A TEMPORAL GENERATIVE GRAPH GRAMMAR FOR HARMONIC AND METRICAL STRUCTURE

Donya Quick, Paul Hudak

Yale University
Department of Computer Science
New Haven, CT USA

donya.quick@yale.edu, paul.hudak@yale.edu

ABSTRACT

Most grammars that have been proposed for automated music composition fall into conventional linguistic categories, such as context-free, context-sensitive, and probabilistic versions of each. For parsing (i.e. musical analysis) these distinctions are important, because of the computational complexity of parsing using these different grammars. But, for generation (i.e. derivation), the complexity issues are sometimes different, and other goals for having a generative grammar come into play.

In this paper we describe a new category of grammar for generating abstract harmonic and metrical structure. This class of grammars has two distinctive features. First, it is *temporal*, meaning that production rules are parameterized by the duration of phrases, thus allowing us to express the metrical structure of a composition. Second, it is a *graph grammar*, meaning that the parse trees (or derivation trees) are actually *graphs* allowing shared nodes, thus enabling us to express the sharing, i.e. repetition, of specific musical phrases. We formally define this class of grammars, describe our generative implementation of it, and present a realistic example of its use: a specific grammar tailored to some styles of classical Western music. In addition, we show that this class of grammars integrates nicely with the notion of *chord spaces* to generate concrete chords from the abstract harmonic structure generated from the grammar.

1. INTRODUCTION

Automated composition is a complex task. Music is multidimensional, and the solution spaces are very large. Even a seemingly simple task such as choosing pitches for chords results in an exponential growth in computational complexity. Having elegant and efficient algorithms is therefore important for traversing these solution spaces. As a further complication, those large solution spaces contain many poor or undesirable solutions. So, not only does a generative algorithm for music have to be efficient, it has to also find satisfactory solutions in a sea of noise.

For example, probabilistic context-free grammars (PCFGs) are both efficient to use generatively and, if constructed appropriately, are capable of producing satisfactory solutions. But they also generate many unsatisfactory solutions. We could do better if at least some notions of "satisfactory" could be captured in the grammar itself, thus eliminating large undesirable regions of the solution space.

To address this problem, we present a new category of grammar that we refer to as a *temporal generative graph grammar* (TGGG). TGGGs are powerful enough to express both temporal and sharing constraints, leading to an effective method for the automated generation of harmonic and metrical structures of music.

There are two distinctive features of a TGGG. First, one often wishes to preserve certain metrical constraints in a composition. For example, one may wish to replace a *I* chord with a *II V I* progression, but with the constraint that the total duration of the *II V I* progression is the same as that of the original *I* chord. In our TGGG framework, this is easily specified with the rule $I^t \rightarrow II^{t/4}\ V^{t/4}\ I^{t/2}$. This allows metrical constraints such as those found in *Generative Theory of Tonal Music* (GTTM) [14] to be captured by the grammar itself in a more formal way.

Technically, this feature results in an infinite number of production rules, but, when used in a generative setting, that is not problematic. The initial duration t_0 associated with a given grammar's start symbol directly determines all the others. One can think of each rule as being a function that takes a time (duration) as an argument.

The second distinctive feature of a TGGG is that it can capture the notion of *repetition* of a phrase—i.e., the sharing of a particular sentential form in a derivation. For example, if part of a composition has the form *ABA*, and we expect both occurrences of *A* to be precisely the same phrase, we can write the following in a TGGG (where we have omitted any temporal superscripts for simplicity):

$$S \rightarrow \textbf{let}\ x = A\ \textbf{in}\ xBx \qquad (1)$$

where S, A, and B are nonterminals. Note that "*A*" in this case means that some arbitrary number of production rules are applied to the non-terminal *A*, and that the result is used identically in the two occurrences of *x* (versus simply writing *ABA*, where each *A* could expand differently). Therefore, a derivation tree is actually a more general type of directed graph where branches can share nodes. These shared nodes occur whenever there is a **let** expression.

Technically, this means that a TGGG is a context-sensitive grammar, which raises the computational complexity of parsing. However, this is not a problem in a generative setting, because the shared nodes also represent shared computation, and thus the computational complexity is actually reduced.

In the remainder of the paper we formalize these two concepts, define an algorithm for generating musical phrases with a set of probabilistic TGGG rules, and describe an implementation of the overall framework. Our implementation includes a seamless integration with *chord spaces* [22, 5, 16] to generate concrete chord progressions from an abstract harmonic structure. Finally, we demonstrate our system using a specific grammar that captures simple classical Western harmonic structure. Our results show promise for development of future algorithmic composition systems based on TGGGs and chord spaces.

2. GRAMMAR DEFINITION

A grammar is a tuple, $G = (N, T, R, S)$ where N is a set of nonterminals, T is a set of terminals, R is a set of rules from $N \rightarrow N \cup T$, and $S \in N$ is the start symbol. Terminals are symbols that can only produce themselves, whereas nonterminals have rules that replace them with one or more other symbols.

Our TGGG is based on the Schenkerian idea that harmony in music is hierarchical in nature, that basic metrical features must be preserved in that hierarchical structure, and that short progressions may be considered elaborations of a single, longer chord. If a $II\ \ V\ \ I$ progression were analyzed as representing a tonic section, it might be replaced with a I-chord at some stage of analysis. This process would be reversed in a generative setting.

Chords in our grammar are treated as both nonterminals and terminals. Production rules replace single chords with longer progressions while observing temporal constraints on those chords and progressions. The grammar integrates metrical and harmonic considerations, handling generation of both features at once (as opposed to a two-step process such as Keller's grammar for jazz riffs [13]). Our grammar operates on an infinite alphabet of duration-parameterized chords, and rules are functions on those chords (although there may be more than one rule for a particular chord).

2.1. Nonterminals

The basic nonterminals of our grammar consist of abstract chords with durations. Classical Western harmony is usually analyzed using the Roman numeral system, where chords are labeled according to the root's scale index. We define the set C of such Roman-numeral-style chord labels.

$$C = \{I,\ II,\ III,\ IV,\ V,\ VI,\ VII\} \tag{2}$$

The quality of a chord is determined by its interpretation within a key. Each chord will also occupy some duration of time. We will use the notation c^t to indicate that chord c has duration t. Chord progressions are written as a sequence of chord symbols and are interpreted from left to right as on a musical score.

The nonterminals in our grammar, N, are the infinite set of all possible chords with durations. According to Schenkerian theory, pieces of music are fundamentally reducible to a I-V-I, or sometimes just I [20, 18, 14]. In keeping with this model of harmony, we define the start symbol, S, as a I-chord with the duration of the entire passage to be generated: I^t.

2.2. Terminals

Terminals of our grammar consist of all nonterminals and some additional modulation symbols. Modulations, or key changes in our grammar take place according to scale index. For simplicity and appropriateness within traditional classical harmonies, we only consider modulations to degrees of the major and minor scale rather than chromatic modulations. There are 7 scale indices, but only 6 of them produce a key change, and there is no need to indicate a modulation from a key to itself. We therefore use 6 symbols, denoting modulations from the second to seventh scale degrees.

$$M = \{M_2, M_3, M_4, M_5, M_6, M_7\} \tag{3}$$

M_2 is a modulation to the 2^{nd} degree of the scale (the key of the II-chord), M_3 is to the 3^{rd} degree, and so on. Modulated sections of progressions of chords are indicated using parentheses. The parentheses serve as a "meta-symbol" to indicate parse tree productions that fall under a modulation. Chords appearing within the parentheses are affected by the modulation, and those appearing outside are left unchanged. The mode of the modulation is determined by the context in which it occurs. For example, consider the progression $(M_5\ \ V^{t_1}\ I^{t_2})\ II^{t_3}\ V^{t_4}\ I^{t_5}$. If the entire progression is interpreted in the key of C-major, the first two chords, $V^{t_1}\ I^{t_2}$, would be in G-major, yielding D-major and G-major chords respectively. The next three chords, $II^{t_3}\ V^{t_4}\ I^{t_5}$, would be interpreted in C-major, giving D-minor, G-major, and C-major chords.

The terminals T, of our grammar include both the chord symbols and modulations described so far: $N \cup M$.

2.3. Sentential Forms

The set K of *sentential forms* of a TGGG (ignoring the temporal superscripts) is defined as:

$$k \in K\ ::=\ c \mid k...k \mid (m\ k) \mid \textbf{let}\ x = k\ \textbf{in}\ k \tag{4}$$

for $c \in C$, $m \in M$, and $x \in V$, where V is a predefined set of variables.

Consider, for example, the form *AABA*, where all *A* sections must be identical and the *B* section is different and also modulated to the dominant. Under Schenkerian theory, the entire *AABA* passage could be thought of as representative of a *I* chord of the total duration occupied by *AABA*. If all sections have the same length, then this structure could be written as a production of the start symbol, I^t:

$$\textbf{let } x = I^{t/4} \textbf{ in } x\, x\, (M_5\, I^{t/4})\, x \qquad (5)$$

2.4. Production Rules

Production rules are functions from chords to chord progressions, $N \to K$. Our grammar is based on the idea that short progressions may be musically representative of a single chord of the same duration. Because N and T are infinite sets, a set of rules on those alphabets could also be infinite. However, we can instead write rules over categories of symbols at the Roman-numeral level. The temporal aspect of the nonterminals is parameterized, and our rules are functions of that parameter. This aspect of the grammar means that meter and larger temporal constraints can be factored into the grammar. For example, consider the following two rules: $I^t \to II^{t/4}\ V^{t/4}\ I^{t/2}$ and $I^t \to V^{t/2}\ I^{t/2}$.

Let w, h, q, e be durations representing a whole note, half note, quarter note, and eighth note respectively. The following is an example production using the two-rule templates above:

String	Rule used
I^w	Start symbol
$V^h\ I^h$	$I^t \to V^{t/2}\ I^{t/2}$
$V^h\ II^e\ V^e\ I^q$	$I^t \to II^{t/4}\ V^{t/4}\ I^{t/2}$

3. IMPLEMENTATION

We have implemented a system to generatively interpret a probabilistic TGGG. The implementation is written in Haskell [15], which provides an elegant way to define production rules, especially when treated as functions parameterized with duration. [1] This section describes the most salient aspects of the implementation.

3.1. Generative Algorithm and Replacement Function

Because all nonterminals are also terminals, strings produced by our grammar can be interpreted musically after any number of generative steps. However, a simple algorithm is still required to ensure a musically acceptable result. For example, consider the following extreme example: what if a production only takes place on the very last terminal in the string, which must be a *I* chord? Aside from the harmonically repetitive nature of the result, the durations would be very long on the left relative to those occurring towards the right. Fortunately, in a truly random setting, this type of completely skewed result would be rare, but so would a more uniform distribution of durations. For a chorale, we might want to see mostly quarter notes, eighth notes, and the occasional half note or whole note, but certainly no chords spanning 4 or more measures in the middle of a piece. How can such a distribution be realized from a probabilistic grammar?

[1] Our implementation is available at `haskell.cs.yale.edu`

Let R be the set of production rules, each of which has an associated probability. A function $choose(c^t, R)$ picks a rule from R that can be applied to c^t according to the probability mass distribution of the rules. The notation $r(c^t)$ indicates applying a rule $r \in R$ to the chord c^t and the notation $[x_1, ..., x_n]$ denotes a sequence of symbols. The function *concat* combines multiple sequences into a single sequence.

Algorithm 1. $replace(t_{min} : \mathbb{R},\ c^t,\ R) =$
Let $r = choose(c^t, R)$*,* $str_c = r(c^t)$*. If* $\exists c'^{t'} \in str_c, t' < t_{min}$*, then return* c_t*. Otherwise, return* str_c*.*

Algorithm 2. $replaceAll(t_{min} : \mathbb{R},\ [k_1, ..., k_n] : K,\ R) =$
Let $k'_i =$ *if* $k_i \in N$ *then* $replace(t_{min}, k_i, R)$*, otherwise* k_i*.*
Return $concat(k'_1, ..., k'_n)$.

Algorithm 3. $gen(str : K,\ i : \mathbb{Z},\ t_{min} : \mathbb{R},\ R) =$
If $i \le 0$ *then return* str*.*
Otherwise, return $gen(replaceAll(t_{min}, str),\ i-1, t_{min}, R)$*.*

Statements of the form **let** $x = A$ **in** exp require that A and exp be expanded separately before replacing instances of x in exp. Once *gen* has been called on all progressions of type K, variables can be replaced by their values to create a single sequence of chords.

3.2. From Numerals to Pitches

The output of our grammar is a sequence of Roman numeral and modulation symbols. This is not yet anything that could be called music, since each string represents many possible musical interpretations that share the same abstract, harmonic structure. In 2012, we introduced a framework for moving between levels of abstraction in music [16]. This framework presents an easy way to turn a series of abstract chords like Roman numerals into concrete chords while satisfying certain constraints.

An *abstract* chord is one that requires additional decisions in order to be used for a particular task. A *concrete* chord is one with sufficient information for that task. Concepts like "I-chord" and "C-major chord" are abstract because they correspond to many concrete chords. We represent concrete chords as vectors of integers, where each integer represents a pitch. (C,0) is taken as 0, so (C,5)=60, (D,5)=62, and so on. Each index in a chord's vector corresponds to a particular voice. This representation is sometimes called a voicing.

3.3. Moving Between Levels of Abstraction

A *chord space* is a way of grouping chords in musically useful ways by using equivalence relations (relations that are reflexive, symmetric, and transitive). The work in [16] adds additional formalization for three important equivalence relations defined by Tymoczko et al. [22] and Callender et al. [5]: octave equivalence (O), permutation equivalence (P), and transposition equivalence (T). Chords are O-equivalent if they share the same vectors of pitch classes, P-equivalent chords share the same multisets of pitches, and T-equivalent chords have the same intervallic structure of pitches. These concepts can also be combined. For example, OP-equivalent chords share the same sets of pitch classes, and OPT-equivalent chords have the same intervallic structure of pitch classes—a level of abstraction that captures chord quality.

Chord spaces can be used to move between different levels of abstraction in music [16]. An equivalence class of chords under any of the O, P, and T relations represents a particular musical concept. For example, the OP-equivalence class to which $\langle 0,4,7 \rangle$ belongs includes all C-major triads.

A chord space is a set of equivalence classes. We can choose a single *representative point* from each equivalence class to form a *representative subset* of the space. The representative subset of a chord space can be thought of as a different level of abstraction, where each point represents many other points in the chord space.

A sequence of representative points from a chord space represents a sequence of equivalence classes. Such a path also represents many possible other paths through non-representative points in those equivalence classes. Given a chord space that has the same level of abstraction as an abstract progression (such as one written as Roman numerals), the task of turning that abstract progression into a concrete progression becomes a path-finding problem.

We use a simple, stochastic method defined in [16] for turning abstract progressions into concrete ones that utilizes chord spaces. Given a chord space, an abstract progression, and some concrete constraints, the algorithm picks concrete chords greedily from the equivalence classes specified by the abstract progression from left to right without backtracking. The constraints used for our examples here are easy to satisfy, so the algorithm is unlikely to become stuck.

3.4. The Right Level of Abstraction

OP-space is useful for turning our strings of Roman numerals into concrete chords with three voices while respecting various voice-leading constraints. However, while both $\langle 48,60,64,67 \rangle$ and $\langle 55,60,64,67 \rangle$ represent a I-chord in the key of C-major, they would belong to different OP-equivalence classes. Therefore, OP is not general enough to support higher numbers of voices for the same application. On the other hand, OPT-equivalence is *too* general, because a string of Roman numerals in a particular key could be mapped to any sequence of chords that shared the same pattern of chord quality transitions.

Callender et al. also describe another concept called *cardinality equivalence* (C), which is intended to relate chords with different numbers of voices but similar pitch content [5]. Callender et al. define C-equivalent chords as those that only differ by neighboring repetitions of voices. So, $\langle 0,4,7 \rangle$ would be C-equivalent to $\langle 0,4,4,7 \rangle$ and $\langle 0,0,4,7 \rangle$ but not to $\langle 0,4,7,0 \rangle$. While this relation is somewhat difficult to map to a useful musical concept by itself, C-equivalence becomes far more intuitive when

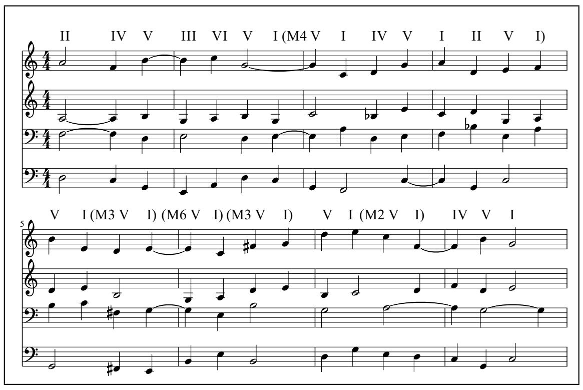
II IV V III VI V I (M4 V I IV V I II V I)
V I (M3 V I) (M6 V I) (M3 V I) V I (M2 V I) IV V I

Figure 1. An example of a long progression from our implementation in the key of C-major.

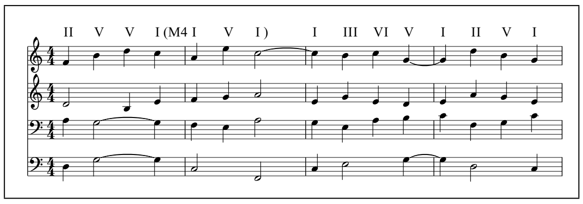

Figure 2. An example of a short progression from our implementation in the key of C-major.

combined with the O and P relations. PC-equivalence relates chords with the same sets of pitches, and OPC-equivalent chords have the same sets of pitch classes.

OPC-equivalence is a useful level of abstraction for mapping concepts like Roman numerals to higher voice counts than triads. For example, $\langle 48,60,64,67 \rangle$ is OPC-equivalent to $\langle 0,4,7 \rangle$ as well as to $\langle 55,60,64,67 \rangle$. All are C-major chords containing the pitch classes C, E, and G. We use OPC-equivalence for turning strings of Roman numerals generated by our grammar into concrete chord progressions.

4. A COMPLETE EXAMPLE

We define in this section a specific instance of a probabilistic TGGG based on a simple model of Western harmony. This grammar is not meant to be the only one possible using the features defined so far, but rather serves as an illustration of the key principals in an algorithmic composition setting.

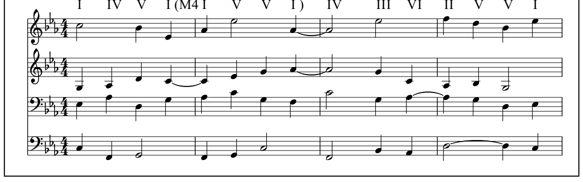

Figure 3. An example of a short progression from our implementation in the key of C-minor.

4.1. A Specific Grammar

As discussed earlier, usage of time in the rules affects the metrical structure of the result. For example, if the rule $I^t \rightarrow V^{t/2}\ I^{t/2}$ were applied to a I-chord spanning a single measure in 3/4, chord changes would fall between beats. This kind of non-metrical division of time in the rules is not prohibited. In fact, some interesting rhythmic results might be achieved by mixing rules with different metrical structures or using rules and add or subtract time from the duration represented by the left hand side.

The example given here is targeted at simple, classical music in 4/4. Therefore, we use metrical structures similar to those given in GTTM. All of the implemented rules have one of the following two forms:

$$c^t \rightarrow c_0^{t_0}\, c_1^{t_1} \dots c_n^{t_n} \mid c^t, c_i^{t_i} \in N, \wedge \sum_{i=0}^{n} t_i = t \qquad (6)$$

$$c^t \rightarrow (m\ c_0^{t_0} \dots c_n^{t_n}) \mid m \in M \wedge c^t, c_i^{t_i} \in N \wedge \sum_{i=0}^{n} t_i = t \qquad (7)$$

Enforcing equal total durations on both sides of the rules allows us to impose metrical constraints. For music in 4/4, an easy way to preserve a sense of meter in generated progressions is to only use even divisions of time, such as dividing time into halves or quarters.

The production rules for our grammar are shown in Table 1. For the right hand sides, we use phrases from existing harmonic analyses of Baroque music, primarily those found in [18]. All nonterminals have an identity rule and some number of other rules that produce short phrases and/or modulations. All nonterminals except for *I* have at least one rule that can produce a modulation. The complete listing of rules is shown in table 1. All progressions produced were interpreted in C-major or C-minor with a 4/4 time signature. Because our grammar is stochastic, each rule must have an associated probability and all probabilities for rules with the same left-hand side must sum to 1.0.

The production rules for *I* enforce the constraint that any progression produced will always end on the global tonic of the progression (the tonic of the start symbol). The rules also observe metric constraints that allow the preservation of a sense of meter in produced strings. Time is assigned evenly between chords when the number of chords is even (two or four chords) and is divided into halves or quarters only when the number of chords is odd. These rules ensure that some of the metrical well-formedness rules from GTTM are preserved, such as that beats at higher levels (more abstract levels) are still beats at lower levels.

The probabilities assigned to each rule were chosen by hand. Rules were assigned relatively uniform probabilities except for rules that produce patterns that should be somewhat rare in music, such as modulations. For these cases, we adjusted the probabilities to allow enough probability mass that the events would be readily observable during testing without completely dominating the output.

Num.	Probability	Rule
1	0.20	$I^t \rightarrow II^{t/4}\ V^{t/4}\ I^{t/2}$
2	0.20	$I^t \rightarrow I^{t/4}\ IV^{t/4}\ V^{t/4}\ I^{t/4}$
3	0.20	$I^t \rightarrow V^{t/2}\ I^{t/2}$
4	0.20	$I^t \rightarrow I^{t/4}\ II^{t/4}\ V^{t/4}\ I^{t/4}$
5	0.20	$I^t \rightarrow I^t$
6	0.80	$II^t \rightarrow II^t$
7	0.20	$II^t \rightarrow (M_2\ V^{t/2}\ I^{t/2})$
8	0.70	$III^t \rightarrow III^t$
9	0.30	$III^t \rightarrow (M_3\ I^t)$
10	0.80	$IV^t \rightarrow IV^t$
11	0.20	$IV^t \rightarrow (M_4\ I^{t/4}\ V^{t/4}\ I^{t/2})$
12	0.10	$V^t \rightarrow V^t$
13	0.15	$V^t \rightarrow IV^{t/2}\ V^{t/2}$
14	0.10	$V^t \rightarrow III^{t/2}\ VI^{t/2}$
15	0.10	$V^t \rightarrow I^{t/4}\ III^{t/4}\ VI^{t/4}\ I^{t/4}$
16	0.10	$V^t \rightarrow V^{t/4}\ VI^{t/4}\ VII^{t/4}\ V^{t/4}$
17	0.10	$V^t \rightarrow V^{t/2}\ VI^{t/2}$
18	0.10	$V^t \rightarrow III^t$
19	0.05	$V^t \rightarrow (M_7\ V^t)$
20	0.10	$V^t \rightarrow VII^t$
22	0.10	$V^t \rightarrow (M_5\ I^t)$
22	0.70	$VI^t \rightarrow VI^t$
23	0.30	$VI^t \rightarrow (M_6\ I^t)$
24	0.40	$VII^t \rightarrow VII^t$
25	0.50	$VII^t \rightarrow I^{t/2}\ III^{t/2}$
26	0.10	$VII^t \rightarrow (M_7\ I^t)$

Table 1. Production rules of a sample grammar that generates Figures 1, 2, and 3.

Clearly this type of feature would be best learned from a data set if the goal is to reproduce an existing style of music, but this is a subject of on-going and future work and, therefore, not yet realized.

4.2. Results

We tested our implementation on the above grammar with global tonics of both C-major and C-minor on two sets of conditions: long and short phrases, each with a minimum duration of a quarter note for chords. Long phrases were 8-measure progressions with 16 iterations of the *gen* algorithm. Short progressions were 4-measure progressions with 8 iterations of the *gen* algorithm. In all cases, OPC-space for four voices with the ranges [40,56], [50,62], [55,70], and [60,78] was used to map Roman numerals to pitches. The lowest voice was constrained to double either the root or fifth of the chord, voice-crossing was prohibited, and movement of no more than 7 half steps in any voice was preferred. Notes within a voice that did not move between chords were tied. Output from our grammar example can be seen in figures 1, 2, and 3.

In general, results were promising, particularly considering the simplicity of the grammar and lack of production differences between major and minor settings. As expected, the process used to create the set of rules for our implementation resulted in many produced passages that were harmonically similar to the style of the source material. The number of transitions that were uncharacteristic for the style increased with the length of the example and the number of iterations. The addition of C-equivalence demonstrated improvement over previous results that used only O and P in a similar task [16], allowing > 3 voices and more natural-sounding transitions between chords because of the voice-doubling.

The results from the combination of our grammar and chord spaces are progressions that could easily be further-interpreted to add melodic elements to what is essentially a harmonic backbone. The introduction of non-chordal scale tones would create richer-sounding music, and this is an area of ongoing work.

Some undesirable dissonances resulted from the fact that our grammar does not account for differences between modes, particularly from transitions between modulations that occur back-to-back and II-chords interpreted in a minor setting. This is partly because our implementation interprets a II-chord in a minor key as a diminished chord, adhering strictly to the scale indices of the triad rooted at the second degree. Because of this, minor results tended to be more dissonant than major results, since II-chords were common in the output.

As shown in our examples, it is possible to get back-to-back modulations, some of which can result in genre-inappropriate chord transitions. Because of problems like this, rules were designed to generally avoid the possibility of nested modulations without intermediate chord terminals, such as $M_5(M_5(M_5(...)))$. Such cases would easily result in transitions like C-major to F#-major or even C#-major, which although acceptable in some styles of modern music, would not be desirable in our target style of music (simple, chorale-like passages).

5. RELATED WORK

5.1. Generating Harmony

Generating harmony is a popular subject in automated composition research. A wide variety of algorithms have been explored, including Markov chain-based approaches [24, 6], neural nets [11, 1, 2], and more specialized systems intended for generating whole compositions [7, 8].

Neural nets and Markov chains have been explored for problems where acceptable harmonies have been learned from data sets and the melody is given [11, 24, 6, 9]. These algorithms can produce harmonies very close to the style of the input music, even while respecting some other music theoretic constraints [11]. Boltzmann machines, a type of neural net, can exhibit different musical behaviors: acting as a classifier, performing fill-in-the-blank problems, or even creating a completely novel composition [1, 2]. Markov chains require context in the states to retain any musical structure, and even variable-length Markov chains [4] can run into a state explosion problem as the amount of context required to make good decisions increases.

Harmony generation has also been a focus of complex, very specialized systems for generating complete compositions. Two such examples are David Cope's EMI [7] and Kemal Ebcioglu's chorale harmonization system [8]. These systems produce impressive results, even if not easily generalizable.

5.2. Musical Grammars

Grammars have been explored both generatively and analytically in music. Studies on brain activity have shown a strong link between language and music in the brain [3], an idea that has become increasingly accepted in music theory through works like GTTM, which presents a grammatical outlook on analyzing music [14] (although it requires additional formalization to be implemented in both analytical and generative settings). Simple grammars like L-systems have been explored for generating various musical features [23] and have also been the subject for fractal-based, algorithmic compositions using features of chord spaces [10]. Grammars have also been used to generate jazz riffs [13].

Martin Rohrmeier introduced a mostly context-free grammar (CFG) for parsing classical Western harmony [19]. The grammar is based on the tonic, dominant, and subdominant chord functions. Terminals are the Roman numerals from I to VII, and the nonterminals are *Piece*, *P* (phrase), TR (tonic region), *DR* (dominant region), *SR* (subdominant region), T (tonic), D (dominant), S (subdominant), and four chord function substitutions.

Nonterminals like T, TR, and P demonstrate that a formal grammar with many accepted music theoretic concepts can correctly parse many pieces of music. However, those nonterminals introduce some problems in a generative setting. The smallest number of generations to produce a I terminal would be the following: $Piece \to P$, $P \to TR$, $TR \to T$, and finally $T \to I$.

The grammar also has more than one rule for each nonterminal, such as $P \to P\ P$ and $TR \to TR\ TR$. Probabilities would need to be assigned to these rules to be used generatively, thereby turning the grammar into a PCFG. Because of the degree of separation between the start symbol and terminals, the PCFG would require significant algorithmic intervention to produce a string of terminals of reasonable length within a finite number of steps. With simple stochastic generation, it would be easy to still have a mixed string of terminals and nonterminals after many generative steps.

One way to fix the terminal-production problems in Rohrmeier's grammar would be to simply interpret symbols like TR and *DR*, which have obvious chordal implications, as their nearest terminal representatives, I and V respectively. This could be implemented by simply applying appropriate rules deterministically at the end of a stochastic generation, but it could also be viewed as blurring the distinction between a chord (terminal) and the more abstract symbols representing chord function (nonterminals). In our TGGG, we make precisely that simplifying assumption: that intermediate structures like TR can simply be re-written as Roman numerals.

The TGGG example we give for generating harmonic structure avoids the terminal-production problems associated with grammars like Rohrmeier's while still producing similar strings to those producible by Rohrmeier's grammar. To accomplish this, we make the simplifying assumption that there is no generative difference between a Roman numeral like I and a higher-level nonterminal like TR. Instead, a TR that produces a section of several chords would be viewed as a I-chord that produces the section. We make one major change, however, from the type of harmonic symbols used by Rohrmeier: temporal aspects of music are built into the grammar's alphabet. This allows our grammar to internally capture the metrical structure, whereas grammars like Rohrmeier's would have to delegate metrical decisions to the algorithm applying the grammar.

5.3. Metrical Structure

GTTM presents the notion of temporal groups. Groups are strictly hierarchical, and any group that is divided into subgroups must be completely divided into subgroups. Preference for grouping musical objects is given to even divisions of objects, with a grouping that divides a larger section in half being preferable to one that divides it into three sections. Metrical aspects of music obviously adhere to this kind of structure, with nesting of sub-beats within beats, beats within measures, and measures within larger phrases. [14] Meter is clearly important in the rules featured in GTTM, since it interacts with harmonic aspects of the music through the grouping and preference rules [14]. Temperley's work [21] as well as a harmonic analysis algorithm by Raphael and Stoddard [17] also emphasize the role of rhythm and meter in the perception of harmony.

Lack of metrical features in the grammar places metrical responsibility on the algorithm responsible for choosing production rules to apply. This is a common problem with simple musical L-systems [23], which are prone to creating results that sound meter-less. Keller and Morrison present a grammar for jazz riffs where metrical constraints are directly addressed [13]. In their grammar, rhythmic generation occurs first in a hierarchical way, similar to the musical groups described in GTTM. Once rhythmic generation has occured, rhythmic symbols are mapped to pitches by a separate set of rules.

5.4. Repetition

Another problem associated with musical grammars is the repetitive nature of music. Consider a fugue: the subject that opens the piece is expected to appear in modified states later on in the music. If these constraints are ignored, the form of the music is violated. The various musical grammars discussed so far have no support for this kind of musical feature.

Many of the other algorithms discussed so far also lack the support for repetition needed to make a coherent, long piece of music. Markov chains are an example of an algorithm that is doomed to failure when trying to capture repetition at more than a very local level. For music in even a relatively simple repetitive form like ABA, replicating the A section at the end of the piece would require the context length for the states to include all previously traversed states. Clearly this creates a combinatorial problem. The neural net and Boltzmann machine implementations discussed previously are not free from this problem either.

5.5. Context Sensitivity

Context-sensitive grammars are more powerful than context-free grammars, and grammars like Rohrmeier's are more concisely represented using context-sensitive features, such as the home key parameters appearing in some of Rohrmeier's rules (different productions are available depending on whether the local tonic is major or minor). However, context-sensitive grammars are harder to parse than context-free grammars, and they are also more difficult to use generatively. Context-sensitivity introduces the problem of pattern-matching into generation, a task that is not always simple, while this is a non-issue in a context-free setting. For this reason, the harmonic production rules of the grammar presented here are context-free, although context-sensitive extensions are easy to imagine.

6. FUTURE WORK

We have presented a category of grammars, TGGGs, that can capture harmonic and metrical structure in addition to repetition–a feature not directly captured in other proposed harmonic grammars. We showed a specific example of one such grammar for classical Western music, but the larger category of grammars is not limited to this style of music. Integration with chord spaces allows the generation of actual music, not just abstract harmony.

An obvious extension of our grammar for classical harmonies would be the introduction of some degree of context-sensitivity to account for the mode of the local tonic for any given production. Another obvious extension to the grammar would be additional chordal symbols to accommodate more modern music. The 7-symbol Roman numeral system for triads is clearly insufficient to handle the nuances of jazz harmonies. Attempting to generate jazz with those symbols would delegate the responsibility of deciding on specific 4-voice chords to another algorithm. Although this is a possibility (it could, for example, be delegated as constraints to the chord spaces framework), it is a non-trivial task, and it seems more intuitive to account for a somewhat more diverse set of chords within the grammar itself.

Another improvement would be to learn aspects of the grammar from data, such as the production probabilities. This would allow more exact replication of a given style of music and would also allow for a more general grammar (such as Rohrmeier's) to be adapted to a specific style of music that it encompasses.

Although difficult, it would also be ideal to learn specific production rules for a grammar like ours from data sets in addition to learning the production probabilities. For our grammar, we hand-picked rules using examples of analyzed phrases taken from existing music (namely analyzed examples of Bach) along with our own intuition. It may be possible to automate this process and find phrases that would make good rules by searching through an analyzed data set. GTTM-inspired phrase segmentation similar to the method implemented by Ito et al. [12] may be a useful step in this process.

Finally, the use of chord spaces as a structural aid is one of the most promising features of our implementation, and it could be extended. More complex musical structures, such as the subject, answer, and counter-subject restrictions of fugue-form, could be encoded as constraints used by the chord space framework we used to turn our grammar's strings, or harmonic phrases, into more elaborate, concrete music. Chord spaces at different levels of abstraction could also be used to alter the style of the music. For example, the Roman numerals generated by a grammar for classical music could be interpreted through a chord space built for jazz to give richer harmonies.

Acknowledgements This research was supported in part by NSF grant CCF-0811665.

7. REFERENCES

[1] M. I. Bellgard and C.-P. Tsang, "Harmonizing music the Boltzmann way," *Connection Science*, vol. 6, no. 2, pp. 281–297, 1994.

[2] ——, "On the use of an effective Boltzmann machine for musical style recognition and harmonization," in *Int. Computer Music Conf.*, 1996, pp. 461–464.

[3] S. Brown, M. J. Martinez, and L. M. Parsons, "Music and language side by side in the brain: a PET study of the generation of melodies and sentences," in *European Journal of Neuroscience*, 2006.

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

[5] C. Callender, I. Quinn, and D. Tymoczko, "Generalized voice-leading spaces," *Science Magazine*, vol. 320, no. 5874, pp. 346–348, 2008.

[6] B. J. Clement, "Learning harmonic progression using Markov models," in *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1998.

[7] D. Cope, "An expert system for computer-assisted composition," *Computer Music Journal*, vol. 11, no. 4, pp. 30–46, 1987.

[8] K. Ebcioğlu, "An expert system for Schenkerian synthesis of chorales in the style of J.S. Bach," in *ICMC*, 1984.

[9] D. Eck and J. Schmidhuber, "Learning the long-term structure of the blues," in *In Proc. Intl. Conf. on Artificial Neural Networks*, 2002, pp. 284–289.

[10] M. Gogins, "Score generation in voice-leading and chord spaces," in *ICMC*, 2006.

[11] D. Hörnel, "Chordnet: Learning and producing voice leading with neural networks and dynamic programming," *Journal of New Music Research*, vol. 33, no. 4, pp. 387–397, 2004.

[12] Y. Ito, Y. Takegawa, T. Terada, and M. Tsukamoto, "A system for memorizing songs by presenting musical structures based on phrase similarity," in *Proc. of the Int. Computer Music Conf.*, 2012, pp. 269–272.

[13] R. M. Keller and D. R. Morrison, "A grammatical approach to automatic improvisation," in *Sound and Music Computing Conf.*, 2007.

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

[15] S. Peyton Jones, "The Haskell 98 language and libraries: The revised report," *Journal of Functional Programming*, vol. 13, no. 1, pp. 0–255, Jan 2003. [Online]. Available: www.haskell.org/definition

[16] D. Quick and P. Hudak, "Computing with chord spaces," in *Int. Computer Music Conf.*, September 2012.

[17] C. Raphael and J. Stoddard, *Computer Music Journal*, vol. 28, no. 3, pp. 45–52, 2004.

[18] W. Renwick, *Analyzing Fugue: a Schenkerian Approach*. Stuyvesant, NY: Pendragon Press, 1995.

[19] M. Rohrmeier, "Towards a generative syntax of tonal harmony," *Journal of Mathematics and Music*, vol. 5, no. 1, pp. 35–53, 2011.

[20] S. W. Smoliar, "A computer aid for Schenkerian analysis," in *Proc. of the 1979 Annual ACM Conf.*, 1979.

[21] D. Temperley, *Music and Probability*. The MIT Press, 2010.

[22] D. Tymoczko, "The geometry of musical chords," *Science Magazine*, vol. 313, no. 5783, pp. 72–74, 2006.

[23] P. Worth and S. Stepney, "Growing music: musical interpretations of l-systems," *Applications on Evolutionary Computing*, 2005.

[24] L. Yi and J. Goldsmith, "Automatic generation of four-part harmony," in *UAI Applications Workshop*, 2007.