Towards controlled grammars

Approaches to automating rule selection for shape grammars

Thomas Grasl¹, Athanassios Economou²
¹SWAP Architekten ²Georgia Institute of Technology
¹www.swap-zt.com ²www.coa.gatech.edu
¹tg@swap-zt.com ²economou@coa.gatech.edu

An approach to automating the rule selection process of shape grammars is introduced. A shape grammar interpreter is extended by a computational framework to allow the rule selection to be executed by agents. Hereby each agent is based on a different paradigm taken from the field of artificial intelligence. The results are compared.

Keywords: Shape grammar, Rule selection, Conflict resolution

INTRODUCTION

Which rule to apply, when, where and how is a decision the user of a shape grammar is continuously confronted with. In most cases the designer selects a rule either to playfully explore the solution space offered by the grammar or to systematically explore how this rule helps approach a desired state of the system. Historically this may partly be due to the lack of computer implementations, and hence the lack of situations where automation of the rule selection can easily be achieved. However, recently shape grammar implementations have received increased attention (Grasl and Economou, 2013; Grasl, 2012; Strobbe et al., 2013; Correia et al., 2012; Jowers and Earl, 2011; Trescak et al., 2009).

This has resulted in several projects that either implement specific grammars or that can be used to create and edit grammars. In any case there is a system with data structures and working memory which can be introspected by rule selection algorithms. Here such methods of automating the rule selection process are investigated.

The methods are divided into two approaches: Extensive enumeration and goal directed generation. Both approaches can be easily found in literature (Stiny and Mitchell, 1978; Flemming, 1981; Koning and Eizenberg, 1981; Cagan and Mitchell, 1993; Schmidt and Cagan, 1997; Wonka et al., 2003).

Complete design catalogue enumeration is interesting because it points to significant questions pertaining to the evaluation of the grammar: How strict or expressive is the grammar? Given some constraints, what is the approximate number of possible solutions? Are some rules more productive than others and how? Enumerations need not be restricted to the complete language specified by the grammar. They can also be given for any desired subset of the complete set of designs specified by the grammar. For example, an enumeration of all possible 5 x 3 grid configurations of dimensionless Palladian designs can be ad-hoc filtered to provide the complete catalogue of all 5 x 3 Palladian plans with a central T-room, or conversely expanded to provide the complete catalogue of all Palladian plans with a central T-room up to an n x m grid systems. Clearly, enumerations are interesting only when the results are small.
Bigger numbers are useful only to the extent they give a sense of the upper boundary of the set of possibilities specified by the grammar. Smaller numbers are useful because they allow an actual, constructive and meaningful visual encounter with these possibilities.

Goal directed design generation is essentially an attempt to reduce the grammars solution space and filter out only those solutions that are of interest. As above, such generation may readily point to significant questions pertaining to the evaluation of the grammar: Are there specific optimization criteria to control the derivation of the shapes? Are these criteria inherent in the formulation of the grammar or external to it? Can they be introduced ad hoc and at any time during the design process or are they required to be specified in the onset of the grammar? And as above, goal directed generation techniques may be given for any desired subset of the complete set of designs specified by the grammar. For example, the needs of a given geometry of a site, a given area requirement, and a given orientation may severely influence the generation and fitness of possible Palladian plans within this design context. More profoundly such design contexts profoundly affect the choice of the rules that lead to the making of the candidate designs. Clearly, grammars guarantee well-formed output. Generating a semantically meaningful result is mostly left to the user, so this is one of the tasks assigned to the program controlling the rule selection.

The well-known Palladian grammar (Stiny and Mitchell, 1976) will be used to demonstrate concepts in this paper. The demonstrated solutions are implemented as an extension to the shape grammar application GRAPE (Grasl and Economou, 2013a and 2013b), thus it is important that they are applicable to arbitrary grammars. Ideally no domain knowledge should be hard coded into the rule selection system. This will sometimes require solutions to be scriptable, other times it may be sufficient to use dedicated control rules.

**FRAMEWORK**

GRAPE is extended by a framework to support rule selection agents. The agents all use the same interface to communicate with the shape grammar engine, but each is based on a different paradigm from the field of artificial intelligence. Agents are not omniscient. They do not have absolute knowledge about what is happening in the environment (GRAPE), rather a sensor-actuator design (Figure 1) as described by Russel and Norvig (2013) was implemented. For example, if a user selects a rule, then the agent can perceive a change in the state of the derivation and it could reason about what caused the state change, but it will not directly know which rule was applied. This has several advantages. For one the agents are more robust and independent, for example they can take over from a manual derivation at any time. Additionally the results are more easily transferable to other systems.

![Figure 1](image)

The common sensor-actuator agent design

The sensors available to the agents are inquiries which are answered by the shape grammar engine. How they are answered is essentially opaque and not of further interest to the agent. GRAPE is based on graph grammars, so the application has paral-
lel representations of the grammars derivation. One graph representation and another based on shapes. The framework offers an interface through which the agent can inquire on things like the number and location of labels, lines and so forth. Essentially mirroring what a human user can perceive through her eyes.

More complex and perhaps domain specific queries can take on the form of rules. They can be edited through the editor just like the shape grammar’s actual rules. These logic or control rules can be used to determine whether a specific condition is met or to return results. Generally they follow the schema

\[ x \rightarrow x \] (1)

That is, a rule that does not make any changes to the derivation, and is only used to determine whether a pattern can be found and in extension how many matches are found. Moving domain specific queries to the grammar is one possibility of keeping the agent free of domain knowledge, thus enabling each agent to be configured to work with any grammar.

The only actions available to the agent are the application of a rule and the undoing of such an application. This is sufficient for the agent to generate a derivation and to adopt a try and error approach if appropriate. The agent can apply a rule, analyse the result and undo the rule application should it have led down a dead end.

Chase (2002) has outlined various strategies of dividing the work between the user and the computer. Options range from fully manual paper and pencil operations to fully automated design generators.

Likewise the agents here offer various levels of support and interactivity. Some will generate entire derivations if given a chance, others will only select the next rule in order to help a user that has got stuck, again others will only observe the user interacting with the system and will only intervene if the user leaves some predefined boundary.

**AGENTS**

The following is an overview of the different agents. Each is based in a different technological approach and they may follow different goals in selecting the rules.

**Extensive Enumeration**

Enumeration of possibilities has always been an important part of shape grammars and of design computation in general. An early example is the counting of Palladian plans by Stiny and Mitchell (1978).

Coming up with a definite number is however not always as straight forward as it might appear at first glance, even if the symmetry of the final derivation is disregarded and all isomorphisms are counted (Economou and Grasl, 2012). There are two issues: Firstly, there are mostly several ways to apply a rule at any given time, and secondly, results can be identical despite a different order of rule application. So applying the same rules twice can lead to different results, and applying different rules, or the same rules in a different order, can lead to the same result.

Some enumerations can make use of specific features of a given grammar, but here an all-purpose tool is required. A grammar can be seen as a tree graph, the initial shape is the root, each rule application is an edge and every derivation is a leaf. So a straightforward approach is to devise some mechanism of traversing the tree and writing out all derivations, possibly limited to a specific subset. This will most likely lead to numerous duplicates, which will have to be filtered out in another step. It is a brute force approach and potentially computationally expensive, but perhaps the only viable one if it is to be applicable to all grammars.

Here a backtracking algorithm (Knuth, 2011) was chosen, it is essentially a depth first search with some additional rules to prune the search tree in order to reduce running time. To configure the agent it must be passed three bits of information.

- A sequence of rules to execute before starting the enumeration
- A set of rules over which to enumerate
- A condition that designates a solution

It then offers a mechanism for cycling through the solutions, generating the derivations on the fly. Duplicates are filtered out by comparing a prospective solution to a database of already found solutions.

The following configuration will be used as an example:

- The sequence of rules to generate a 5 by 4 grid
- All rules of the collapse phase
- If the central label is marked E and the layout consist of exactly five spaces, then a solution has been found

The results shown in figure 2 are necessarily reminiscent of "Counting Palladian plans" by Stiny and Mitchell (1978), but it is interesting to note that arbitrary conditions can be defined to filter the solutions even further.

It is difficult to provide an estimate of the number of derivations that is general enough for all grammars. Instead a percentage is given, quantifying the amount of the solution space that has already been searched. In general the agent has proven to be a useful tool to quickly get an idea of the possibilities offered by a grammar, or by a subset of the grammars rules. Time complexity can be an issue, but by restricting the agent to the subset of interest, results can mostly be achieved within reasonable time frames.

**Goal directed generation**

In many ways goal directed generation is the more interesting problem. The aim will mostly be to generate meaningful and good results, instead of just well formed ones. This will mostly require some domain knowledge, making it difficult to provide general solutions.

**Probabilistic approaches.** One fairly easy way of selecting rules is to use a random number generator. The result of a naïve, purely random approach of selecting fifteen rules for the

Palladian grammar is shown in figure 3a. The derivation cannot progress because certain obligatory rules are not executed or are called in the wrong order. The performance of such a rule selection strategy largely depends on the structure of the grammar. Given enough time the agent can improve upon figure 3a, and will eventually select the required obligatory rules to continue the derivation. Nevertheless this agent can be seen as the worst case scenario; ob-
viously all following agents should outperform it.

Two steps up is to use sequenced and weighted randomness (Figure 3b), which means creating a sequence of pools out which to select rules and to assign a different probability to each rule. The sequence will most likely be based on the structure of the grammar, the weights can be based on intuition or on an analysis of the body of work a grammar is attempting to model. The generation is then essentially to pick a variable number of rules from each pool. This approach can easily be formalized, to enable the designer to control the generation, either through a user interface or via a simple scripting language. The City Engine software uses a similar mechanism (Wonka et al., 2003).

Such an approach of course has little intelligence above that embedded in the rules themselves. The derivation in figure 3b is complete, and even better results can be achieved with this approach, but this is mainly thanks to the well devised phasing of the Palladian grammar and its cleverly constructed rules. Still the derivation suffers from a lack of doors and an unconventional layout that most likely would not have pleased Andrea Palladio, things that are hard, if not impossible, to control with shape rules alone.

Genetic algorithms. Here two different approaches are proposed. If it is possible to describe a fitness value, including a method to determine the fitness of a derivation, then an automatic approach can be taken. Again state labels can be used to hold information relevant for calculating the fitness. Figure 4 shows a sequence of individuals taken from the genetic algorithm agent. The goal was to find individuals based on a 5 by 3 grid with eight rectangular spaces, one complex space and a portico.

Alternatively one can chose a user guided approach; here the user manually selects the most promising derivations from a given set to be used as parents for the next generation. The human eye acts as the fitness function, similar to Richard Dawkins'
biomorph application described in 'The Blind Watchmaker' only that several individuals can be selected as the parents for the next generation. Again this is a good all-purpose, easy to implement method, which can be used to explore a grammar's solution space. It also has a nice level of interactivity.

A certain amount of logic may also be embedded in the mapping from genotype to phenotype, such as guaranteeing that obligatory rules are executed. Currently the mapping is hard coded, but it is fairly straightforward to implement a generic mapping and then to offer the designer control over this phase via a user interface. Several mappings are possible. Currently the genome is a string of bits. Some bits determine whether or not a rule is to be applied, other bits are decoded to integers and are used to select the appropriate match. Obligatory rules can be activated by setting the appropriate bits and declaring them immutable.

**Rule based systems.** This is an especially intriguing approach for shape grammars, since the control logic would be based on a similar paradigm as the shape grammar rules themselves. Depending on the exact implementation of the shape grammar interpreter, the control logic could even use the same underlying mechanisms. For example, GRAPE is based on graph grammars, which can easily be adapted as the engine for a rule based system.

For this project it was decided to merge the two systems into one graph grammar, this facilitates the inspection of the shape grammar using control rules, to decide which action to take next.

In order to obtain varying results, first a set of parameters is randomly generated. These determine things like the size of the grid, whether a portico should be attached and whether there should be an l-shaped space or not. These parameters are then used to guide the agent's actions. The creation of an l-shaped space requires several rules to be selected in the right order and to be applied to the right match, so some reasoning will become necessary. A backward chaining technique is used to find the appropriate rule sequence for such cases.

Generally rule based systems are used to determine a reaction to a given state. Here the state is the state of the derivation in combination with the target values. This enables random derivations with an extra layer of intelligence.

Figure 5 shows some such derivations. Here a parallel, topological representation (Duarte, 2005) is maintained throughout the generation. It can be used to overcome some of the problems encountered by the weighted random generation. For example, this rule selection logic can guarantee that
- the doors added to the floor plan connect all the rooms
- the grid has an appropriate size
- the proportions are satisfying
- or that the provisions formulated by Stiny and Gips (1978) are satisfied.

**Extensive Enumeration 3**

**AFTERWORD**

Grammars themselves are impressive devices with enormous generative capabilities based on a finite
and mostly simple set of rules. This generative ability does not come without its toll. The famous sentence “Colourless green ideas sleep furiously” coined by Chomsky nicely illustrates the complexity and ambiguity inherent in such settings. Execution of rules can result in syntactically correct, yet semantically ambiguous sentences. Clearly, the meaning of the sentence is fixed only with respect to a context that validates its usage. The same sentence that is useless in the context of a particular kind of prose may be very useful in some other context when it is used to illustrate humour, intended pans and/or aspects of wordplay. Pairing grammars with rule selection logic is an attempt to address this issue. Rule selection logic fixes the conventions of rule application and clarifies questions pertaining to the selection of rules and the when, where and how these rules may be applied in a design setting. In addition this pairing rule of grammars and rule selection logic helps to show the generative strength that lies in grammars.

The presented agents nicely show how even some simple mechanisms of rule selection can lead to fairly sophisticated results. More sophistication is needed for truly engaging results and they require a fair amount of tailoring and domain knowledge to succeed.

REFERENCES
Correia, R, Duarte, J and Leitao, A 2012 'GRAMATICA: A general 3D shape grammar interpreter targeting the mass customization of housing', Digital Physicality - Proceedings of the 30th eCAADe Conference, Prague, pp. 489-496
Grasl, T and Economou, A 2013b 'Unambiguity', Proceedings of eCAADe 2013, Delft, pp. 617-620