Shapes, grammars, constraints and policies

Manuela Ruiz-Montiel, Lawrence Mandow, José-Luis Pérez-de-la-Cruz, and J. Gavilanes

Universidad de Málaga
{mruiz,lawrence,perez}@lcc.uma.es,jgavilanes@uma.es

Abstract. A proposal is presented to enhance the usability of shape grammars by using explicit constraints and policies on rule selection.[1]

Keywords: shape, grammar, constraints, learning

1 Introduction

The problem of computational design [8], [4] has been considered from many perspectives since the decade of 1960. Our research tackles the problem of generating shapes for a design, given a certain set of criteria. Different variants of this problem are of interest in many fields like engineering, architecture or packaging, where geometric design plays a crucial role; certainly, an assistant design system able to automatically generate and propose design alternatives can be quite worthy for design stakeholders.

In this paper we report our experiences with the application of *shape grammars* ([11], [9], [10]) for the synthesis of designs using additional control mechanisms. Shape grammars are a well-known formalism to describe the generation of forms. This formalism has been used in the literature for many design tasks such as architectural design [3], industrial design [6] or virtual reality [7]. Additionally, Agarwall et al. [1] suggested the use of shape grammars as a framework for geometric-based engineering expert systems. Along these lines, shape grammars have been used to generate geometric designs according to functional requirements in a number of ways.

The use of shape grammars for geometric design implies taking into consideration some guidelines or sets of requirements suitable for the specific domain the designer is working in. For example, in the field of architectural design not every derived shape may be a feasible floor plan for a housing unit: there are room adjacencies and minimum area constraints that must be satisfied, among many other conditions.

According to Knight "different approaches to connecting grammars and goals have been suggested. One approach is direct. It involves writing rules with the foreknowledge that the generated designs will meet, or start to meet, given goals. In order to do this, the behaviors and outcomes of rules must be predictable in some way" [5]. The execution of such *expert rules* would thus lead to acceptable

[1] Supported by grant TIN2009-14179, Plan Nacional de I+D+I, Gobierno de España.

designs: the structure of the shape grammar itself guarantees that the produced designs will be feasible. Nevertheless, this approach can suffer from two major shortcomings: (1) expert shape grammars can be very difficult to create, modify and maintain. A great deal of control information (in the form of *labels* or *control marks*) must be included into the rules in order to restrict execution possibilities; (2) we are at risk of sacrifying the divergence capacity of shape grammars in the pursuit of predictability. In this way we are missing one of the reasons for using this formalism as design framework, that is, the possibility of obtaining many different, unexpected solutions.

In this paper we propose the use of *weak* shape grammars whose rules have little or no control knowledge. In comparison to expert grammars, weak grammars generate a greater number of designs. Unfortunately, many of them will be unfeasible solutions. To overcome this problem, two additional mechanisms are proposed: (1) constraints and goals; (2) policies on rule applications. The presentation of these mechanisms is the aim of this paper. We will consider an example in the domain of architectural design. In the following section we will present the ideas concerning constraints and goals and in section 3 analogously for rule policies and learning.

2 Constraints and goals

2.1 Shape grammars

In order to formalize the concept of shape grammar some ideas must be defined first. A *segment* or *line* l, $l = \{p_1, p_2\}$ is defined by any pair of two distinct points p_1 and p_2, the so-called *end points* of the line. A *shape* is defined by a finite set of distinct lines that cannot be combined to form another line, that is, they are maximal, since they are not part of longer lines inside the shape. The representation of a shape is thus unique.

A *labelled shape* consists of two parts: a shape and a set of *labelled points*. A labelled point (p, A) is a point p with a symbol A. A labelled shape σ is an ordered pair $\sigma = \langle s, P \rangle$ where s is a shape and P is a finite set of labelled points.

We define a shape grammar as the 4-tuple $\langle S, L, R, I \rangle$:

- S is a finite set of shapes
- L is a finite set of symbols
- R is a finite set of rules $\alpha \to \beta$, where α is a non-empty labelled shape and β is a labelled shape
- I is a non-empty labelled shape, called *initial shape* or *axiom*.

A rule applies to a shape γ when there is a transformation τ such $\tau(\alpha)$ is a subshape of γ, that is, $\tau(\alpha) \subseteq \gamma$ (a labelled shape s_1 is subshape of another labelled shape s_2 if and only if every line and every labelled point of s_1 is in s_2). Usually, τ is supposed to be a general geometric transformation. In this work, the considered transformations are translations, rotations and regular scales.

The labelled shape produced by the application of the rule $\alpha \rightarrow \beta$ to the labelled shape γ under transformation τ is given by the expression $\gamma - \tau(\alpha) + \tau(\beta)$. This labelled shape is obtained by substituting the appearance of $\tau(\alpha)$ inside γ with $\tau(\beta)$. More details of these definitions can be found at [9].

2.2 Constraints and goals

Ideally, all shapes generated by a shape grammar should result in feasible designs. However, creating a grammar that *implicitly* incorporates all design constraints and goals for a given domain is generally a very difficult problem. We propose in this section the use of *explicit* design constraints and goals to control the execution and avoid the production of unacceptable designs. Hopefully, this explicit knowledge (constraints and goals) will be easier to elicit, debug and maintain than "wired" control labels set inside the shapes in the rules.

The aim of constraints and goals is to reduce the search space introduced by the shape grammar. In this context, a *state* is every potential shape produced by the grammar, by the application of its rules starting from the axiom. The *state space* is thus the set of all possible states. Using weak shape grammars, that is, that do not use control labels in order to produce feasible designs, leads to a vast state space in which the majority of states would not be acceptable.

A constraint (or goal) is a predicate that returns `true` or `false` when applied to a shape. The process of producing a shape taking into account a set of constraints and goals starts by choosing a random pair *(rule, transformation)* to be applied. When a single pair is applied, two processes trigger sequentially:

1. *Constraint checking*: in each step of the process all the problem constraints must be satisfied. If one constraint is violated when applying a pair (*rule*, *transformation*), then the rule is not applied and the process chooses another pair. If no more alternatives are available, then the process *backtracks*, that is, undoes the previous application and tries another alternative.
2. *Goal checking*: if all the problem constraints are satisfied, then the goals are checked. If every goal is satisfied, then execution stops; but if there are still unsatisfied goals, the execution goes on. If at a certain moment no more pairs *(rule, transformation)* are available and goals have not been fulfilled, then the process backtracks.

In figure 1 we can see a diagram of a design generation process following this approach. Notice that the complete search space could be much wider than the fragment which is truly explored. That is, constraints *prune* the search tree. When the algorithm runs out of possible rule derivations and either goals or constraints are not fulfilled, then no rules are applied, returning "failure".

Backtracking search has exponential time complexity, so it is not evident that in real cases the above described process can be performed in a reasonable amount of time. We will show that by the choice of suitable constraints/goals and the judicious ordering of rules the computation of a feasible design can be sometimes done in a reasonable amount of time.

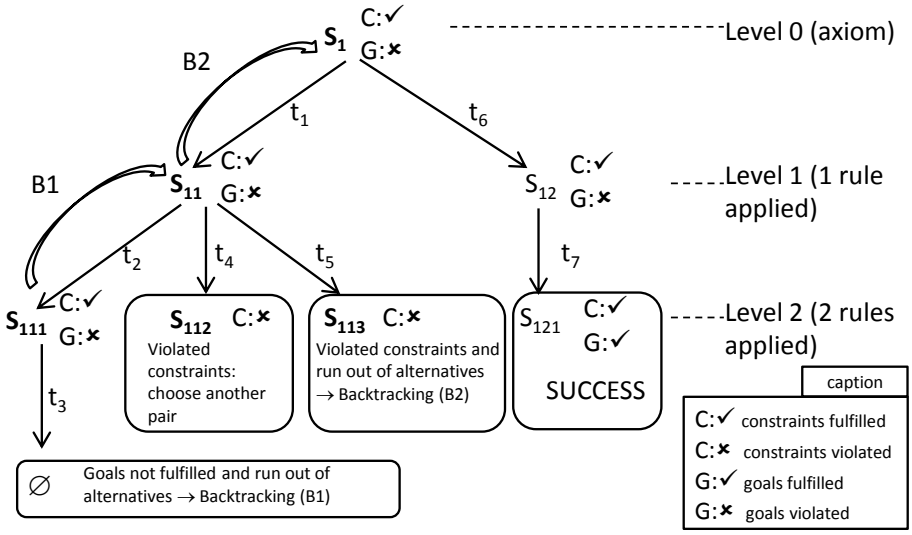


Fig. 1. Schematic example of the generation process

Kitchen requirements
-Minimum linear space must be at least of 6 modules of 60x60 cm
-Minimum distance between modules and walls: 1,10 m
-Minimum distance between modules: 1,10 m
Bath requirements
-At least two modules of 90x180 cm must exist
Non-specialized spaces requirements
-The area of each non-specialized space must be bigger than 9 m^2
-A 2,8 m-diameter circle should fit inside each non-specialized space
Complementary spaces requiremens
-A support space that allows the circulation between spaces must exist

Table 1. Requirement set for a single-family basic housing unit (adapted from [2])

2.3 An example

We have developed a generic interpreter for rules, constraints and goals that performs the implementation of these kind of shape grammars[2].

By means of this tool we have implemented the housing program proposed by the studio Montaner & Muxí [2] for the regional government of Andalusia (Spain). This program details the criteria that a *basic housing unit* must fulfill depending on the number of people that are going to live in it. Table 1 summarizes the imposed constraints.

The final outcome of our program should be a set of acceptable floor distribution schemes in 2D (in the following, *schemes*) for basic, two-people one-story housing units. Our system also considers two additional constraints: (1) the entrance to the house must be near both the kitchen and the distribution hall; and (2) distance between parallel walls must be greater than 1.5 meters.

The process is sequentially structured in seven main phases:

- Phase 1: labelling the distribution hall.
- Phase 2: placing the first kitchen module.
- Phase 3: placing the rest of the kitchen modules.
- Phase 4: placing the first bath module.
- Phase 5: placing the rest of the bath modules.
- Phase 6: pabelling non-specialized spaces.
- Phase 7: labelling the entrance.

There is one grammar for each phase, and there is just one rule for each grammar (see figure 2). Each rule is applied while possible until a final state (for that phase) is reached. To define a final state, constraints and goals for that phase are taken into account. It is also possible to set a maximum number of derivations for each phase.

The axiom for the first phase is a given contour. Each phase applies its grammar using as axiom the shape generated by the prior phase.

Let us start the process from a rectangle of 6 m $\times$ 10 m (figure 4 (a)). The program generates many acceptable schemes, each one in about 10 s. Two of them are displayed in figure 3. For the final presentation of the dwellings, two additional operations are performed: a) erasing of the labels and b) addition of significant text in order to distinguish each space. In figure 3 we can see the schemes before and after applying these additional operations.

3 Policies and learning

Let us start now the process in section 2.3 from the contour in figure 4 (b). Unfortunately, in this case the sole use of constraints and goals cannot avoid the combinatorial explosion, and the program can backtrack for days before a solution is found.

[2] ShaDe 2.0 for SketchUp, `http://www.lcc.uma.es/~perez/ntidapa/`

To avoid this, we consider a different approach that assigns preferences to the application of rules. Since hard-coded preferences are difficult to elicit, we evaluate the compliance of constraints and goals in complete designs, and apply machine learning techniques to learn rule preferences.

3.1 Reinforcement learning

Reinforcement learning [12] is a technique for learning a *policy* for a given problem (ideally, an *optimal policy*). A policy defines which action to take for every possible situation with the goal to maximize a given, long term, reward. Initially, each pair (*state*, *action*) is assigned a random *value*, and the policy is determined by the following rule: "in state s choose that action a that yields a pair (s, a) of maximum value". Optimal values are learned automatically through an iterative process known as *reinforcement learning*. Usually it is neccesary to *generalize*, i. e., rather than enumerating values for all possible states, these are assigned values as a function of a limited number of their *features*.

In our case a state is a shape generated by the application of a rule to a prior shape. An action is a pair (*rule*, *transformation*) such that *rule* is applicable to the shape after applying *transformation*. We have used the learning algorithm $Q(\lambda)$ [13] with linear approximators and binary features, as in [12] (p. 213).

3.2 Our example

We have used the same phases and grammars as in section 2.3. Some of the phases use a policy in order to decide which rule application is better. These policies are generated by means of reinforcement learning processes. The considered features and rewards for each phase are described next. The learned policies are linear functions defined over these features.

As an example, we have chosen phase 3 in order to explain the used features and rewards in detail. Phase 3 is in charge of placing the kitchen modules. Its corresponding grammar is depicted in figure 2. As we can see, the rule simply states that an additional module can be put contiguous to an existing kitchen module. Six binary features were selected in this phase, following the recommendations of the guideline in [2]:

- $f_1(s) = 1$ iff every module is inside the axiom contour.
- $f_2(s) = 1$ iff every module is accessible.
- $f_3(s) = 1$ iff the distance between modules and walls is bigger than 1,1m.
- $f_4(s) = 1$ iff the distance between non-contiguous modules is larger than 1,1m.
- $f_5(s) = 1$ iff the modules are at a proper distance from the distribution hall (more than 1,2m and less than 6).
- $f_6(s) = 1$ iff there are at least six modules.

The reward of every state s inside this phase is assigned the following expression: $reward(s) = 3 * f_1(s) + f_2(s) + f_3(s) + f_4(s) + f_5(s) + f_6(s)$. The more

features the state complies with, the more reward it gets. The first feature has more importance than the others, thus it is multiplied by 3 instead of 1.

The rest of features and rewards are gathered in table 2. Learning times for each phase ranged from 60 s to 500 s.

Phase 2
$f_1(s) = 1$ iff the module is at a proper distance from the distribution hall (more than 2m and less than 4).
$f_2(s) = 1$ iff the distance between the module and the walls is larger than 1,1m.
$reward(s) = f_1(s) + f_2(s)$
Phases 4-5
Analogously to 2-3
Phase 6
$f_1(s) = 1$ iff in each label a 3 meter-diameter circle can be centered, without overlapping with walls or modules.
$f_2(s) = 1$ iff there is a label separated from the bath less than 4,5m.
$f_3(s) = 1$ iff there is a label separated from the kitchen less than 4,5m.
$f_4(s) = 1$ iff there are 2 non-specialized labels and they are separated at least by 3m.
$f_5(s) = 1$ iff the distance from each non-specialized label to the distribution hall label is larger than 2m and shorter than 10.
$reward(s) = 3 * f_1(s) + f_2(s) + f_3(s) + f_4(s) + f_5(s)$
Phase 7
$f_1(s) = 1$ iff the entrance is separated at least 1 m from every kitchen module.
$f_2(s) = 1$ iff the distance from the entrance to some kitchen module is shorter than 2m.
$f_3(s) = 1$ iff the entrance is separated at least 4 m from every bath module.
$f_4(s) = 1$ iff the distance from the entrance to the distribution hall label is shorter than 4m.
$reward(s) = f_1(s) + f_2(s) + f_3(s) + f_4(s)$

Table 2. Set of features and rewards for phases 2, 4, 5, 6 and 7. Phase 3 is described in section 3.2

Once all the necessary policies were learned, solutions were generated by executing the generation processes, guided by the policies when necessary. Ten results where generated, being five of them acceptable and the other five unacceptable. Some of them are displayed in figures 5(a) and 5(b). Once the policies were learned, each solution was generated in less than 60 s.

4 Conclusions and future work

We have reported some experiments using weak shape grammars for synthesizing designs in the field of architectural design.

In the traditional approach design goals and constraints are *implicitly* coded inside grammar rules. This paper explores two different alternatives using *weak* rules and *explicit* goals and constraints.

The first alternative uses a simple backtracking algorithm in which compliance with goals and constraints is always enforced. This scheme seems to work fine in underconstrained problems.

The second one uses reinforcement learning to automatically learn to apply rules that implicitly lead to acceptable shapes. The learning process is governed by rewarding those generated schemes that comply with *explicit* constraints and goals.

In our experiments goals and constraints have been organized in a hierarchical fashion reflected in seven sequential *phases*, each one implemented with a one-rule grammar. Albeit this separation manages to reduce the ramification factor of the search space, it also introduces important shortcomings: in the case of shape grammars enhanced with policies, the learning process is not global, and thus the policies cannot reflect global issues. For example, the ideal policy would tell us something about how to place the kitchen regarding the future placing of non-specialized spaces.

In future works we hope to address both the problem of combining reinforcement learning with backtracking and that of learning globally optimal preferences.

References

1. Agarwal, M., Cagan, J.: On the use of shape grammars as expert systems for geometry-based engineering design. AI EDAM 14(05), 431–439 (2000)
2. arquitectes", M.M.: Propuesta de nueva normativa de viviendas. Tech. rep., Dirección general de ordenación del territorio, Junta de Andalucía (2008)
3. Duarte, J.P.: A discursive grammar for customizing mass housing: the case of Siza's houses at Malagueira. Automation in Construction 14, 265–275 (2005)
4. Eastman, C.M.: Cognitive processes and ill-defined problems: a case study from design. In: IJCAI'69. pp. 669–690 (1969)
5. Knight, T.W.: Applications in architectural design, and education and practice. Tech. rep., NSF/MIT Workshop on Shape Computation (1999)
6. Lee, H.C., Tang, M.X.: Evolving product form design using parametric shape grammars integrated with genetic programming. Artificial Intelligence in Engineering Design, Analysis and Manufacturing 23, 131–158 (2009)
7. Muller, P., Wonka, P., Haegler, S., Ulmer, A., Gool, L.V.: Procedural modeling of buildings. ACM Transactions on Graphics 25(3), 614–623 (2006)
8. Simon, H.A.: The sciences of the artificial. MIT Press, Cambridge, MA. (1968)
9. Stiny, G.: Introduction to shape and shape grammars. Environment and Planning B 7, 343–351 (1980)
10. Stiny, G.: Shape. Talking about seeing and doing. MIT Press, Cambridge, Ma. (2006)
11. Stiny, G., Gips, J.: Shape grammars and the generative specification of painting and sculpture. In: Information Processing 71, pp. 1460–1465. North-Holland (1972)
12. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction. MIT Press, Cambridge, Ma. (1998)
13. Watkins, C.J.: Learning from delayed rewards. Ph.D. thesis, University of Cambridge (1989)

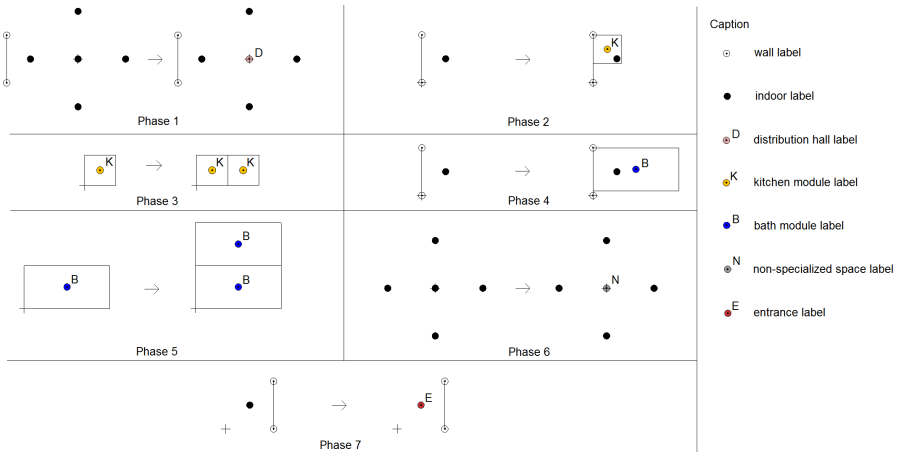


Fig. 2. Grammars

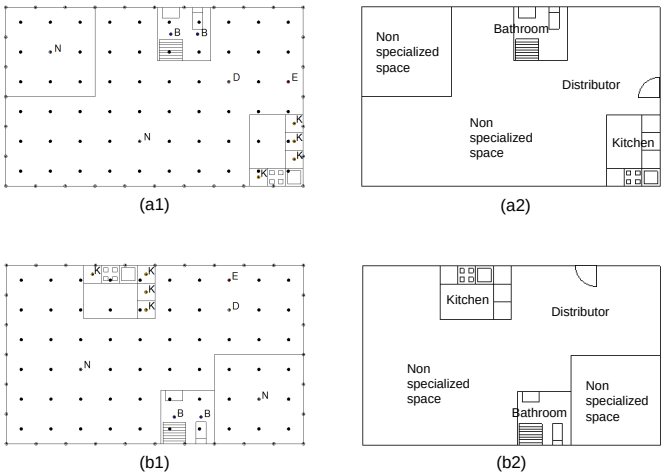


Fig. 3. Two schemes generated with constraints and goals

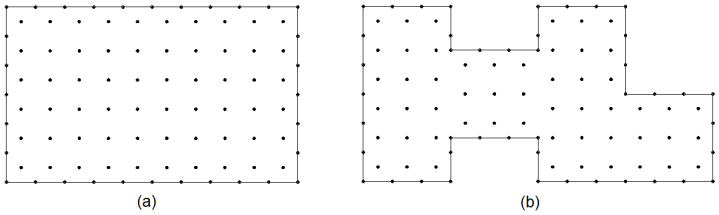
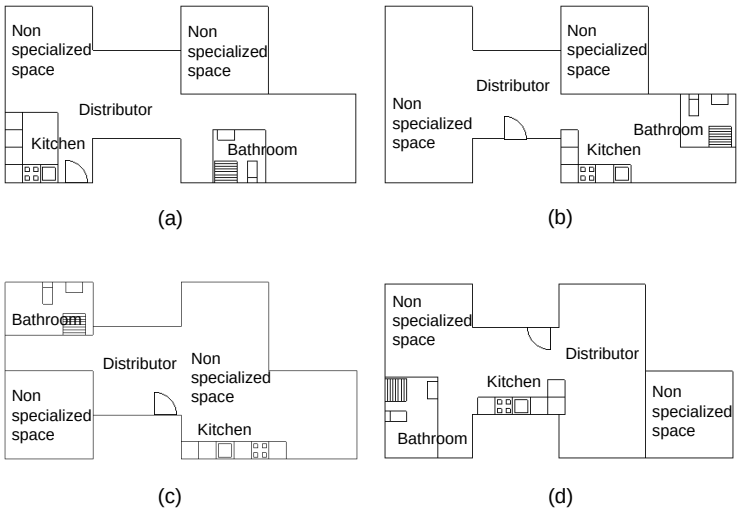
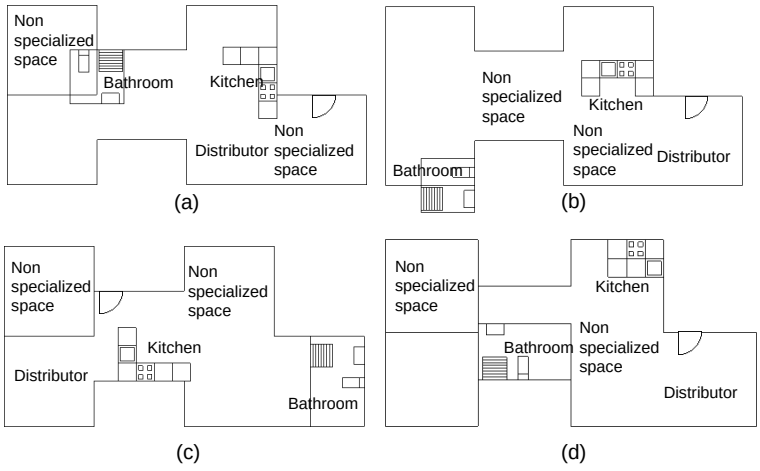


Fig. 4. Starting contours



(a) Some good schemes



(b) Some bad schemes

Fig. 5. Some good and bad schemes generated with learned policies