design results in Figure 10. It would also reduce the number of fitness criteria required.

5.2. LEARNING REPRESENTATIONS

What has been successfully presented has been both the concepts and a demonstration example of the evolutionary learning of a genetic representation of a set

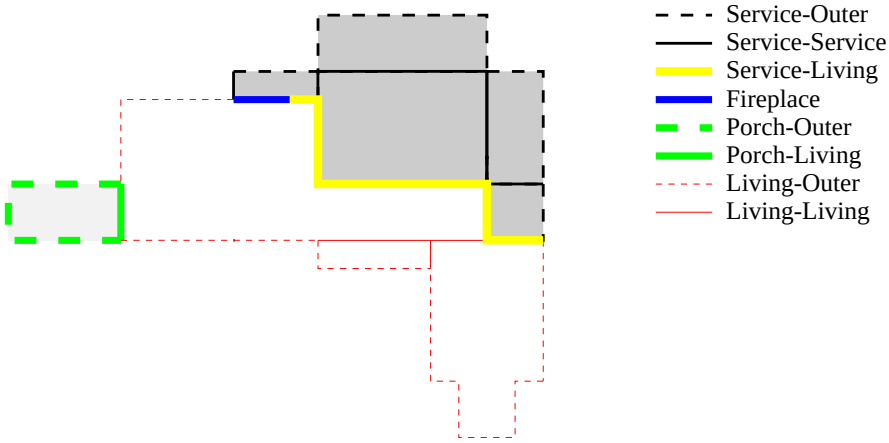


Figure 12. Thomas house drawn as design example with increased number of line types.

of building layouts. This representation can then be used to generate layouts with a similar style. The approach is based on the use of genetic engineering concepts to evolve not just the solution to a problem but also to allow the genes which are used to represent it also to evolve. This results in the evolution of complex genes, genes which are capable of forming increasingly large and complex parts of the phenotype or design. Evolutionary systems commence with a knowledge-lean representation that often contains little or no domain knowledge. What is happening here is that the evolved genes increasingly contain knowledge about the domain under consideration. They turn a knowledge-lean representation into a knowledge-rich representation. From an evolutionary viewpoint there are two distinct activities. In the first the genes are allowed to evolve with a fitness associated with the designs which act as exemplars. Then these evolved genes are used to generate designs with a completely different set of fitnesses. The resulting designs embody the knowledge which has been encoded in the evolved representation. This opens up possibilities in case-based design as well as an alternative approach to the generation of design grammars.

Acknowledgements

This work is supported by a grant from the Australian Research Council and by a University of Sydney Postgraduate Research Award.

References

Buelinckx, H.: 1993, Wren's language of city search designs: a formal generative classification, *Environment and Planning B* **20**, 645–676.

Chan, C.-S.: 1992, Exploring individual style through Wright's designs, *Journal of Architectural and Planning Research* **9**(3), 207–238.

Chan, C.-S.: 1995, A cognitive theory of style, *Environment and Planning B: Planning and Design* **22**, 461–47.

Chase, S. C.: 1989, Shapes and shape grammars: from mathematical model to computer implementation, *Environment and Planning B* **16**, 215–242.

Gero, J. S.: 1994, Towards a model of exploration in computer-aided design, *in* J. S. Gero and E.Tyugu (eds), *Formal Design Methods for CAD*, North-Holland, Amsterdam, pp. 315–336.

Gero, J. S. and Schnier, T.: 1995, Evolving representations of design cases and their use in creative design, *Preprints Computational Models of Creative Design.* (to appear).

Horn, J. and Nafpliotis, N.: 1993, Multiobjective optimization using the niched pareto genetic algorithm, *Technical Report 93005*, Illinois Genetic Algorithms Laboratory (IlliGAL), University of Illinois at Urbana-Champaign.

Knight, T.: 1989, Transformations of De Stijl art: the paintings of Georges Vantongerloo and Fritz Glarner, *Environment and Planning B* **16**, 51–98.

Knight, T.: 1990, Mughul gardens revisited, *Environment and Planning B* **17**, 73–84.

Knight, T.: 1994, Shape grammars and color grammars in design, *Environment and Planning B: Planning and Design* **21**, 705–735.

Knight, T. W.: 1981, The forty-one steps, *Environment and Planning B* **8**, 97–114.

Koning, H. and Eizenberg, J.: 1981, The languages of the prairie: Frank Loyd Wright's prairie houses, *Environment and Planning B* **8**, 295–323.

Langton, C. G.: 1988, Artificial life, *in* C. G. Langton (ed.), *Artificial Life*, Vol. VI of *SFI Studies in the Sciences of Complexity*, Addison-Wesley, Reading, pp. 1–47.

Mackenzie, C. A.: 1989, Inferring relational design grammars, *Environment and Planning B* **16**, 253–287.

Radford, A. D. and Gero, J. S.: 1988, *Design by Optimization in Architecture and Building*, Van Nostrand Reinhold, New York.

Stiny, G.: 1980, Introduction to shape and shape grammars, *Environment and Planning B: Planning and Design* **7**, 343–351.

Stiny, G. and Mitchell, W. J.: 1978, The Palladian grammar, *Envirnoment and Planning B: Planning and Design* **5**, 5–18.

Stiny, G. and Mitchell, W. J.: 1980, The grammar of paradise: on the generation of Mughul gardens, *Envirnoment and Planning B: Planning and Design* **7**, 209–226.