Live Coding Through Rule-Based Modelling of High-Level Structures: exploring output spaces of algorithmic composition systems

Ivan Paz
Computer Sciences Department, Barcelona Tech
ivanpaz@cs.upc.edu; ivnpaz@gmail.com

ABSTRACT

Live coding commonly takes pieces of code from algorithmic composition systems. However, sometimes algorithmic generators either do not consistently show high-level properties, like dramatic transition among parts, or simply, our aesthetic criterion prefers some particular cases among all the possible. In such cases it is useful to have tools for exploring the output space of generative systems, in order to identify and categorize outputs with specific properties. This paper presents an approach to creating linguistic rules out of human-evaluated patterns and their potential uses in live coding to create high-level structures. The methodology starts with a set of sampled examples from an algorithmic system that are evaluated by the user through qualitative linguistic variables. Then, the examples along with the user's evaluation are analysed through an inductive and rule extraction methodology. For a particular example case, these rules are extracted and evaluated. Its application then as information used for writing code on the fly, as well as its implementation in the form of filters or generators is presented.

1. INTRODUCTION

Live coding is closely related with algorithmic composition (Collins 2003a). Many times structures are taken from one domain to the other to supply new material. However, it is common that the outputs of the algorithmic systems either do not consistently show high-level properties, like real dramatic transition among parts (Collins 2009), or simply, our aesthetic criterion prefers some particular cases, among all the possible, for specific parts. In addition, some systems have a large amount of input parameters (see for example Povel, 2010) that make them difficult to set or control, specially in the context of live performance. Also, the output space of the systems is normally huge and difficult to explore by hand. Furthermore, in manual explorations, it is usually hard to establish a relationship between the variables or parameters of the system and the desired outputs. To overcome this problem within a live coding context, instead of starting from a blind exploration, it is helpful to have tools to explore the sample space of the algorithmic systems, in order to identify and categorize the outputs that we want to use. An interesting example, is presented in Dahlsted (2001), who proposed a system, based in interactive evolution, that finds preferred spots in the output space of sound objects created by synthesis and pattern generation algorithms. This work undertakes the exploration of the algorithmic system's output spaces by looking for subspaces with high-level structures. By high-level structures we mean musical objects that are derived from the combination of lower level musical objects, in this case from the combination of the input variables. This work presents an approach to creating linguistic rules out of human-evaluated patterns, characterizing those subspaces, and discusses their potential uses in live coding to create high-level structures. The rest of the paper is structured as follows: Section 2 describes the rule extraction methodology through a specific example, and section 3 discusses the implementation of the rules in the live coding context and outline some ideas for possible further work.

2. AN INDUCTIVE APPROACH FOR HIGH-LEVEL FEATURES RULE GENERATION

This section describes the rule extraction process through a specific example. For further details about the rule extraction methodology, the reader is referred to Mugica et al, (2015), Nebot and Mugica (2012), and Castro et al, (2011). The system's design is shown in Figure 1.

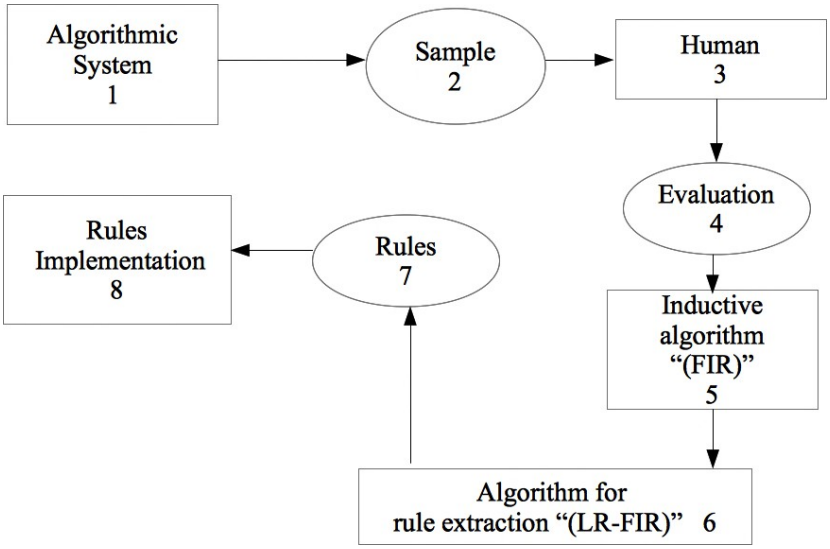

Figure 1. General chart flow of the rule extraction process.

The first module (1) is the algorithmic system. For this example, a simplified algorithm (shown in Table 1) based on (Collins, 2003b) was used. It consists of a set of probability templates for 1 bar, divided in semiquavers for kick, snare and hihat, that produces beat patterns in the style of UK garage. Each number is the probability (normalized up to 1) for an instrument to play at that point in time.

kick	[0.7, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0, 0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3]
snare	[0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.5, 0.0, 0.2, 0.0, 0.0, 1.0, 0.0, 0.0, 0.3]
hihat	[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.7, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.7]

Table 1. Templates for UK garage based on Collins, (2003b).

From these templates a training set of 100 patterns, structured in A + B form where A and B are instances of the algorithm and A ≠ B, was created. The training set was evaluated by the user by means of linguistic variables. In this example, for each pattern in the training set, the aesthetic quality of the structure was labeled as low, medium or high. The data was represented in arrays of length 33. The first 32 values are the semiquavers of the pattern A + B, with a 4 denoting silence, 1 kick, 2 snare, and 3 hihat. The final value at index 33 was the user's evaluation: 1 = low 2 = medium and 3 = high. The examples were evaluated and the data was used as input for the Fuzzy Inductive Reasoning methodology (FIR, module 5). FIR is a qualitative non-parametric, shallow model based on fuzzy logic (Nebot and Mugica 2012). It obtains a set of pattern rules, that describe the qualitative behaviour of the system, by means of searching algorithms, like trees and genetic algorithms. This rule base can be very large and difficult to understand. The Linguistic Rules in FIR (LR-FIR, module 6) contain the algorithm for linguistic rule extraction starting from the pattern rules set generated by FIR, and deriving a reduced and manageable set of linguistic rules. The algorithm performs an iterative process that compacts (or compresses) the pattern relationships obtained by FIR in order to create interpretable rules. This is done by following several steps that include algorithms for rule compaction and optimization processes (Castro et al 2011). An example of the resulting rules is shown in Table 2. Rules are expressed in terms of the values in its variables (V) representing the 33 indexes of the representation discussed above. For example RULE 1 should be read as: If location 16 has a snare or a hihat and location 24 has a kick and location 32 has a kick, then the pattern will be evaluated as high. For each rule "Classified pattern" denotes the amount of patterns (taken in A + B form) that satisfy that rule among all others in the sample space of the algorithm.

RULE	EVALUATION
RULE 1: V16 in 2-3 AND V24 in 1 AND V32 in 1 THEN V33 in 3 Classified patterns 512 **out of** 16384	TP = 16, FP = 4, PRECISION = 0.8
RULE 2: V24 in 1 AND V26 in 2 AND V32 in 1 THEN V33 in 3 Classified patterns 512 **out of** 16384	TP = 16, FP = 4, PRECISION = 0.8
RULE 3: V10 in 4 AND V24 in1 AND V32 in 1 THEN V33 in 3 Classified patterns 512 **out of** 16384	TP = 15, FP = 5, PRECISION = 0.75

Table 2. If-then rules obtained by the system and its evaluation in terms of precision. The three rules describe aesthetic structures user-evaluated as "high" (V33 in 3).

The resulting rules were evaluated using the standard confusion matrix. The Precision (True positive (TP) / True positive + False positive (FP)) was used for the evaluation of each rule. In this case, for each rule a sample of 20 pattern satisfying the rule was created and evaluated. TP patterns were patterns perceived as belonging to the class established by the rule. FP patterns were patterns that satisfy the rule but that are not perceived as belonging to the class of the rule.

3. RULE IMPLEMENTATION FOR LIVE CODING

In this section two approaches for the implementation of these rules in live coding are discussed. Both have been used during live performance with each having their own pros and cons. The standard way of implementing the rules, is to analyse them and use the resulting information as a guide to write our code on the fly. However, as the number of variables in the rules increases, this strategy becomes impractical. In addition, depending on the venue, sometimes we cannot take the risk of have awkward transition moments during the performance. In that cases, the safety of having a pre-programmed structure is preferred. Next, these two strategies are discussed.

As examples of information that can directly be inferred from the rules, let's consider two cases. First, in Table 2 it can be seen that kicks are always found in variables 24 and 32. Then, a safe bet would be to start by setting kicks in those variables. Another insight that can be directly inferred from inspection of the rules is: If we compare the amount of silences (variables in 4) contained in rules describing the high class against silences in rules describing the low and medium classes shown in Mugica et al. (2015), we find only 1 in the high class, while we find 15 and 9 in the medium class and low class rules respectively. From this, we can infer that to produce the desired structure as obtained from the listener's evaluation, it is better to avoid silences, i.e. patterns with higher content of sound information were preferred.

When instead of starting from the white screen we choose to write a patch, or prepare some code from the rules, this can be done in two main forms (without excluding possible combinations): 1. Implement the rules as a filter, to filter out patterns that satisfy rules that do not belong to the chosen class. 2. Implement the rules as a module or class for pattern generation.

3.1 Filter of rules v.s. generative implementation

Implementing the rules as a filter can be thought of as follows. Create a pattern, and if the created pattern does not satisfy any of the rules in the filter, then play the pattern. This implementation produces patterns that show more variety compared with the generative implementation (discussed next). However, we have to be careful, given that the rules do not cover the whole output space. The current methodology produces rules covering around 60% of the space and the precision is approximately 72% (Mugica et al, 2015). Thus, we may be exposed to surprises that have to be managed as we play.

The generative implementation considers the selected rules, and modifies the original algorithm to produce only patterns satisfying one of them. This implementation is perceived as more stable, but at the same time, it produces less variability because of the number of fixed variables. To achieve more variability, implementations can vary ranging from the random selection of the rules to be satisfied, to the use of pondered schemes or probability distributions, assigned according to a

ranking of the preferred rules. We can also listen to a sample of the patterns created by each rule, and select the rules for its application in different sections of the performance by aesthetic criteria.

Finally, a more conservative approach could be to pre-select some structures and launch them in turn. This has also been tried by using the rules of the prior example. Patterns in A + B form were selected and then structured in the form n times (A+B) + n times (A' + B ') + etc., to handguide the development of the piece. In this context the cells found could be used as a raw material to fill sections, guiding the evolution of the piece by hand.

Acknowledgments

My sincere thanks to Diego Garcia-Olano for the careful revision of this text, and to the anonymous reviewers that made suggestions to improve the trend of thought of the document.

REFERENCES

Collins, Nick. 2003a "Generative Music and Laptop Performance" Contemporary Music Review. Vol 22, No. 4, 67-79.

Collins, N. 2003b "Algorithmic Composition Methods for Breakbeat Science" ARiADA No.3 May.

Collins, Nick. 2009 "Musical Form and Algorithmic Composition" Contemporary Music Review Vol. 28, No. 1, February 2009, pp. 103–114.

Dahlstedt, P. 2001 "Creating and exploring huge parameter spaces: interactive evolution as a tool for sound generation". *Proceedings of the International Computer Music Conference,* Habana, Cuba.

Nebot, À. Mugica, F. 2012. "Fuzzy Inductive Reasoning: a consolidated approach to data-driven construction of complex dynamical systems." International Journal of General Systems. Volume: 41(7) Pages. 645-665.

Mugica, M. Paz, I. Nebot, À. Romero, E. 2015. "A Fuzzy Inductive Approach for Rule-Based Modelling of High Level Structures in Algorithmic Composition Systems". In proceedings of IEEE international conference on fuzzy systems.

Povel, D. 2010 "Melody Generator: A Device for Algorithmic Music Construction" J. Software Engineering & Applications, 2010, 3, 683-695.