Recursive Generation of Rhythmic Structures with PTGGs

Brian Heim
Affiliation1
author1@unt.edu

Donya Quick
Affiliation2
author2@unt.edu

ABSTRACT

[Will need to be tweaked at the end to reflect any additions, but the original intro seems like a good fit here at the moment]

We present an approach for generating passages of rhythm via the use of a probabilistic temporal graph grammar with rule sets that operate via recursive subdivision. This project is motivated by three factors: (1) the recognition that the literature on computer-generated rhythm is generally not as robust as those for melody and harmony; (2) my desire as a composer of avant-garde classical music to have more tools for computer-assisted composition; and (3) the hope that such an approach, having already produced compelling results for some styles of music, may provide insight into the structure of rhythms found in existing repertoires.

1. INTRODUCTION

There has been extensive work in the areas of analyzing and generating pitch-related features in music such as melody and harmony. However, rhythmic structure has received substantially less attention, particularly in the area of automated composition. Typically generated rhythms a either very simple, such as those found in Bach chorales, or they struggle to retain a sense of meter.

We present a grammatical approach to generating diverse rhythmic patterns in multiple time signatures while preserving hypermetrical structures. Our approach also allows the generation of repeated patterns at multiple levels of granularity, an extremely important feature of rhythm in music. Finally, the probabilistic grammars described here also have the potential for rule probabilities to be derived from real musical examples.

1.1 Related Work

One of the most commonly applied algorithms in computer music analysis and generation is the Markov chain [1, 2, 3, 4]. However, Markov chains are fundamentally limited in the temporal scope of features that they can capture, which is problematic for modeling musical features like metrical structure and rhythmic patterns—for which temporal context and repetition are extremely important. As the order of the chain increases, more complex features can be modeled, but observations of states become sparse and the number of states increases exponentially. Even approaches such as variable-length Markov chains [5, 6] and the extensions noted above do not entirely overcome these problems.

Some extensions to Markov chains exist that can overcome some of the problems of preserving metrical structure observed with more traditional approaches. Roy and Pachet model meter in melodic generation [7] using and extension known as Markov constraints [8] to achieve properties such as properly filled measures. This method involves enumeration of possible outcomes for metrical constraints, although the complexity is mitigated to some extent by a filtering step to avoid a truly brute force search.

Work such as that of Lerdahl and Jackendoff [9], as well as the work of Temperley [**?**], presents a statistical view of metrical structure, where metrical positions are assigned a probability distribution and downbeats are weighted more heavily. While these models are useful analytically and can be used to generate new rhythmic patterns, as with Markov chains, there is no support for modeling repetition, and relatively common, syncopated rhythmic patterns are unlikely due to having little downbeat content.

Keller and Morrison present a grammatical model for jazz solos that includes a rhythmic component. This grammar is context-free and generates rhythmic structure independently of melodic structure. The rhythmic portion of the grammar is fundamentally based on dividing times in half. Although some degree of syncopation is supported, the strategy of halving durations through much of the generation fundamentally limits it to duple meters. The grammars we introduce in later sections of this paper use a tiered, multi-rule-set approach that supports many other time signatures.

TODO: *Maybe cite other things if there is space*

1.2 Probabilistic Temporal Graph Grammars

A grammar is defined as a 4-tuple: $G = (N, T, R, S)$ where the alphabet $N \cup T$ comprises nonterminals, N, and terminals, T. R is a set of rules of the form $x \in N \to (N \cup T)^*$ and $S \in N$ is the starting symbol. Terminals are symbols that only produce themselves, i.e. cannot be further altered, while nonterminals may produce any combination of terminals and nonterminals. The generative algorithm is similar to that of L-Systems: at each generative iteration of the grammar, starting with the single-symbol sequence of $X = S$, the appropriate rule $(x \to y) \in R$ is applied to each nonterminal symbol from left to right to produce a new sequence X', which may contain more symbols than X. This process of iteration is called expansion, because the sequence length is a nondecreasing function of the number of iterations. The condition that produc-

tion rules operate on individual symbols without knowledge of the surrounding symbols means that this definition also describes a Context-Free Grammar (CFG). Probabilistic context-free grammars (PCFGs) have more than one rule with the same left-hand side. Rules sharing the same left-hand side in a PCFG must have associated probabilities that sum to 1.0.

Kulitta features a category of grammars called Probabilistic Temporal Graph Grammars[10, 11], or PTGGs. PTGGs are essentially parameterized PCFG, where production rules are functions on parameters. Allowable functions on symbol parameters include simple conditional logic and let-in statements for variable creation and instantiation. These let-in statements allow the generation of repetitions, which are an important feature of rhythm in music.

Rules in PTGGs have the following format:

$$\begin{aligned} x^p \rightarrow \quad & f_1(p) \\ & \mid \textbf{if } cond(p) \textbf{ then } f_1(p) \textbf{ else } f_2(p) \\ & \mid \textbf{let } x = f_1(p) \textbf{ in } \{x \mid f_i(p)\}^+ \end{aligned}$$

where each $f_i(p)$ indicates some function from parameter values to sequences of terminals and nonterminals and $cond(p)$ is a conditional test on the left-hand-side symbol's parameter (such as $p < 2$).

2. A PTGG FOR RHYTHM

In most PTGGs used by Kulitta, the alphabet consists of Roman numeral chord symbols parameterized by duration, and rules are *duration preserving*, meaning that the right-hand side symbols divide up the duration of the left-hand symbol, typically by a power of two. This division always results in a string that has equal duration to the original symbol. A rule such as $I^t \rightarrow I^{t/2}V^{t/4}I^{t/4}$ indicates that a I-chord may be replaced by a I-V-I progression where the chord durations are $\frac{1}{2}$, $\frac{1}{4}$, and $\frac{1}{4}$ the durations of the original chord. This implies a general approach of beginning with a long duration and iteratively dividing it into progressively smaller durations; since one would typically like to have control over the exact duration of a generated composition, this is a reasonable approach.

The grammar proposed here is based on the observation that many theories of Western counterpoint, including those of Fux, Schoenberg, and Schenker [**?**], treat rhythm as a hierarchy of nested structures, discuss metrical patterns such as anapest (short-short-long) in the same light regardless of the level of duration on which they occur, and regard small note values as elaborations or ornamentations of a less complex metrical pattern.

We also introduce the notion of *production phases* into the PTGG framework to account for the need for different rules to take place at different granularities in the generative process.

Durations are treated as both terminals and nonterminals of an infinite parametrized alphabet of duration types in a four-phase generation algorithm process, each step of which subdivides an initial duration into smaller and smaller units.

Throughout the following sections, we use 1.0 to represent the duration of a whole note.

2.1 Nonterminals

The nonterminals of the grammar are $N = \{Beat, Dotted, QuarterDotted, Measures\} \cup TS$ where TS is a set of *time signature symbols*.

The *Beat* symbol represents a note with duration $2^n/m$ where n and m are integers with $n \geq 0$, $m > 0$. The *Dotted* symbol represents a duration analogous to a dotted note; it lasts 1.5 times its undotted value. Similarly, a *QuarterDotted* lasts for 1.25 times its undotted value. Although the notion of a "quarter dotted" note has no counterpart in common practice notation (although some composers such as George Crumb have invented their own symbols for it), a duration of $\frac{5}{8}$ is found frequently in situations where, for example, a quarter note is tied to the first of a series of sixteenth notes. Because the grammar has no way to represent ties, this symbol is included for robustness.

One could argue that, for the sake of completion, symbols for $\frac{7}{8}$, $\frac{9}{8}$, and so on ought also to be defined. We have chosen to forego defining these symbols because (1) their durations occur less frequently than $\frac{3}{8}$ and $\frac{5}{8}$, (2) we are still able to express a wide variety of rhythms without them, (3) there are other, more elegant routes to the same solution (see Future Work).

Measures represents a nonzero length of consecutive measures; each measure has nonzero duration but has not yet been assigned a time signature. TS is the set of time signatures that may appear in the final output; symbols in TS are named for the meters they represent: *FourFour*, *ThreeFour*, *SixEight*, and so on.

2.2 Terminals

The sole terminal is $T = Short$.

Short stands for a *Beat* which has been subdivided to the point where its duration is less than or equal to $1/2^n$, where n is an arbitrary integer constant. Intuitively, a very short note is prevented from being further subdivided. This behavior is designed to mimic the limits of human performers? rhythmic articulation.

The notations B^d, S^d, D^d, Q^d will be used hereafter for $Beat$, $Short$, $Dotted$, and $QuarterDotted$ symbols of duration d respectively. Since a $Measures$ symbol does not have duration, but rather length, we use the notation M^l for *Measures* of length l.

2.3 Production Phases

Production occurs in four phases of rulesets. A sentence is not considered well-formed until passing through all four phases. Practically, these phases correspond to the generation of (1) phrase structure, (2) time signatures, (3) intra-measure beat patterns, and (4) recursive rhythmic subdivision. Having four separate rulesets, each with a well-defined purpose, allows us a great deal of flexibility without forcing us to overspecify.

Phase 1 begins with the string $\alpha = M^l$ where $l = 2^a, a \geq 0$, and halts with the string $\beta = \{M^1\}^*, |\beta| = l$.

Phase 2 generates γ from β by replacing each M^1 in β with a symbol $ts \in TS$, where TS is a nonempty set of time signature symbols.

Phase 3 generates δ from γ by replacing each ts in γ with a sequence of symbols from $\{Beat, Dotted, QuarterDotted, Short\}$ such that each sequence's duration sums to the

duration of the time signature it replaces. For instance, the sequence $[B^{\frac{1}{2}}D^{\frac{3}{8}}B^{\frac{1}{8}}]$ may validly be generated from *FourFour* because $\frac{1}{2}+\frac{3}{8}+\frac{1}{8}=1$; the sequence $[B^{\frac{1}{2}}D^{\frac{3}{4}}]$, however, may not be generated from *FourFour*.

Phase 4 generates the output string by repeatedly subdividing δ. Rules in this phase operate within the duration-parametrized set $\{Beat, Dotted, QuarterDotted, Short\}$. In order to preserve the duration of δ, rules are constrained such that the durations of the right-hand side of a rule must sum to the duration of the left-hand side.

2.4 Production Rules

Production rules in Phase 1 are functions from measures to measure groups, $M \rightarrow K$, and in Phase 2 from beats to beat groups, $B \rightarrow L$. The strict definition of PFCG says that a single rule must produce a single string of symbols, as in the case of $B \rightarrow BB$ and $B \rightarrow SS$. Yet *Short*, as noted above, in only produced when its parameter meets certain conditions. Since the decision to output B, D or S can be made solely on the basis of the parameter value, we define rules as functions:

$B^a \rightarrow \mathbb{X} \in \mathbb{Q}^p$ where

$p > 0, \forall x \in X : 0 < x \leq 1, \sum_{x \in X} x = 1$

and then use the resulting parameters $[d_1, d_2, ?, d_p] = aX$ to label the produced symbols. This allows us to define patterns of subdivision clearly and succinctly. For example, let two Phase 4 rules be $B \rightarrow [\frac{3}{4}, \frac{1}{4}]$ and $B \rightarrow [\frac{1}{2}, \frac{1}{4}, \frac{1}{4}]$. Here are two iterations of production using these rules and a *Short* threshold of $\frac{1}{16}$:

Rule	String
(start)	B^1
$B \rightarrow [\frac{3}{4}, \frac{1}{4}]$	$D^{\frac{3}{4}}B^{\frac{1}{4}}$
$B \rightarrow [\frac{1}{2}, \frac{1}{4}, \frac{1}{4}]$	$D^{\frac{3}{4}}B^{\frac{1}{8}}S^{\frac{1}{16}}S^{\frac{1}{16}}$

3. IMPLEMENTATION

We use Haskell-based PTGG generation framework from Kulitta and the MIDI export capabilities offered by the Euterpea library [12] to allow for elegant rule definition and parsing. Below are some of the key features of this implementation.

3.1 Four-Phase Generation Algorithm

Assuming the rule sets have already been defined, the complete generation algorithm requires four parameters: (1) the total length $l = 2^n$, $n > 0$, of the passage in measures; (2) the shortest allowable duration $1/2^n$ expressed as n; (3) the number of Phase 4 iterations to perform; and (4) a random seed.

As stated above, generation begins with the string $\alpha = M^l$. Phase 1, phrase structure generation, uses only two rules: $M^l \rightarrow M^{\frac{l}{2}}M^{\frac{l}{2}}$ and **let**$x = M^{\frac{d}{2}}$**in**xx. Both rules subdivide the group of measures into two equal halves, but one does so through variable instantiation. Since l is a power of two, Phase 1 halts in $log_2 m$ iterations.

In Phase 2, time signature assignment, each measure is assigned a time signature from the set TS. Since most pieces of common practice music are in a single time signature, this step is trivial and will only consist of one rule $M \rightarrow ts$.

In Phase 3, beat pattern assignment, each time signature symbol is replaced with a sequence of beats that sum to the duration of that meter. An example Phase 3 rule might be $FourFour \rightarrow B^{\frac{1}{2}}B^{\frac{1}{2}}$, which divides a 4/4 measure into two independent halves. This Phase may also be trivial for generating common practice passages, but for complex or irregular time signatures such as 7/8, explicitly defining the abstract beat pattern ? as $2+2+3$, or $3+2+2$, or even a mix of both ? in a separate phase of generation is helpful.

4. EXAMPLES

We now present three example passages generated from this grammar. First, we present an example in 4/4 demonstrating major features of the grammar implementation. Secondly, we show that the grammar easily extends to other time signatures with an example in 3/4. Finally, we demonstrate unusual but idiomatic possibilities with an example in mixed irregular meters.

TODO: *FINISH SECTION*

5. LEARNING PRODUCTION PROBABILITIES

An algorithm for deriving production probabilities from a training corpus has been described for probabilistic temporal graph grammars [**?**], and is part of the Kulitta framework. This learning method is based on the inside-outside algorithm[**?**], which, in turn, is a tree generalization of the more commonly used forward-backward algorithm for hidden Markov models.

Kulitta's production probability learning-strategy is oracle-based, allowing PTGG for which the oracle can be defined concretely to have production probabilities estimated from real data. This strategy also supports PTGGs with rules that feature let-in expressions and conditional statements. The oracle must be able to identify candidate parent symbols that may have produced mixed sequences of terminals and nonterminals.

For the PTGGs described in this paper, the learning oracle is simple to define because all of the rules are *duration preserving*. A rule can be tested against a sequence of symbols by normalizing the durations of the input sequence to sum to 1.0 and comparing against symbol parameters in the rule. For example, the sequence B^2B^2, which normalizes to $B \rightarrow B^{\frac{1}{2}}B^{\frac{1}{2}}$ can clearly be produced by the rule $B \rightarrow B^{\frac{1}{2}}B^{\frac{1}{2}}$, but B^1B^2 cannot.

As a proof of concept, we tested a reduced rule set using a small collection of rhythmic patterns from Bach cantatas. The production probabilities derived from this experiment are shown in Table 1, and example rhythms generated from the learned model are shown in Figure 1.

6. CONCLUSION

I have presented an application of PTGGs to the topic of automated composition along the sole parameter of rhythm. I detailed one implementation of such a grammar and examined a sample of its output. The grammar shown above

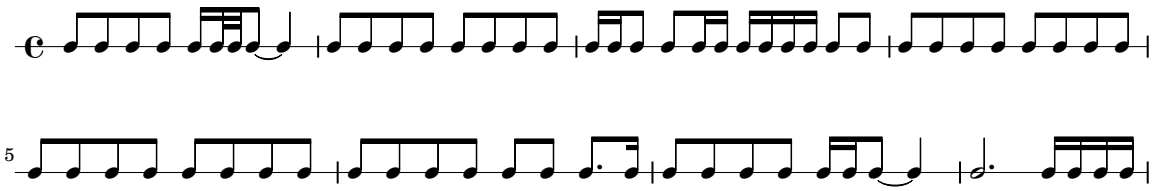

Figure 1. Examples of rhythms generated from the grammar in Table 1, the probabilities for which were derived from a small number of Bach cantata excerpts.

0.851	$B \rightarrow B^{\frac{1}{2}} B^{\frac{1}{2}}$
0.004	$B \rightarrow B^{\frac{1}{2}} B^{\frac{1}{4}} B^{\frac{1}{4}}$
0.030	$B \rightarrow B^{\frac{1}{4}} B^{\frac{1}{2}} B^{\frac{1}{4}}$
0.002	$B \rightarrow B^{\frac{1}{4}} B^{\frac{1}{4}} B^{\frac{1}{2}}$
0.071	$B \rightarrow D^{\frac{3}{4}} B^{\frac{1}{4}}$
0.010	$B \rightarrow B^{\frac{1}{4}} D^{\frac{3}{4}}$
0.010	$B \rightarrow B^{\frac{1}{4}} B^{\frac{1}{4}} B^{\frac{1}{4}} B^{\frac{1}{4}}$
0.010	$B \rightarrow B^{\frac{1}{3}} B^{\frac{1}{3}} B^{\frac{1}{3}}$

Table 1. Rule set with probabilities derived from training on examples of rhythms from Bach cantatas.

is a preliminary attempt, and there are some possible routes for its extension.

TODO: *need more conclusion*

6.1 Future Work

TODO: *rewrite*

One clear choice would be to try introducing rhythmic concepts that this grammar cannot currently express. These include ties, rests, and triple (3/4) meter. The former two items suggest the introduction of more alphabet symbols: a Tied symbol, produced from a Beat or Short, could be interpreted as tied over from the previous note during playback. A Rest symbol could be similarly used to eliminate some articulated notes from the interpreted string. Adding compatibility for triple meter would imply reworking Dotted as a nonterminal symbol.

Another concept that seems to be missing from the current alphabet is that of an embellishment or ornamentation. Such symbols could add short note values before or after more metrically significant beats, and in so doing introduce a third level of metrical hierarchy. A separate way to approach this would be by allowing existing let-in structures in the string to be resolved during or between iterations, rather than at the end of the entire cycle. This would preserve high-level symmetry while allowing for small-scale variations.

With certain adaptations, the proposed grammar could generate rhythms in more contemporary styles: for instance, the composition of mixed-meter passages would require a reworking of the first, Measure-based production phase, and might be better generated additively as opposed to recursively. The grammar already supports the complex and nested tuplets which are often found in classical music of the 20th and 21st centuries, and by slackening the imposed constraint on constant total string duration, it might also be able to generate passages in free meter or ?non-geometric? time.

Another direction to explore is context sensitivity. In some cases, it might be helpful to know certain contextual details about a symbol such as: (1) its position within the measure, which typically determines its emphasis (downbeat vs. upbeat), (2) its position within the overall duration of the passage, and (3) whether the passage is increasing or decreasing in rhythmic activity. Otherwise, it is not possible to account for what Schoenberg termed ?the tendency of the smallest notes? [10]?a contrapuntal phenomenon wherein shorter durations are used more frequently toward the end of a passage or phrase?without adding other constraints to the rule set.

Finally, these variations and improvements could also be useful for interfacing my grammar with other automated composition algorithms that produce melodies, harmonies, and counterpoint, either by using its output as a blueprint or by including it in a ?composition by committee? approach. In that case, it would be important to embed as much contextual and functional knowledge as possible into the alphabet and the symbol parameters so that other grammars and algorithms can make informed decisions. For example, the knowledge that a downbeat is tied over or that two halves of a phrase have some rhythmic symmetry could be interpreted by a harmonic composition algorithm to produce more contextually aware chord sequences.

7. REFERENCES

[1] P. Chordia, A. Sastry, and S. Senturk, "Predictive Tabla Modeling using Variable-length Markov and Hidden Markov Models," *Journal of New Music Research*, vol. 40, no. 2, pp. 105–118, 2011.

[2] K. T. Jon Gillick and R. M. Keller, "Learning Jazz Grammars," in *Proceedings of the Sound and Music Computing Conference*, 2009, pp. 125–130.

[3] L. Yi and J. Goldsmith, "Automatic Generation of Four-Part Harmony," in *UAI Applications Workshop*, 2007.

[4] F. Pachet, "The Continuator: Musical Interaction With Style," in *Proceedings of the International Computer Music Conference*, 2002, pp. 2011–218.

[5] D. Ron, Y. Singer, and S. Tishby, "The Power of Amnesia: learning Probabilistic Automata with Variable Memory Length," *Machine Learning*, vol. 25, pp. 117–149, 1996.

[6] P. Bühlmann and A. J. Wyner, "Varable Length Markov Chains," *The Annals of Statistics*, vol. 27, no. 2, pp. 480–513, 1999.

[7] P. Roy and F. Pachet, “Enforcing Meter in Finite-Length Markov Sequences,” 2013.

[8] F. Pachet and P. Roy, “Markov constraints: steerable generation of Markov sequences,” *Constraints*, vol. 16, no. 2, pp. 148–172, 2011.

[9] F. Lerdahl and R. S. Jackendoff, *A Generative Theory of Tonal Music*. The MIT Press, 1996.

[10] D. Quick and P. Hudak, “Grammar-Based Automated Music Composition in Haskell,” in *Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling, and design*, 2013, pp. 59–70.

[11] D. Quick, “Kulitta: a Framework for Automated Music Composition,” 2014.

[12] P. Hudak *et al.* (2014) Euterpea. [Online]. Available: http://hackage.haskell.org/package/Euterpea