Journal of New Music Research

ISSN: 0929-8215 (Print) 1744-5027 (Online) Journal homepage: http://www.tandfonline.com/loi/nnmr20

Learning production probabilities for musical grammars

Donya Quick

To cite this article: Donya Quick (2016) Learning production probabilities for musical grammars, Journal of New Music Research, 45:4, 295-313, DOI: 10.1080/09298215.2016.1228680

To link to this article: http://dx.doi.org/10.1080/09298215.2016.1228680

Published online: 16 Nov 2016.

Journal of New Music Research, 2016
Vol. 45, No. 4, 295–313, http://dx.doi.org/10.1080/09298215.2016.1228680

Learning production probabilities for musical grammars

Donya Quick

Department of Computer Science, Yale University, Dallas, TX, USA

(Received 6 September 2015; accepted 9 August 2016)

Abstract

While there is a growing body of work proposing grammars for music, there is little work testing analytical grammars in a generative setting. We explore the process of learning production probabilities for musical grammars from musical corpora and test the results using Kulitta, a recently developed framework for automated composition. To do this, we extend a well-known algorithm for learning production probabilities for context-free grammars (CFGs) to support various musical CFGs as well as an additional category of grammars called probabilistic temporal graph grammars.

1. Introduction

Music has many features in common with spoken language and is now often analysed using methods inspired by linguistics. Learning grammars, grammatical features and statistical models for grammars are common subjects of computational linguistics research, with learning algorithms having been proposed for various categories of grammars. Context free grammars (CFGs) are popular subjects for their simplicity, and probabilistic context-free grammars (PCFGs) in Chomsky Normal Form are possible to learn in $O(n^3)$ time with the *inside–outside algorithm* (Lari & Young, 1990). Learning algorithms for some categories of context-sensitive grammars have also been proposed (Clark, 2010).

Just as spoken language has the notion of parts of speech (noun, verb, etc.), music has abstract labels that are applied to various features (I-chord, passing tone, etc.), allowing a symbolic representation. The Generative Theory of Tonal Music (GTTM) argues for a linguistic treatment of music (Lerdahl & Jackendoff, 1996), although many of its definitions are mathematically informal and subject to ambiguity. Masatoshi Hamanaka et al. present a formal implementation of a subset of GTTM for homophonic music that resolves these ambiguities using a probabilistic approach (Hamanaka, Hirata, & Tojo, 2006). Rens Bod also showed that music and language can be parsed by the same statistical models (Bod, 2002), adding further weight to the connection between spoken language and music.

Stochastic and data-driven approaches to automated composition are appealing since they can yield more diverse results with less human effort. David Cope's EMI (Cope, 1987, 1992) is one such system, and learning algorithms such as Markov decision processes, neural nets and Boltzmann machines have also been used to generate various musical features (Bellgard & Tsang, 1994, 1996; Hörnel, 2004; Yi & Goldsmith, 2007). Kemal Ebcioğlu's expert system for harmonizing chorales emphasizes the importance of considering multiple aspects of music during generation. Raymond Whorley at al. present a more recent system that uses 'multiple viewpoints' to optimize the quality of the harmony generated for a given melody and compares the results and runtime of different collections of viewpoints (Whorley, Wiggins, Rodes, & Pearce, 2013). Learning-based approaches to automated composition are also useful for hypothesis testing since generating new data points is an easy way to see what features are and are not captured by a given model. Kulitta (Quick, 2014) is a recently proposed generative framework that is useful for this type of musical hypothesis testing and features a learning component that is detailed in the following sections.

In addition to harmony-based approaches, different methods have been proposed for melodic analysis and modelling such as geometric approaches (Yust, 2009) and the use of segment classes (Conklin (2006), 2006), which are a concept from linguistics. In Schenkerian theory, melodies contain a mixture of harmonic tones and other notes, many (but not necessarily all) of which are analysed away to determine the structure of the music (Schenker, 1954), and there have been some attempts to automate Schenkerian analysis (Smoliar, 1979; Kirlin & Utgoff, 2008). In line with the previously mentioned work, this suggests that there are also many levels

Correspondence: Donya Quick, Yale University Department of Computer Science, 8259 Southwestern Blvd. Apt 1066, Dallas, TX 75206, USA. E-mail: dquick@smu.edu

of abstraction present within a single melody and that notions of musical hierarchy are important for both analysis and generation.

We show that musical PCFGs and a new category of musical grammars used by Kulitta, called probabilistic temporal graph grammars (PTGGs), can have production probabilities derived from a corpus of analysed musical data. We accomplish this through the use of an oracle extension to the inside–outside algorithm for learning production probabilities for PCFGs.

Finally, we examine these learned models generatively, by using Kulitta's generative ability to create new phrases of music with the learned production probabilities. Our results show support for the notion that music can be modelled grammatically, while also suggesting that metrical structure should be an important part of musical grammars.

2. Related work

Both Markov chains and probabilistic grammars have been used extensively for similar musical tasks, such as analysing and generating melodies and harmony. Indeed, the inside–outside algorithm we extend in this paper is closely related to learning algorithms for hidden Markov models (HMMs). Some of the extensions involve a new category of grammars, probabilistic temporal graph grammars, which are a fundamental component of the automated composition system called Kulitta. The following sections review important aspects of each of these topics.

2.1 Markov chains

One of the most commonly applied learning algorithms in computer music is the Markov chain (Chordia, Sastry, & Senturk, 2011; Gillick & Keller, 2009; Pachet, 2002; Yi & Goldsmith, 2007). There are two reasons for this: the algorithm is simple, and music is sequential in nature, lending itself to modelling a score as a series of state transitions (Alty, 1995). Markov chains are also a popular model in computational linguistics, with HMMs having shown promise in analytical tasks in both musical and linguistic domains for tasks such as speech recognition (Gales & Young, 2007; Juang & Rabiner, 1991) and harmonic analysis (Ponsford, Wiggins, & Mellish, 1999; Raphael & Stoddard, 2004; White, 2013a). The forward–backward algorithm (Rabiner, 1989) for learning HMMs is also very closely related to the inside–outside algorithm (Lari & Young, 1990) described in later sections. Essentially, the inside–outside algorithm is a generalization of the forward–backward algorithm over linear sequences to handle trees.

However, Markov chains are fundamentally limited in the temporal scope of features that they can capture in time series data like music and speech. As the order of the chain increases, more complex features can be modeled, but observations of states become sparse and the number of states explodes exponentially. Even approaches such as variable-length Markov chains (Bühlmann & Wyner, 1999; Ron, Singer, & Tishby, 1996) and the extensions noted above do not entirely overcome this problem. For example, consider musical phrases with repetition, such as the form AA, where the same musical phrase or section occurs twice. Capturing repetition for this form requires the entire A section to be part of the state. The more material that occurs between repetitions, such as ABA or ABCA form, the longer the state has to be.

Some extensions to Markov chains exist that can overcome certain structural problems seen in more traditional approaches. Two examples are partially observable Markov decision processes (White, 1991) and Hierarchical Dirichlet Process Hidden Semi-Markov Models (Johnson & Willsky, 2013), both of which allow for greater control over large-scale patterns (although not necessarily repetition as noted previously). In the musical domain, Roy and Pachet present an approach to modelling meter in melodic generation (Roy & Pachet, 2013) using Markov constraints (Pachet & Roy, 2011), which are another extension to traditional Markov chains. Roy and Pachet's approach involves a reformulation of the Markov model such that it incorporates a constraint satisfaction problem. This allows for properties in the output such as properly filled measures.[1] Roy and Pachet's method involves some degree of enumeration of possible outcomes for metrical constraints, even though the complexity of doing so is mitigated by a filtering step to avoid truly searching all possible paths. Kulitta's PTGGs differ significantly in their approach to generating acceptable metrical structures by iteratively working from large-scale metrical patterns towards smaller ones rather than searching through different possible combinations at the lowest desired level of detail.

The expressiveness of traditional Markov chains and HMMs is a subset of that of PCFGs (Smith & Johnson, 2007). PCFGs, therefore, are more powerful, and capturing long-term structures is a distinct advantage of other categories of grammars over Markov chains. Many other more powerful categories of grammars also exist that may be better suited to representing music. For example, supervised learning with combinatory categorial grammars has achieved better accuracy for harmonic analysis of jazz than more traditional Markovian approaches (Granroth-Wilding & Steedman, 2012). Kulitta's PTGGs are yet another category of grammars that feature a let-in structure for variable instantiation specifically to easily address repetition over long spans of symbols (Quick, 2014).

2.2 Musical grammars

A CFG 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. CFGs can be used both analytically to parse sequences of symbols or generatively to produce new sequences. A PCFG

[1]Ending at a nonsensical metrical position is a common problem for more traditional Markov chains, particularly those that contain explicit end states.

is a CFG for which each rule has an associated probability of application, also called a production probability. In a PCFG, each nonterminal has a probability distribution over its rules; the probabilities for rules with the same left-hand side should sum to 1.

Lindenmayer systems (Prusinkiewicz & Lindenmayer, 1990), or L-systems, are a special category of terminal-less CFG for generating fractal-like structures. Typically, a specific generative algorithm is also assumed for L-systems: beginning with the start symbol, the algorithm iteratively passes over the sequence from left to right, applying a rule to each symbol. During one generative iteration, each symbol is expanded, and, because there are no terminals, generation can proceed infinitely. L-systems can also be probabilistic, with more than one possible rule to apply to each symbol.

Grammars for music have been explored both generatively (Gogins, 2006; Keller & Morrison, 2007; McCormack, 1996; Worth & Stepney, 2005) and analytically (Abdallah, Gold, & Marsden, 2016; Lerdahl & Jackendoff, 1996; Rohrmeier, 2011; Winograd, 1968). Studies on brain activity have shown a strong link between language and music in the brain (Brown, Martinez & Parsons, 2006), an idea that has become increasingly accepted in music theory through works like GTTM.

Martin Rohrmeier introduced a mostly CFG for parsing classical Western harmony (Rohrmeier, 2011). 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. A modification of this grammar by W. Bas de Haas et al. has been used for automatic harmonic analysis on jazz standards (De Haas, Rohrmeier, Veltkamp, & Wiering, 2009).

Rohrmeier's grammar also exhibits a small amount of context sensitivity based on mode. For example, tonic chords, denoted as T, are given different, modally determined productions (Rohrmeier, 2011). While unambiguous in an analytical setting as a CFG, if turned into a PCFG, Rohrmeier's grammar would allow for different probabilistic behaviour depending on the mode. To examine the roll of such modal differences in music, in one of our experiments, we examine differences in learned production probabilities using major and minor subsets of a corpus of Bach chorales.

2.3 Kulitta

Kulitta is a modular framework for automated composition in a variety of styles (Quick, 2014). Kulitta employs an important principle in order to tackle these sorts of problems in a tractable way: musical abstraction. This helps to break daunting tasks with large solution spaces into smaller problems, allowing a solution to emerge in progressively finer levels of detail, much like a sculpture being chiselled out of stone.

As the first step in generating a new composition, Kulitta uses a category of musical grammars called *Probabilistic Temporal Graph Grammars* (Quick, 2014; Quick & Hudak, 2013a) to generate abstract chord progressions. These progressions are then processed through chord spaces (Callender, Quinn, & Tymoczko, 2008; Tymoczko, 2006) to obtain concrete, playable chords. Finally, style-specific foreground algorithms alter the chords and insert additional notes to produce a complete piece of music.

2.4 Probabilistic temporal graph gramars

Probabilistic Temporal Graph Grammars, or PTGGs, are a category of generative grammars used by Kulitta. They have two interesting properties: (1) the set of parameterized symbols is potentially infinite and (2) the rules are functions that can exhibit complex behaviour.

PTGGs may have a variety of information stored in the symbol parameters and can use any finite alphabet of symbols. For example, in addition to the harmony-based PTGGs described later on, PTGGs can also model melodic motion (Quick, 2015) and the parameters can be complex, storing information on duration, note onset, key/mode, and other musical contexts. The PTGG's illustrated in this paper use an alphabet consists of chords parameterized with durations.[2] For a finite set of chord symbols, C, the set of nonterminals would be $N = \{c^t | c \in C, t \in \mathbb{R}_{>0}\}$. C must be a finite set of symbols, such as the Roman numerals, $\{I, II, \ldots, VII\}$, or chord functions, $\{T, S, D\}$ (tonic, subdominant and dominant). PTGGs make no formal distinction between terminals and nonterminals, although symbols that do not occur on the left-hand side of a production are functionally terminals. An example of these would be modulation symbols used in some PTGGs, which create nested harmonic structure but cannot be expanded (Quick, 2014, Quick & Hudak, 2013b).

Rules in the PTGGs used in our experiments are functions of duration from chords to chord progressions, having the form $c^t \rightarrow f(t)$. These PTGGs are also *duration preserving*, such that the sum of durations on the right-hand side of a rule must equal the input duration on the left. For example, $I^t \rightarrow V^{t/2}\ I^{t/2}$ takes a I chord and divides its duration equally between the two chords of a V-I progression.

One of the purposes of PTGGs is to encapsulate information within a grammar's rules that would otherwise be delegated to a generative algorithm. A PTGG allows the creation of structural specifications while utilizing a generative process that has no domain-specific knowledge. The generative al-

[2]Durations in music have a number of possible representations, including symbolic notations, measurements relative to metrical units and specific measurements of time in seconds. PTGGs do not place any particular constraint on which representation should be used. For the purposes of this paper, we consider durations as positive real numbers to capture both relative representations such as whole note (1.0) and half note (0.5). as well as potentially values in seconds. We also do not exclude the potential for PTGGs with other alphabets that may make use of zero-duration symbols, which could be used to indicate a non-note event, or even negative durations, which might be useful for addressing ornaments or anacruses.

gorithm used with PTGGs in Kulitta is essentially that of an L-System, iteratively applying rules to symbols left to right according to their production probabilities. Generative iterations continue until either a pre-specified stopping point is reached (such as five iterations), or, in some cases, may proceed indefinitely and achieve a fixed point.[3] This allows the same generative algorithm to be applied to a variety of PTGGs that have different alphabets and parameters. If the same parameterized information were handled outside of the rules, a new generative algorithm would be required for different varieties of parameters.

PTGGs also permit two features that are common in programming languages, but somewhat unusual in other domains: conditional behaviour (based on the parameter of a symbol) and let-in expressions for variable declaration (Quick, 2014, 2015; Quick & Hudak, 2013b). Conditional behaviour allows for rules to create different results at different stages in the generative process. This allows for greater control over metrical divisions and can be used to prevent excessively small durations. Let-in expressions are useful for producing repetition in music. For example, PTGGs can capture generation of AABA format musical features:

$$X^t \rightarrow \text{ let } x = A^{t/4} \text{ in } x\ x\ B^{t/4}\ x$$

where X, A and B are nonterminals in some musical alphabet, such as Roman numerals or chord functions. While these rules are typically most useful early in the generative process, they can also add smaller-level repetitions. Additionally, they allow easy generation of variations with a three-step process: first generating a sequence that contains variables, expanding the sequence to replace variables with their definitions, and then continuing generation again such that instantiated variable instances are allowed to diverge.

We present an oracle-based generalization of the inside–outside algorithm that allows learning of production probabilities from a musical corpus for both musical PCFGs, such as a probabilistic version of Martin Rohrmeier's CFG for harmony, and Kulitta's PTGGs. In this paper, we focus on extensions necessary to address the parameterization and function-based rules of PTGGs, although the same general framework can also be used to address rules exhibiting conditional behaviour and those with let-in expressions.

3. The inside–outside algorithm

The inside–outside algorithm is an approach for learning production probabilities for a PCFG analogously to how the forward–backward algorithm learns state transition probabilities for HMMs (Lari & Young, citeyearlari1990). Given a PCFG and a data-set presumed to be generated by that PCFG, the algorithm's goal is to find production probabilities for that PCFG that maximize the probability of the data-set. Instead of computing probabilities over a linear sequence of symbols, the inside–outside algorithm computes probabilities over parse trees. The *inside* probability of a particular node in the parse tree is the probability of generating the subtree rooted at that node, and the *outside* probability of a node is the probability of generating the rest of the parse tree minus that node's subtree. Inside and outside probabilities are, respectively, analogous to the forward and backward probabilities of a HMM.

The rules of a PCFG must be supplied as input to the inside–outside algorithm along with initial estimates for the production probabilities. The algorithm then iteratively re-estimates the production probabilities for each rule. Rules are expected to be in *Chomsky normal form* for a PCFG. This form allows the only two[4] types of rules, where capital letters indicate nonterminals and lowercase letters represent terminals: $A \rightarrow BC$ and $A \rightarrow x$.

Before production probabilities can be re-estimated, strings in the training data must first be parsed to determine which rules might have been applied. Parsing must also be done in a way that accounts for ambiguity since some strings may have more than one possible parse tree. The CYK algorithm can be used for this.

3.1 CYK parsing

John Cocke, Daniel Younger and Tadao Kasami described a parsing algorithm that is now called the CYK algorithm, or sometimes CKY algorithm (Younger, 1967; Kasami, 1965, as cited in Mitkov 2003). It approaches the parsing task by filling in a table rather than building a parse tree directly. The resulting table can be used to determine whether a given string is accepted by a language, but finding a specific parse requires some extra work. A similar approach to the CYK algorithm was described by J. Opatrny and K. Culik II for parsing EOL-languages (which include L-Systems and the CFGs for harmony described in this paper), with an overall complexity of $O(n^4)$ (Opatrny & Culik II, 1976).

A CYK parse table shows which nonterminals can produce each substring of the input sequence. For a string of length n, rows are numbered from 0 to n and columns from 1 to n. Row 0 contains the string itself. A symbol at row r and column c must be able to produce symbols c through $c+r-1$ via some series of rule applications. Consequently, strings accepted by a given language will yield the start symbol in the topmost cell, which accounts for the entire string. Tables are constructed from the bottom up. For example, given the rules $S \rightarrow AA$, $A \rightarrow AA$ and $A \rightarrow a$, the CYK table for aaa would be:

	1	2	3
3	S, A		
2	S, A	S, A	
1	A	A	A
0	a	a	a

[3]A fixed point is an input, x, for a function, f, such that $f(x) = x$.

[4]Sometimes a third type of rule, $S \rightarrow \epsilon$, is also included to allow the start symbol to produce the empty string. However, such a rule is not useful in the context of this paper and therefore is not considered.

Note that this particular sequence is ambiguous, and all possible parses are represented in the table as well as extraneous symbols that can produce portions of the string but are not involved in any full parse of the string. The A appearing at row 3, column 1 is an example of this, as are the start symbols, S, appearing below row 3.

3.2 Learning production probabilities

The symbol ψ is used to denote the probability mass function over rules: for a rule, r, with probability p, we will have $\psi(r) = p$. Probabilities for rules with the same left-hand side are assumed to sum to 1. The inside probability of a nonterminal A spanning terminals i through j is denoted $\alpha(A, i, j)$. This gives the probability of generating the subtree rooted at nonterminal A and ending in terminal symbols i through j. The outside probability of a nonterminal, A, spanning symbols i through j, is denoted $\beta(A, i, j)$. It accounts for all portions of the tree not addressed by $\alpha(A, i, j)$. In other words, $\beta(A, i, j)$ is the probability of generating the sequence: $x_1, \ldots, x_{i-1}, A, x_{j+1}, \ldots x_n$.

The α and β values are combined to calculate the probability of a rule appearing at a particular point in a string's parse tree. This quantity is called μ. For a given rule, this quantity is then summed or 'counted':

$$count(A \to x) = \sum_i \mu(A \to x,\ i) \tag{1a}$$

$$count(A \to BC) = \sum_{i,k,j} \mu(A \to BC,\ i,\ k,\ j) \tag{1b}$$

Let $count^s$ denote the counting equation over a particular string, s, in the data-set (all equations to this point have been for single strings). Re-estimation of the production probability for a rule $A \to \nu$ with data-set $S = \{s_1, \ldots, s_m\}$ is defined by normalizing its counts as follows:

$$\psi'(A \to \nu) = \frac{\sum_{s\in S} count^s(A \to \nu)}{\sum_{A\to\nu'\in R}(\sum_{s\in S} count^s(A \to \nu'))} \tag{2}$$

This recalculation of production probabilities can be done iteratively until the values converge to within some threshold. The inside–outside algorithm's approach of associating nonterminals with spans or substrings mirrors that of the CYK algorithm. For example, the value $\alpha(A, i, j)$ will be nonzero if A appears in column i of row $j - i$ in the parse table, and zero if A is not present in that cell. Similarly, $\beta(A, i, j)$ will only be nonzero if A is part of a parse tree. The CYK parse table can be used directly to locate (A, i, j) tuples over which calculations should be done.

4. Learning production probabilities for PCFGs

A slightly modified version of the inside–outside algorithm can be used to learn production probabilities for Martin Rohrmeier's CFG for harmony. However, Rohrmeier's grammar is not in Chomsky normal form. Rather than modifying the grammar to place it in Chomsky normal form, we can also modify the inside–outside algorithm to handle new types of rules. Rules such as $TR \to T$ and $I \to I\ IV\ I$ in Rohrmeier's grammar require that the algorithm handle rules of the form $A \to BCD$ and $A \to B$. This enhances readability without greatly impacting the runtime for grammars such as Rohrmeier's and also is useful for later extensions for PTGGs described in later sections.

The *rank* of a rule is the number of symbols that appear on the right-hand side. Rohrmeier's grammar uses rules of rank 1, 2 and 3. Rank 3 and rank 1 are relatively straightforward additions to the equations, although handling rules of rank 3 increases the worst-case complexity for the algorithm. The definition for the $i \neq j$ case of the inside probability formula becomes:

$$\begin{aligned}\alpha(A, i, j) = &\sum_{k,l,\ A\to BCD\in R}^{i\le k<l<j} [\psi(A \to BCD) \\ &\times \alpha(B, i, k) \times \alpha(C, k+1, l) \times \alpha(D, l+1, j)] \\ &+ \sum_{k,\ A\to BC\in R}^{i\le k<j} [\psi(A \to BC) \times \alpha(B, i, k) \\ &\times \alpha(C, k+1, j)] \\ &+ \sum_{A\to B\in R} [\psi(A \to B) \times \alpha(B, i, j)]\end{aligned} \tag{3}$$

The changes to β are also fairly straightforward, although rather verbose. To simplify the definition, we will denote β for rules of rank 2 as β_2 and the new definition for rules of rank 3 as β_3.

$$\begin{aligned}&\beta_3(A, i, j) \\ &= \beta_2(A, i, j) \\ &+ \sum_{k,\ l,\ B\to ACD\in R}^{j<k<l\le n} [\psi(B \to ACD) \\ &\times \alpha(C, j+1, k) \times \alpha(D, k+1, l) \\ &\times \beta(B, i, l)] \\ &+ \sum_{k,\ l,\ B\to CAD\in R}^{1\le k<i\le j<l} [\psi(B \to CAD) \\ &\times \alpha(C, k, i-1) \times \alpha(D, j+1, l) \times \beta(B, k, l)] \\ &+ \sum_{k,\ l,\ B\to CDA\in R}^{1\le k<l<i} [\psi(B \to CDA) \\ &\times \alpha(C, k, l-1) \times \alpha(D, l, i-1) \times \beta(B, k, j)] \\ &+ \sum_{B\to A\in R} [\psi(B \to A) \times \beta(B, i, j)]\end{aligned} \tag{4}$$

A final modification must take place in the formula for μ and *count* as well to handle rules of rank 3, although the definitions of μ and *count* for rules of rank 2 remain the same. This increases the computational complexity of the algorithm, but, in this case, this was judged to be outweighed by the simplicity of conforming to Rohrmeier's rule formulations and by the usefulness of accommodating rules of this form in extensions for PTGGs described in later sections.

$$\mu(A \to B, i, j) = \beta_3(A, i, j) \times \psi(A \to B) \times \alpha(B, i, j) \tag{5}$$

$$\mu(A \to BCD, i, k, l, j) = \beta_3(A, i, j) \times \psi(A \to BCD) \times \alpha(B, i, k) \times \alpha(C, k+1, l) \times \alpha(D, l+1, j) \tag{6}$$

$$count(A \to B) = \sum_{i,j}^{1 \le i \le j \le n} \mu(A \to B,\ i,\ j) \tag{7}$$

$$count(A \to BCD) = \sum_{i,k,l,j}^{1 \le i \le k < l < j \le n} \mu(A \to BCD,\ i,\ k,\ l,\ j) \tag{8}$$

5. Learning production probabilities for PTGGs

A PTGG is a parameterized grammar with a potentially infinite alphabet and rules that are functions. All of these features are problematic for the inside–outside algorithm even with the extensions discussed so far. Three key features of the grammar must be addressed:

(1) PTGGs make no distinction between terminals and non-terminals.
(2) Symbols in PTGGs must carry extra information, such as the parameters indicating duration, and the set of possible parameters is potentially infinite (duration, for example, can be any real number). However, a CYK-style parse table of a given progression will still be finite.
(3) Rules are *functions* that are later instantiated with concrete values. Productions such as $I^w \to V^h \ \ I^h$ and $I^h \to V^q \ \ I^q$ must be recognized as *instances* of the same rule, $I^t \to V^{t/2} \ \ I^{t/2}$.

In a more general sense, what we need is a way to learn a statistical model for a grammar where the full extent of the alphabet is unknown and where rules have the form of $X \to f(X)$. Here, we show an oracle-based approach to the inside–outside algorithm that allows this, and, when $f(X)$ has certain properties, we also show that an algorithm exists to implement the oracle.

5.1 An Oracle approach to the inside–outside algorithm

Suppose we don't know exactly what the rules for a grammar look like except for the assumption that they are context-free (in the sense that only one symbol can appear on the left-hand side of a rule). The rules could even exhibit conditional behaviour based on symbols' parameters, as is possible in a PTGG. In fact, the details of the rule set are unnecessary for the learning of production probabilities as long as the learning algorithm has access to the following things:

(1) An identifier for each abstract rule. This can simply be a number (i.e. 'rule 1' 'rule 2,' and so on).
(2) The production probability for each abstract rule (or the initial estimates of those probabilities).
(3) A partition of the abstract rules' identifiers into groups that share the same left-hand side.
(4) An oracle that takes a sequence of symbols and returns all *rule instances* that could have directly produced it along with their associated identifiers. The distinction between a rule and its instance is described in later sections.

Note that for a typical PCFG, there is no function/instance separation in the rules, so an oracle would return the rule itself. However, for a PTGG, the oracle would only return a rule instance, such as $I^w \to V^h \ \ I^h$, while the exact function that created it, $I^t \to V^{t/2} \ \ I^{t/2}$, would remain unknown to the learning algorithm. Importantly, the learning algorithm has neither an enumeration of the alphabet (which is potentially infinite for a PTGG) nor an enumeration of all possible rule instances. The algorithm only needs information about the symbols and rule instances that can be used to accomplish a CYK-style parse of the training data. Given the four pieces of information described above, production probabilities can be re-estimated by:

(1) Building a CYK-style parse table of rule instances for each string in the training data. Storing the full rule instances and their associated identifiers avoids any subsequent queries to the oracle once the parse table is complete.
(2) Traversing the parse table to compute α and β values as described in Section 5.5.
(3) Summing counts for rule instances by their rule identifiers when re-estimating production probabilities.

Step 1 above, building a CYK-style parse table, must address the lack of terminal/nonterminal distinction and the rule function/instance distinction for PTGGs. These cause some small, but cascading changes through the probability calculations. These issues are described in more detail in the following sections.

5.2 Removing the terminal/nonterminal distinction

Mathematically, the inside–outside algorithm does not actually enforce any important distinction between terminals and nonterminals in the traditional sense. Terminals are simply symbols that exist in row 0 and provide a stopping point for the recursive α calculations, much like the start symbol serves as a stopping point for β.

A lack of terminal/non-terminal distinction is an important property of Lindenmayer Systems, or L-Systems, a category of grammars commonly used for modeling fractals, where infinite self-similarity must be accounted for but only finite sequences can realistically be calculated. Consider an L-System with start symbol A and two rules: $A \rightarrow AB$ and $B \rightarrow A$. This grammar, defined by Prusinkiewicz and Lindenmayer (1990), consists entirely of non-terminals and produces strings such as $ABAAB$. All strings produced by the grammar can be further expanded. How much they are expanded depends on the generative algorithm used.

PTGGs also enforce no terminal/nonterminal distinction, and the generative algorithm used by PTGGs in Kulitta is fundamentally the same as that used with L-Systems. Although it would be possible to convert a PTGG to a more traditional parameterized grammar with strict terminals and nonterminals, allowing this distinction complicates the generative process even though it has no impact on CYK parsing. With a simple generative algorithm like those in L-Systems, two problems with a terminal/nonterminal distinction exist from a generative standpoint: (1) terminals may be generated unreasonably early and (2) nonterminals will be left over at the end, requiring a final corrective step to force nonterminals to terminals.

An example of the first case is the following sequence from Rohrmeier's grammar: $Piece \rightarrow P \rightarrow TR \rightarrow T \rightarrow I$. In this example, the entire piece ends up consisting of a single I chord. Similarly, illustrating the second problem, Rohrmier's grammar can generate a sequence such as $I\ TR\ V\ T$, which contains a mix of terminals and nonterminals. These problems could, of course, be solved by use of a more sophisticated generative algorithm. However, a generative algorithm that attempts to mitigate these problems may also effectively alter the probabilities of the rules. For example, suppose the two rules, $P \rightarrow TR$ and $P \rightarrow P\ P$ are equally weighted, but the generative algorithm waits until some number of applications of $P \rightarrow P\ P$ before considering a $P \rightarrow TR$ production. This essentially decreases the production probability for $P \rightarrow TR$ under the assumption of a simpler generative algorithm.

Knowledge of the generative grammar is critical to being able to accurately estimate production probabilities from observed sequences, and the generative algorithm for PTGGs is quite simple: apply rules to symbols left to right at each iteration in a way that fully respects assigned production probabilities. Even with such simplicity, it is still possible to avoid the two problems described due to the parameterized form of the alphabet. This shifts the burden of such decisions from the generative algorithm to the grammar itself in a way that is possible to still accommodate with CYK-style parsing.

Removing the terminal/nonterminal distinction in a grammar also implies that any point during generation of a sequence of symbols is a valid stopping point—again a useful feature in a generative setting that is less important analytically. In a grammar without terminals, rules of the form $A \rightarrow x$ are mathematically no different from rules of the form $A \rightarrow B$ where $A \neq B$ (situations involving 'identity rules' or 'self productions' of the form $A \rightarrow A$ will be addressed later). Rohrmeier's grammar has many of these, such as $TR \rightarrow T$ and $DR \rightarrow D$. Allowing for these rules, the equation for inside probabilities for single symbols in the sequence $X_1, \ldots, X_n$ becomes:

$$\alpha(A, i, i) = \begin{cases} 1.0 & \text{if } A = X_i \\ \sum\limits_{A \rightarrow B \in R} [\psi(A \rightarrow B) \times \alpha(B, i, i)] & \text{otherwise} \end{cases} \tag{9}$$

The definition for $\alpha(A, i, j)$ where $i \neq j$ remains the same as in Equation 3. True terminals will cause no problems; the only difference is that they are now candidates to be supplied to the α calculations.

5.3 Rule functions and rule instances

One of the trickiest aspects of PTGGs from a learning standpoint is the distinction that exists between rules, which are functions, and their instances, which are applications of those functions to specific values. For clarity, we will refer to PTGG rules as *rule functions* when referring to the function itself, such as $I^t \rightarrow V^{t/2} \quad I^{t/2}$ where t is a variable. We will refer to the applications of those rules to specific values as *rule instances*. For the rule function $I^t \rightarrow V^{t/2} \quad I^{t/2}$, the productions $I^w \rightarrow V^h \quad I^h$ and $I^h \rightarrow V^q \quad I^q$ are just two of many possible *instances* of the function.

For a traditional PCFG, the probability mass function, ψ, can simply perform a table lookup, pattern matching against either the left-hand side or right-hand side (or both) for calculating α and β values, respectively. For a PTGG, the process is more complicated. In training data, chords will have concrete durations, such as h (half note) rather than a variable, meaning that parsing must take place with concrete values as well. The parse tree must, therefore, be constructed using rule instances rather than rule functions.

The concept of rule functions and instances of those functions is broader than simply those in PTGGs. In fact, any grammar with rules of the form $X \rightarrow f(X)$ where X is a single symbol and $f^{-1}(X)$ is computable are covered by this paradigm, including the more standard category of PCFGs. The rules in PCFGs can be thought of as functions where each has only one instance.

5.4 Parsing with rule instances

With a PCFG, rule instances are no different from the actual rules in the grammar. However, with a PTGG, one rule can have many instances. How can we efficiently find these instances to recursively compute α and β values without knowing about the rules themselves? The answer lies in a simple change to the parse tree representation.

Consider the progression $II^q \quad V^q \quad I^q \quad I^q$ produced by the rules (1) $I^t \rightarrow V^{t/2} \quad I^{t/2}$, (2) $I^t \rightarrow I^{t/2} \quad I^{t/2}$ and (3) $V^t \rightarrow II^{t/2} \quad V^{t/2}$. A CYK-style parse tree would look like the following.

4	I^w			
3				
2	V^h	I^h	I^h	
1	II^q	V^q	I^q	I^q
	1	2	3	4

Notice that the removal of the terminal/nonterminal distinction now means that there is no need for a 0-row. This representation can be used to derive the portion of the alphabet relevant to the string and the spans over which each symbol should have α and β computed. We can derive the following combinations for which α and β would need to be computed:

i	j	Symbols
1	2	V^h
2	3	I^h
3	4	I^h
1	4	I^w

For all other symbols and spans, α and β would be zero. However, we also need to be able to determine which rules produced each cell in order to recursively compute α and β probability values. Unfortunately, the standard CYK representation does not tell us which rules were applied at each point, since, under the oracle model, we don't know anything about the abstract rules in the grammar and can only ask about rule instances. Rather than checking through the rule set again to re-derive which rules were applied to which cells, a better solution is to simply label the cells with rule instances during the parsing process. The table now becomes:

4	(1) $I^w \rightarrow V^h\ I^h$			
3				
2	(3) $V^h \rightarrow II^q\ V^q$	(1) $I^h \rightarrow V^q\ I^q$	(2) $I^h \rightarrow I^q\ I^q$	
1	II^q	V^q	I^q	I^q
	1	2	3	4

The operation of deriving rule instances for a subsequence is delegated to an oracle. The input to the oracle is a sequence of symbols, and the output from it is a set of rule instances with right-hand-sides matching the input sequence. While we present a concrete algorithm that implements this task for PCFGs and the PTGGs used in this paper (all of which have simple parameters and easily reversible rule functions), the definitions given later on achieve greater generality through the use of a traditional 'black box' oracle for which no particular algorithm is assumed.

For PCFGs, looking up rule instances for a symbol sequence is just a matter of checking symbol membership in the strings appearing on the right-hand side of rules. For PTGGs, the operation is a little more complicated due to the parameterization of the symbols. For symbols $x_i^{p_i}...x_j^{p_j}$ to be accounted for by an instance of rule function $y^t \rightarrow f(t)$, there must be some parameter value, p, for which $f(p) = x_i^{p_i}...x_j^{p_j}$. Algorithm 1 shows the process used to implement the oracle's operation for PTGGs.

Algorithm 1. *Given a sequence of parameterized symbols, $x_1^{p_i}...x_n^{p_n}$:*
$backtrackPTGG(x_1^{p_i}...x_n^{p_n}) =$

```
(1) Let $Y = \{y^t \rightarrow v_1^{f_1(t)}...v_n^{f_n(t)},\ |\ v_i = x_i\}$ be the collection of rule functions with matching right-hand side symbols for the input sequence.
(2) Let $Y' = \emptyset$ be a set of rule instances.
(3) For each rule function, $y^t \rightarrow v_1^{f_1(t)}...v_n^{f_n(t)} \in Y$:
    (a) Let $p' = f_i^{-1}(p_i)$ be the unique parent parameter, if one exists, such that $v_i^{f_i(p')} = x_i^{p_i}$.
    (b) If $p'$ exists, then add the instance $y^{p'} \rightarrow x_1^{p_1}...x_n^{p_n}$ to $Y'$.
(4) Return $Y'$
```

5.5 Modifications to the inside–outside algorithm

The changes required to support the oracle model and CYK parse table described in the previous sections cause subtle modifications throughout the inside–outside algorithm's equations. Changes must be made where the equations for α and β make reference to the rule set, R, since the rule functions are inaccessible and only rule instances can be accessed through an oracle. However, these changes are fairly small, substituting some reference to the rule set's oracle for the rule set itself. For an oracle, O, we will use the notation $P(O, i, j)$ to refer to the cell in the oracle-assisted parse table that can produce symbols from i through j. This cell will contain a list of rule instances. Using this notation, α would be redefined as follows:

$$\alpha(A, i, i) = \begin{cases} 1.0 & \text{if } A = X_i \\ \sum\limits_{A\rightarrow B \in P(O,i,i)} [\psi(A \rightarrow B) \times \alpha(B, i, i)] & \text{otherwise} \end{cases} \tag{10}$$

$$\alpha(A, i, j) = \sum_{k,l,\ A\rightarrow BCD \in P(O,i,j)}^{i \le k < l < j} [\psi(A \rightarrow BCD) \times \alpha(B, i, k) \times \alpha(C, k+1, l) \times \alpha(D, l+1, j)] + \sum_{k,\ A\rightarrow BC \in P(O,i,j)}^{i \le k < j} [\psi(A \rightarrow BC) \times \alpha(B, i, k) \times \alpha(C, k+1, j)] \tag{11}$$

Equations for β and μ would be modified similarly, replacing instances of R with $P(O, i, j)$.

5.6 Identity rules

Identity rules, or productions of the form $A \rightarrow A$, are an important feature in PTGGs for allowing variable length sequences and creating a diversity of durations in output sequences. While this is another feature that could be delegated to the generative algorithm, doing so would risk introducing problems like those discussed in Section 5.2. Therefore, we consider the impact of handling identity rules here.

So far, productions of the form $A \to B$ have been allowed, but with the requirement that $A \neq B$. Allowing for rules of the form $A \to A$ is somewhat problematic for various statistical reasons. Consider the following, terminal-less PCFGs:

Grammar 1		Grammar 2	
Rule	Probability	Rule	Probability
$A \to AA$	0.4	$A \to AA$	0.5
$A \to A$	0.2	$A \to B$	0.5
$A \to B$	0.4		

Both of these can generate strings defined by the regular expression $[A|B]^*$, which is all strings consisting of As and Bs. However, for a fixed number of generative steps, the probability of generating the string BB is lower for the first grammar than for the second. For Grammar 1, the probability of generating BB is at most $0.4^3 = 0.064$ from one application of $A \to AA$, two applications of $A \to B$, and no applications of $A \to A$. For Grammar 2, the probability of generating BB is exactly $0.5^3 = 0.125$.

From this, it is clear that the two distributions above can exhibit different behaviour even under the same generative algorithm. For the same number of total productions, grammar 1 is likely to produce a shorter string than grammar 2. However, what if the generative algorithm and number of productions are unknown? Distinguishing between the two distributions becomes tricky since each string producible by grammar 1 can be produced by grammar 2 with fewer steps.

To fully accommodate the features of PTGGs, we extend the inside–outside algorithm once more with a *heuristic* for counting occurrences of identity rules. It must be emphasized that this heuristic will not suit all generative algorithms since knowing where identity rules are likely to be applied requires knowing how the generative algorithm works. Unlike other rules, identity rules cannot simply be inserted into the parse tree wherever possible, since they can be applied infinitely many times in sequence while achieving the same overall result. Identity rules also cannot be ignored; some probability mass should be assigned to them if they are present in the grammar.

As a middle ground, we count one identity rule per parse tree in the μ calculation. In Equations 5–8, the requirement that $A \neq B$ is simply removed. However, importantly, we do *not* count identity rules in the α and β calculations. Doing so would risk placing disproportionate weight on identity rules when re-estimating production probabilities for a PCFG. By factoring identity rules into the μ calculation as shown above but nowhere else, it avoids assigning truly excessive weight to identity rules. Unfortunately though, this does not guarantee a conservative guess for the probability of identity rules, and the estimated probability can still end up being disproportionately high. Consider the following grammar.

Probability	Rule
0.3	$A \to AA$
0.3	$A \to A$
0.4	$A \to B$
0.5	$B \to B$
0.5	$B \to C$

It takes a minimum of two productions to generate a string containing at least one C. If a data-set is generated with very few iterations of an L-System-like algorithm, such as only three iterations, attempting to learn the probability for the $B \to B$ using the heuristic described above will fail by assigning a disproportionately large probability to the rule. This is because the restricted number of productions makes it unlikely that C will be encountered relative to B, and each instance of B will constitute one count for applying $B \to B$. This is a difficult problem to avoid with 'deep' grammars that require many productions to reach certain symbols. Symbols closer to the top of the symbol hierarchy are more likely to have accurate probabilities learned for their identity rules.

5.7 Computational complexity

For a PCFG, the complexity of this oracle-based version is actually unchanged, since the oracle would have a worst case runtime of $O(l)$, where l is the number of rules in the grammar, and will perform the same search as would have to be done in an oracle-less model anyway. Therefore, for PCFGs, the complexity is still $O(n^3)$ for grammars with rules of rank 2 and $O(n^4)$ for rules of rank 3, where n is the length of of the string to be parsed. Since the oracle model requires that $f^{-1}(X)$ be computable for rules of the form $X \to f(X)$ but does not bound the complexity of that computation, it is possible that the oracle's complexity could overtake that of the CYK-parsing for other categories of grammars.

For PTGGs, however, the complexity is much more variable, being strongly impacted by the specific functions that appear in the grammar and the particular implementation of the oracle for that grammar. The backtracking algorithm implementation for the PTGGs used our experiments requires checking whether the durations in a rule instance are possible to produce by a given grammar—if they are, then a unique parent parameter exists. For duration-preserving PTGGs, this unique parent parameter is the sum of all child parameters. For PTGGs that produce simple metrical structures over short sections of music, one way to check for valid combinations of child parameters is to simply enumerate the number of possible valid durations and check for membership. For example, if we know that a PTGG can only produce durations of a 1.0, 0.5, 0.25 and 0.125 (analogous to whole, half, quarter and eighth notes), then it is very straightforward to simply check against these four values. If two symbols sum together to produce a duration of 0.5, which is valid—but if they sum to produce $0.25 + 0.125 = 0.375$, there will be no valid rule instance. The exact set of values will be controlled by the format of the rules, the overall duration of the progression, d_{total}, and the smallest duration present in the progression, d_{min}. For example, for rules that only divide duration evenly, there will be $log_2(d_{total}/d_{min}) + 1$ possible durations. For such PTGGs, the number of enumerated values to check will grow much more slowly than the length of the input sequences, which is bounded by (d_{total}/d_{min}). Therefore, the complexity for parsing in this case will still be dominated by $O(n^3)$ from the CYK algorithm, where n is the number of symbols in the input sequence.

While the experiments in later sections only use PTGGs of rank 2, other PTGGs used for generating music use higher ranking rules, such as those for using PTGGs' more programming language-like features. PTGGs also permit let-in expressions for variable instantiation as described in Section 2.4. Although we do not explore the details of implementing a specific oracle for let statements in this paper, they are possible to accommodate within the same framework by creating oracles of higher complexity that check terminals within a particular nonterminal's subtree in addition to the nonterminals themselves. The complexity of such an oracle would then depend on both the number or rules and the length of the input sequence.

6. Training on Bach Chorales

Bach chorales offer a source of abundant and stylistically consistent musical examples that are also relatively easy to analyse to the level of Roman numerals. Because of this, it is a good data source for testing the performance of grammars like Rohrmeier's. Here, we present the results of such an experiment. A corpus of Bach chorales was analysed to the Roman numeral level and to the chord function level, and short phrases were extracted as training data. A modified version of Rohrmeier's grammar for harmony (Rohrmeier, 2011) and a simpler grammar over chord function symbols were used as the candidate grammars to parse the data and learn production probabilities. Finally, Kulitta's generative framework was used to create short, chorale-styled phrases according to the learned production probabilities.

6.1 Data-set

A collection of Bach chorales analysed by White (2013b) were taken as the starting data. The chorales in this data-set had already been analysed to determine the key of each chord. Further processing was then done to assign a Roman numeral to each chord, which was fairly straightforward since most of the chords were simple triads (e.g. $\langle 0, 4, 7\rangle$ in C-major is a I-chord). Phrases were taken as four-bar sections if all within the same key, and shorter same-key segments for four-bar sections that changed key. To eliminate noise resulting from key change boundaries, phrases containing three or fewer chords were removed. The remaining phrases were then divided by mode, creating major and minor training data-sets containing 824 and 455 phrases, respectively.

6.2 Musical PCFGs

Two musical PCFGs were tested using the Bach corpus: a modification of Rohrmeier's CFG and a more simplified CFG over the three-symbol alphabet of chord function symbols. These grammars are shown in Tables 1 and 2, respectively.

Rohrmeier's grammar for harmony is a CFG suitable for learning with the inside–outside algorithm. A reduced version of it that lacked modulations was used to parse the Bach corpus. However, not all of the data were parsable with this reduced grammar. Some of this may have been due to noise in the Roman numeral assignment, but other instances were likely due to limitations of Rohrmeier's grammar. To parse a larger subset of the data while minimizing redundancy in the rules, a modified version of Rohrmeier's grammar was used as the candidate grammar for learning production probabilities. This grammar is shown in Table 1.

Important changes to the grammar include the introduction of the C nonterminal to mean 'plagal cadence'. The 'phrase' level in Rohrmeier's grammar added redundancy and was largely irrelevant to parsing such small sections of music. Similarly, 'piece' as a nonterminal was not needed either. Instead, TR, or 'tonic region' was used as the start symbol. Repetitions at the Roman numeral level were also removed to avoid overly ambiguous parses.

6.3 Method

Our extended version of the inside–outside algorithm was run on the major and minor data-sets, taking samples of 200 phrases for each of several independent runs. Samples were taken uniformly at random from each data-set using a pseudorandom number generator and a fixed seed to ensure reproducibility of the samples. A total of five random samples were taken (seeds 0 through 4). The inside–outside algorithm was run until the change in probability mass fell below a specified threshold. This threshold was set to be 1% of the total probability mass, or 0.07 (since there were 7 nonterminals and the probabilities for each nonterminal must sum to 1.0).

6.4 Results

Averages of five runs of the learning algorithm with each CFG for the major and minor data-sets are shown in Figures 1 and 2. In each case, the different runs converged to nearly the same values.

The grammar from Table 1 demonstrates an interesting problem: the progressions in the data-set were all at least

Table 1. A modified version of Rohrmeier's grammar for harmony, where TR is the start symbol.

1	TR	→	T	8	SR	→	S	15	D	→	VII
2	TR	→	TR TR	9	SR	→	SR SR	16	D	→	V
3	TR	→	DR T	10	T	→	VI	17	S	→	IV
4	TR	→	TR DR	11	T	→	III	18	S	→	II
5	DR	→	DR DR	12	T	→	I	19	S	→	IV III IV
6	DR	→	D	13	T	→	I II VI	20	C	→	IV I
7	DR	→	SR D	14	T	→	T C	21	C	→	IV C

Table 2. A further simplification of the grammar in Table 1 that uses only a three-symbol alphabet of chord function symbols. Unlike the grammar in Table 1, this grammar features identity rules, or self-productions.

1	*T*	→	*T*	4	*T*	→	*T D*	7	*D*	→	*S D*
2	*T*	→	*T T*	5	*D*	→	*D*	8	*S*	→	*S*
3	*T*	→	*D T*	6	*D*	→	*D D*	9	*S*	→	*S S*

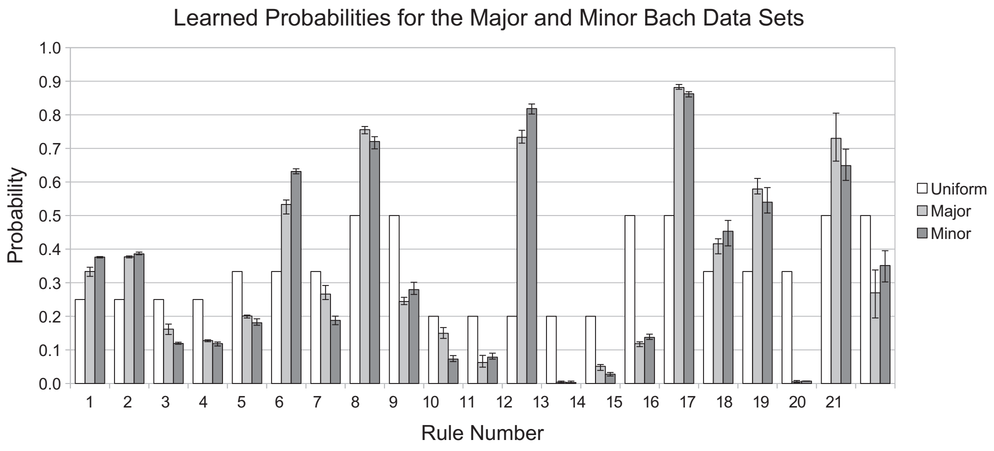

Fig. 1. Averages of learned production probabilities using the grammar from Table 1 and five runs of the learning algorithm. Error bars indicate the lowest and highest learned values for a particular rule. The 'uniform' data series represents uniform production probabilities over the rule set.

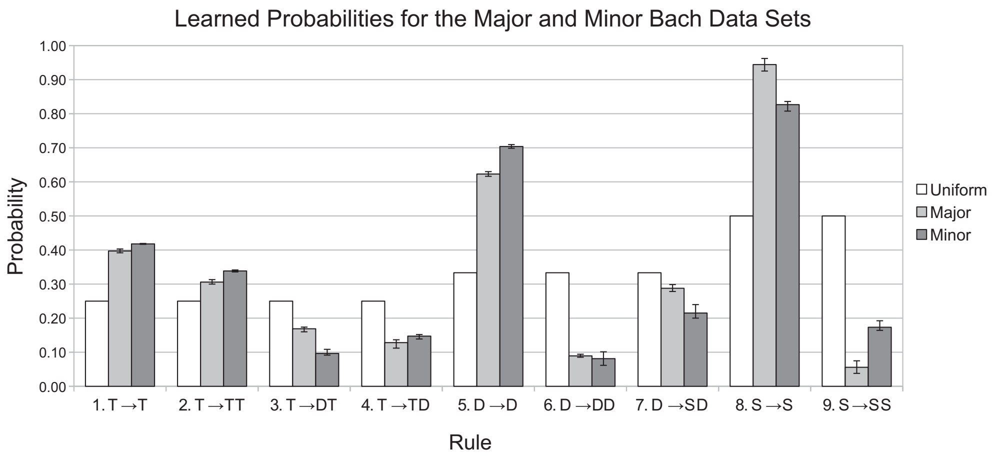

Fig. 2. Averages of learned production probabilities using the grammar from Table 2 and five runs of the learning algorithm. Error bars indicate the lowest and highest learned values for a particular rule. The 'uniform' data series represents uniform production probabilities over the rule set.

four chords long, but the start symbol, TR, has a substantial probability of generating a single chord progression, 0.33 for major and 0.38 for minor. This is due to the rule $TR \rightarrow T$, which turns a tonic region into a tonic chord. This is clearly not representative of the input data, which had no single chord progressions. Although perhaps less severe, the same issue would be present in Rohrmeier's original grammar as well, with the probability of generating a only a single chord being equal to that of the production series $Piece \rightarrow P \rightarrow TR \rightarrow T$. The only correction to this issue would be a generative algorithm that accounts for constraints such as progression length.

The grammar in Table 2, on the other hand, is not subject to the same progression-length problem. Because there are no terminals, it behaves like an L-System and the distribution over progression lengths becomes a function of the number of generative steps. This aspect of the grammar makes it easy to convert into a duration-preserving PTGG for generating new musical phrases where rules have the forms $A^t \rightarrow B^t$ and $A^t \rightarrow B^{t/2}C^{t/2}$. Major and minor probability distributions for the grammar were then supplied to Kulitta to generate four-measure-long phrases in three steps: (1) applying Kulitta's generative algorithms for PTGGs, (2) stochastically expanding chord functions to Roman numerals using a chord space derived from the Bach data-set and (3) Kulitta's foreground algorithm for chorales in the style of J.S. Bach. Example output is shown in Figures 3 and 4.

7. Training on synthetic data

We have extended the inside–outside algorithm with an oracle, which allows for learning production probabilities of PTGGs. However, construction of a PTGG that will reliably parse arbitrary human-made music[5] is still a subject of ongoing work. Thus, it is currently not possible to test PTGGs on such corpora with the key-finding and chord labelling algorithms used in the previous section. A detailed explanation of these problems is given in Section 7.3. Briefly, the root of the problem is that PTGGs typically have strict divisions of duration, a feature that is not necessarily the case for automated key/chord analyses. This results in PTGGs being underaccepting to a degree that makes analysis of any large corpus of human-made music problematic. To be parsable by a PTGG, a substantial human-made corpus would either need to be analysed in a way that simplifies rhythmic and harmonic irregularities or the parsing process would need to be somewhat error-tolerant. Automated key and chord detection algorithms such as the one used in the previous section are prone to adding noise that both would not be present in hand analyses and can easily create parsing problems with PTGGs—although even some very clean manual parses of Bach chorales can be problematic due to anacruses as described later on.

In the absence of using a human-made corpus, testing on synthetic data is still important since it serves as a way to demonstrate the correctness of the learning algorithm. Synthetic data provide a way to directly compare probabilities used for generation of the data-set (which do not exist for human-made music such as the Bach corpus) with those derived from the learning process.

To verify that our PTGG-related extensions to the inside–outside algorithm behave as expected, we created a synthetic data-set using the rules shown in Table 3. A total of 1000 phrases were created using the PTGG rules and probabilities shown in Table 3 using Kulitta's generative algorithms. Phrases were created using four generative iterations over the progressions, a starting duration of four bars in 4/4, and a minimum duration of a quarter note.

7.1 Methods

The learning algorithm was run using samples of 200 phrases taken from the total set of 1000. Learning was considered complete when the change in probability mass from one iteration to another fell below 1% of the total (0.04, since the rules in Table 3 have a total mass of 4). Phrases were selected uniformly at random using a random number seed. Learning was repeated with five different samples from random number seeds 0 through 4. Random samples of 200 progressions were taken for each of five runs using different random number seeds. Examples of synthetic progressions used in this experiment are shown in Table 4.

7.2 Results

The learned production probabilities are shown in Figure 5. All runs of the learning algorithm converged to the same result rapidly, requiring only a few iterations. This and the identical nature of the results is due to the small rule set and the fact that the candidate grammar was, in fact, the exact grammar used to produce the data-set—thus being completely free of noise. The learned probabilities differ slightly from the probabilities shown in Table 3, but are nevertheless quite close.

Musically, when evaluated via Kulitta, the results of this test are more dissonant than those from the Bach corpus with PCFG to PTGG conversion. This is partly because the PTGG used for this experiment was designed primarily to test the performance of the learning algorithm rather than to model a particular type of music well. Clearly it would be preferable to perform learning on a corpus of real music rather than synthetic data in order to obtain results that match a particular style.

While the results of this experiment show that learning of production probabilities is certainly feasible for a PTGG, we also note that it is possible to observe anomalies in the learned probabilities. Three situations can produce skewed, but not

[5]By 'human-made', we mean music written in a traditional manner, such as many classical Western scores, where the exact models and process used to create the music is not yet known. This is in contrast to algorithmically generated music where the process is well-defined. For example, the output from Kulitta could easily be parsed using the same PTGG used during generation.

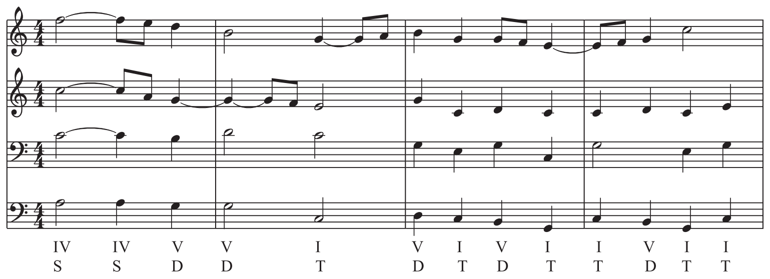

Fig. 3. A chorale-styled phrase produced by Kulitta using the grammar from Table 2 and the major production probabilities shown in Figure 2.

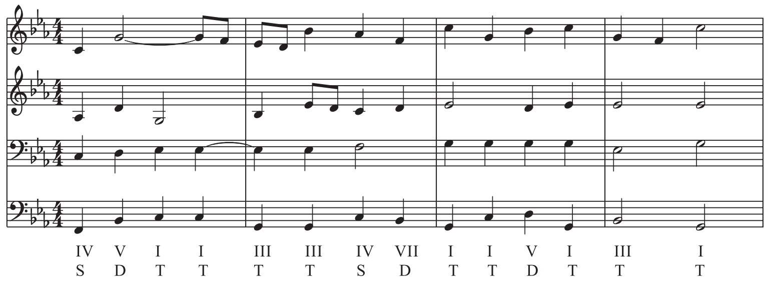
IV V I I III III IV VII I I V I III I
S D T T T T S D T T D T T T

Fig. 4. A chorale-styled phrase produced by Kulitta using the grammar from Table 2 and the minor production probabilities shown in Figure 2.

Table 3. A small PTGG that exhibits conditional behaviour based on the duration of the input chord, where w is a whole note and h is a half note. The probabilities shown were used to generate the synthetic training data-set.

Num.	Probability	Rule
1	0.7	$I^t \rightarrow V^{t/2}\ I^{t/2}$
2	0.3	$I^t \rightarrow$ **if** $t \leq 2w$ **then** $IV^{t/2} I^{t/2}$ **else** $IV^{t/4}\ V^{t/4}\ I^{t/2}$
3	0.7	$III^t \rightarrow III^{t/2}\ VI^{t/2}$
4	0.3	$III^t \rightarrow I^{t/2}\ III^{t/2}$
5	0.6	$IV^t \rightarrow$ **if** $t \leq h$ **then** $II^{t/2}\ VI^{t/2}$ **else** $IV^{t/4}\ III^{t/4}\ IV^{t/2}$
6	0.4	$IV^t \rightarrow V^{t/2}\ IV^{t/2}$
7	0.2	$V^t \rightarrow IV^{t/2}\ V^{t/2}$
8	0.3	$V^t \rightarrow II^{t/2}\ V^{t/2}$
9	0.4	$V^t \rightarrow III^t$
10	0.1	$V^t \rightarrow VII^t$

Table 4. The first five sequences in the synthetic data-set generated for the purpose of testing the PTGG from Table 3. The letters h, q, e and s denote durations of half (0.5), quarter (0.25), eighth (0.125) and sixteenth (0.0625) notes, respectively.

1.	$III^s\ VI^s\ III^s\ IV^s\ III^e\ VI^e\ III^e\ IV^e\ V^e\ I^e$
2.	$II^q\ III^q\ III^e\ IV^e\ IV^e\ I^e$
3.	$VII^h\ IV^e\ V^e\ V^e\ I^e$
4.	$II^q\ III^q\ II^e\ V^e\ V^e\ I^e$
5.	$II^q\ III^q\ III^q\ V^e\ I^e$

Table 5. A PTGG extension to the grammar from Table 2 that can be used to parse phrases from Bach chorales. This grammar allows parsing of patterns such as $T\ S\ T$, which appear in some harmonic analyses (Czarnecki, 2013), but are not permitted by the grammar in Table 2.

1	T^t	$\rightarrow$	$T^{t/2}\ T^{t/2}$	5	T^t	$\rightarrow$	$T^{t/2}\ S^{t/2}$	9	D^t	$\rightarrow$	$S^{t/2}\ D^{t/2}$
2	T^t	$\rightarrow$	$D^{t/2}\ T^{t/2}$	6	T^t	$\rightarrow$	T^t	10	D^t	$\rightarrow$	D^t
3	T^t	$\rightarrow$	$T^{t/2}\ D^{t/2}$	7	D^t	$\rightarrow$	$D^{t/2}\ D^{t/2}$	11	S^t	$\rightarrow$	$S^{t/2}\ S^{t/2}$
4	T^t	$\rightarrow$	$S^{t/2}\ T^{t/2}$	8	D^t	$\rightarrow$	$D^{t/2}\ S^{t/2}$	12	S^t	$\rightarrow$	S^t

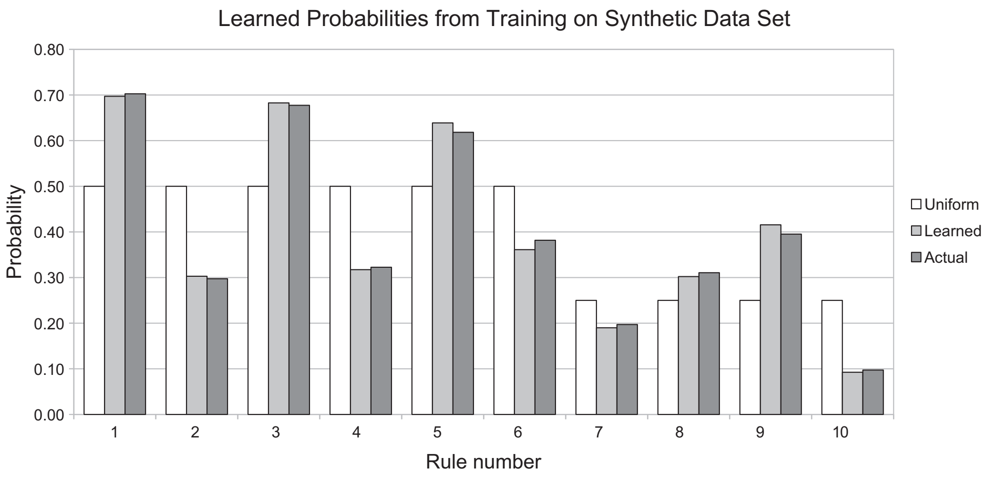

Fig. 5. Averages of learned production probabilities using the grammar from Table 3 and five runs of the learning algorithm. The 'actual' data series represents proportions of actual rule applications during the generation of the data-set. This measure was used rather than the production probabilities to eliminate deviations from the production probabilities that may have been introduced by the generative algorithm. Error bars are not present in this graph since the algorithm converged to the same values each time. The 'uniform' data series represent uniform production probabilities over the rule set.

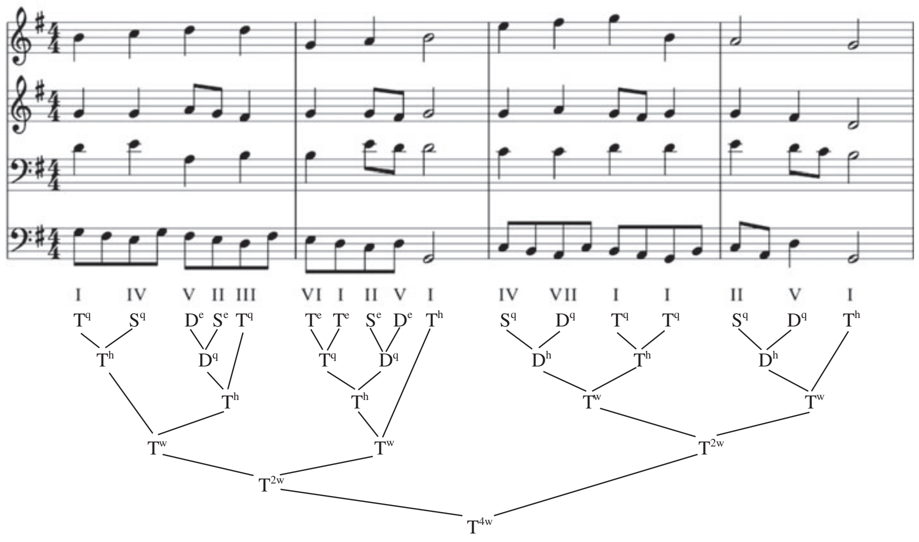
I IV V II III VI I II V I IV VII I I II V I
T^q S^q D^e S^e T^q T^e T^e S^e D^e T^h S^q D^q T^q T^q S^q D^q T^h
T^h D^q T^q D^q D^h T^h D^h
T^h T^h T^w T^w
T^w T^w T^{2w}
T^{2w}
T^{4w}

Fig. 6. A parse of the first phrase of BWV 115.6 using the Roman numeral analysis from Czarnecki 2013, a simplification to the chord function level, and the PTGG from Table 5. For visual simplicity, identity rules are not shown in the parse tree, although they would be necessary to recreate this structure using a generative algorithm such as Kulitta's.

necessarily incorrect probabilities as observed in earlier experiments involving learning production probabilities for PTGGs from a synthetic corpus (Quick, 2014):

(1) A combination of identity rules, $X^t \rightarrow X^t$, and rules conditional on duration with an identity rule component, $X^t \rightarrow$ **if** $t < d$ **then** X^t **else** z, where z is a sequence of two or more chords. If durations of d or less are reached too quickly during generation, probability mass may be disproportionately taken from conditional rules and given to the identity rules.

(2) An inadequate number of generative iterations and a 'deep' grammar. If many generative iterations are required to produce a symbol, its production rules may not appear in representative numbers in synthetically produced data.

(3) The same distribution of data is possible to produce with different PTGG production probabilities using a small number of generative iterations. A common way this situation arises is when there are two grammars for the same language and one has identity rules while the other does not, such as the example presented in Section 5.6.

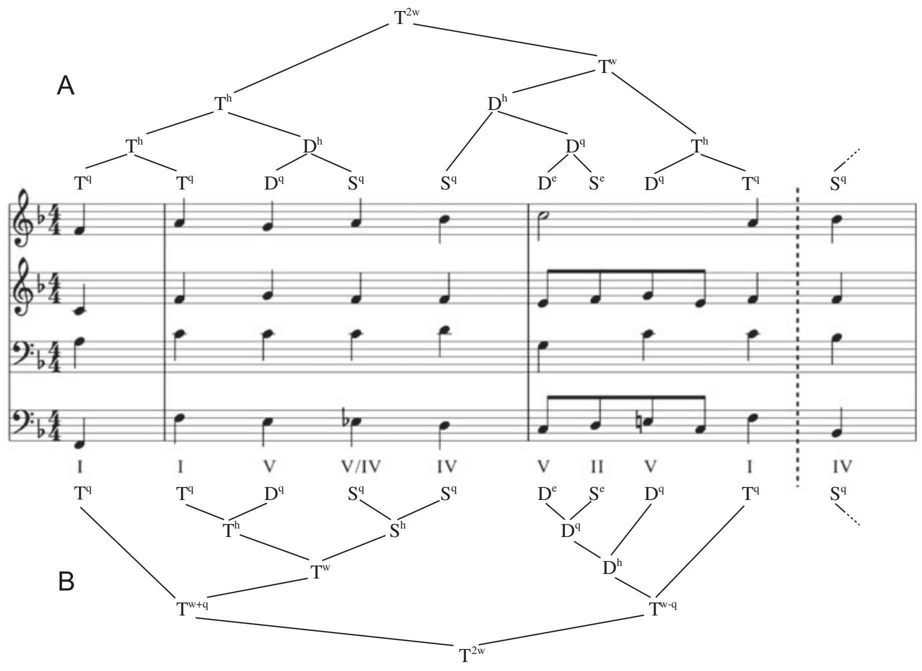

Fig. 7. Two possible parses of BWV 281 using the Roman numeral analysis from Czarnecki 2013 and a simplification to the chord function level. As in Figure 6, identity rules are not shown in the parse tree for simplicity. Durations w, h and e represent a whole note, half note and eighth note, respectively. Parse A uses the PTGG from table 5 directly, but the pickup causes productions to span metrical boundaries in an unnatural way. Preserving the metrical boundaries along with the pickup-based structure of the phrases in this chorales may be better accomplished with a hypothetical PTGG that allows un-even distributions of duration.

Any of the features above can easily cause skewed learned production probabilities. The first two cases simply require that the PTGG be constructed with careful attention to the length of the phrases and the durations used in conditional rules. The third case is more tricky since it may not be obvious when the situation occurs. In all cases, the learned production probabilities would not actually be wrong—they just deviate from those supplied to the generative algorithm to build the data-set. Instead, the differences are actually due to assuming a slightly different generative algorithm during learning than was actually used to produce the data.

7.3 Parsing human-made music

PTGGs attempt to handle harmonic and metrical information simultaneously to allow for more intelligent decision-making during the generative process. Even with a very simple parsing algorithm similar to those used by L-Systems, a PTGG can capture enough structural information to create harmonic and metrical structures that would otherwise require special consideration in the generative algorithm. In fact, some phrases from Bach chorales are possible to parse with a very simple PTGG. The slightly larger rule set shown in Table 5 is able to parse phrases such as those shown in Figures 6 and 7. In these cases, the harmonic analysis to the Roman numeral level was done by hand by Christopher Czarnecki Czarnecki (2013) rather than performed by an algorithm. Czarnecki's analyses contain a number of T S T and similar transitions that require the addition of some extra rules beyond the grammar in Table 2.

Figure 6 is an example of a phrase that parses fairly well, with metrical relationships that are sensible relative to the time signature and note durations in the music. However, Figure 7 is an example of a problematic case. Many phrases in Bach's chorales feature an anacrusis, which causes the phrase to start before the beginning of the first full measure and often end with an incompletely filled measure. If presented with just a sequence of chord symbols tagged with durations, the grammar from Table 5 is actually able to parse the first 16 beats of the music—but not in a terribly meaningful way. Metrical units spanning measure boundaries seem a poor fit for the structure of the music and directly violate metrical models such as those proposed by GTTM (Lerdahl & Jackendoff, 1996) and David Temperley Temperley (2010). Similarly, it would be somewhat incorrect to ignore the anacrusis and treat the first four bars in isolation. A more reasonable hypothetical approach is shown in parse B of Figure 7, adding and subtracting durations in a way that accounts for the relative alignment of harmonic and metrical features in the music. Support for features like anacruses within PTGGs is a subject of ongoing work. In combination with better algorithms for automatically labelling chords, we hope to be able to parse the Bach chorale corpus used in the previous section with PTGGs in the future.

8. Conclusion

Our work demonstrates an important merging of analytical and generative work in the area of musical grammars. We have shown that it is possible to learn the weights for multiple categories of probabilistic grammars, including those derived from the CFG for harmony proposed by Rorhmeier as well the more complex category of PTGGs. Utilizing an oracle to perform certain operations in the inside–outside algorithm greatly broadens the category of grammars that the inside–outside algorithm can address well beyond typical CFGs. This broadening also does not necessarily come at the expense of complexity. Although there would clearly be categories of grammars for which an oracle would be quite complicated and thus drive up the complexity of the learning process, the oracle approach also allows for tailoring to a particular grammar, which means that the complexity can substantially less for a particular grammar than might be the upper bound for a larger category of grammars. The amount of ambiguity in the grammar is also a big factor in the overall runtime: the less ambiguity there is in a grammar, the more it will avoid the worst-case runtime for its category.

We have made several important observations using generation as a way to qualitatively assess learning of musical grammars:

(1) Much in line with existing work such as GTTM (Lerdahl & Jackendoff, 1996), music's temporal nature is important to consider when modelling harmony, both in analytical and generative settings. This is possible to do with PTGGs, which feature rhythmic consideration as a part of the specification within the rules rather than as a separate process.
(2) Learning production probabilities for a grammar is only as useful as the grammar is well-suited to representing the data-set. Selection of a candidate rule set remains as a separate task, but there are many such models proposed in music theory that are suitable for testing the types of experiments presented here.
(3) When the above are appropriately addressed, our extension of the inside–outside algorithm in conjunction with Kulitta represents a useful method for testing candidate grammars generatively rather than analytically alone. This allows for the possibility more diverse kinds of model evaluation (addressed in more detail in Section 8.2).

The various phrases generated after training show a clear need for a temporal element in the grammar and also a need for ways to control the number of produced chords—or, rather, a way to avoid being immediately funnelled into a single-chord series of productions. PTGGs like those in Table 3 can do this well, but parsing real music with them is tricky. Small amounts of noise in the analysis could easily render a phrase unparsable by a PTGG if the durations of chords were affected, whereas a more standard, non-temporal PCFG is able to handle these types of deviations with the inclusion of a few extra rules. Currently, this actually makes PCFG to PTGG conversion a somewhat more robust approach when trying to learn features from human-made music. However, learning a PTGG directly could be equally robust if some degree of error tolerance was built into the parsing process or if the corpus was analysed in a way that simplifies problematic structures.

The selection of the candidate grammar for a set of training data is incredibly important. Unfortunately, it is quite easy to construct a trivial grammar for a data-set that successfully parses all of it yet captures no aspects of the input data. Consider a grammar for English with only one nonterminal, W, which is also the start symbol, and two types of rules: $W \to W\ W$ and $W \to w$, where w is any word in an English dictionary. This will successfully parse any English sentence while capturing none of the structure of the language. We observe a very mild case of this with the modified version of Rohrmeier's grammar (shown in Table 1) when applied to a corpus of Bach phrases that are filtered by length: the production probability for the production $TR \to T$ becomes somewhat meaningless since it implies that there is some probability of generating a one-chord phrase—something that exists nowhere in the input data. In this respect, the simplified grammar over $\{T, S, D\}$ (shown in Table 2) has the potential to model the data better, since length of the progression is a function of the number of generative iterations used.

8.1 Future work

While our work shows promising results, there is also ample room for improvement. Perhaps the most notable limitation lies with PTGGs since they currently cannot be used to easily parse large quantities of human-made music. Methods for parsing human-made music with PTGGs are a subject of our ongoing research. Two main possibilities exist for accomplishing this:

(1) The addition of a post-processing step or the use of a more rhythmically aware harmonic analysis algorithm with the goal of producing a cleaner, more metrically simplified analysis that is more in-line with hand-analyses performed by human experts. Quantization-like processes such as those used in rhythmic analysis (Raphael, 2001; Taylan & Peter, 2000; Temperley, 2009) may be useful in achieving this.
(2) The construction of an error-tolerant or 'fuzzy' parsing algorithm that is able to match a PTGG rule against sequences of chords that are near but not exact matches for the durations in the rule. This type of feature would also help to correctly interpret anacruses.

While hand-analyses that are noise-free could be manually encoded into a suitable format for the learning algorithm described in this paper, doing so is also highly inefficient for large data-sets. Ideally we would like to have a more automated method to obtain suitable analyses from digital formats such as MIDI, which may be possible by some combination of the tactics described above.

Parsing efficiency for grammars with more complex parameters where the oracle's backtracking complexity exceeds that of the CYK algorithm may also benefit from using Opatrny and Culik's approach for EOL languages (Opatrny & Culik II, 1976). Although similar to the CYK algorithm, Opatrny and Culik's leverages knowledge of the generative algorithm to minimize fruitless backtracking by keeping track of the number of generative steps. This may also help in obtaining more accurate estimates for identity rule production probabilities.

Testing our system on additional musical corpora would be a useful step to determine how many aspects of a style of music can be captured within a musical grammar over abstract harmonic alphabets. Modelling additional styles of music will likely also require testing different alphabets, such as those proposed by Winograd (1968), which address a greater level of musical detail than plain Roman numerals.

8.2 Performance metrics

Further empirical work can be done to evaluate the performance of our algorithm with candidate grammars, both in analytical and generative settings. The approaches described here would benefit from additional experimentation in the following areas:

(1) Statistical evaluations of generated sequences to determine their similarity to the input data.
(2) Structural comparisons of parse trees formed by different models.
(3) Evaluation of generated output with participant studies.

Some statistical comparisons are also relevant to Markov chains would be worthwhile here. Specifically, are local distributions of chords the same in newly generated sequences as they are in the output? This would be an important indicator as to whether the musical grammars used are appropriately capturing small-scale structures. However, comparisons on a larger scale would also be required to check for longer term coherency. One possibility for this is structural comparison of parse trees against those produced by music theorists.

Information theoretic approaches to assessing statistical models focus on evaluating likelihood: How probable is it that the model would generate the observed data-set? A model that fits the data better will have a higher likelihood of having produced that data. Although CYK parsing does not provide information on specific parses, it does provides the overall likelihood of generating the sequence (which will factor in more than one parse in the case of ambiguous sequences). Likelihood-based comparisons are, therefore, important for determining how well-suited a grammar is to a particular data-set. Samer Abdallah et al. and Rens Bod use several likelihood-based methods for evaluation of statistical musical models (Abdallah et al., 2016; Bod, 2002). Their experiments include measures such as entropy, simplicity, compression and variational-free energy. These types of measures allow for comparison of multiple grammars over the same data-set, helping to illustrate which model best captures the data. If sufficient simplification of input data-sets can be achieved and/or if parsing error tolerance can be added to PTGGs, then the performance of this new category of music could be more directly compared to other grammars in the literature using similar methods.

Finally, while statistical and information theoretic approaches to evaluating algorithm performance are useful for automated comparisons of different musical models, ultimately it is human perception that is the gold standard by which newly created music is judged to be reasonable or not. We feel that this is an important area of consideration when evaluating statistical models for music, and a strong reason for testing models generatively. Weaknesses in models can become quite obvious in a generative setting when evaluated by the human ear. For example, a low-order Markov chain that captures local patterns in music will likely generate new sequences with obvious long-term coherency problems due to the model's inability to capture such structures in music. Some preliminary participant studies examining perception of new chorale-style phrases generated by Kulitta against those of J.S. Bach show that music generated with PTGGs can be convincingly human-like (Quick, 2014). The ability to train Kulitta's probabilistic models on different data-sets permits testing of different candidate grammars with similar participant-based studies. This type of participant-based experimentation could serve as an additional performance metric for evaluation of statistical models through generation.

Acknowledgements

Thanks to Dana Angluin and Ian Quinn for providing support, guidance and expertise to aid in the development of the work presented in this paper.

References

Abdallah, S., Gold, N., & Marsden, A. (2016). Analysing symbolic music with probabilistic grammars. In D. Meredith (Ed.), *Computational Music Analysis* (pp. 157–189). Cham: Springer International Publishing.

Alty, J. L. (1995). Navigating through compositional space: The creativity corridor. *Leonardo, 28*(3), 215–219.

Bellgard, M. I. & Tsang, C. (1994). Harmonizing music the Boltzmann way. *Connection Science, 6*, 281–297.

Bellgard, M. I., & Tsang, C. (1996). On the use of an effective Boltzmann machine for musical style recognition and harmonization. In *Proceedings of the international computer music conference* (pp. 461–464), Hong Kong.

Bod, R. (2002). *A general parsing model for music and language*. In *Music and artificial intelligence: Second international conference, ICMAI 2002 Edinburgh, Scotland, UK, 12–14 September 2002 Proceedings* (pp. 5–17). *Berlin, Heidelberg*: Springer, Berlin Heidelberg.

Brown, S., Martinez, M. J., & Parsons, L. M. (2006). Music and language side by side in the brain: A PET study of the generation of melodies and sentences. *European Journal of Neuroscience, 23*, 2791–2803.

Bühlmann, P. & Wyner, A. J. (1999). Varable length Markov chains. *The Annals of Statistics, 27*, 480–513.

Callender, C., Quinn, I., & Tymoczko, D. (2008). *Generalized voice-leading spaces. Science Magazine, 320*, 346–348.

Chordia, P., Sastry, A., & Senturk, S. (2011). Predictive tabla modeling using variable-length Markov and hidden Markov models. *Journal of New Music Research, 40*, 105–118.

Clark, A. (2010). Efficient, correct, unsupervised learning of context-sensitive languages. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning* (pp. 28–37), Uppsala, Sweeden.

Conklin, D. (2006). Melodic analysis with segment classes. *Machine Learning, 65*, 349–360.

Cope, D. (1987). An expert system for computer-assisted composition. *Computer Music Journal, 11*, 30–46.

Cope, D. (1992). On the algorithmic representation of musical style. In *Understanding music with AI* (pp. 354–363). Cambridge, MA: MIT Press.

Czarnecki, C. (2013). *J. S. Bach 413 Chorales analyzed: A study of the harmony of J. S. Bach*. SeeZar Publications.

De Haas, W. B., Rohrmeier, M., Veltkamp, R. C., & Wiering, F. (2009). Modeling harmonic similarity using a generative grammar of tonal harmony. In *Proceedings of the 10th international society for music information retrieval conference* (pp. 549-554). Kobe, Japan.

Gales, M. & Young, S. (2007). The application of hidden Markov models in speech recognition. *Foundations and Trends in Signal Process., 1*, 195–304.

Gillick, J., Tang, K., & Keller, R. M. (2009). Learning jazz grammars. In *Proceedings of the sound and music computing conference* (pp. 125–130), Porto, Purtugal.

Gogins, M. (2006). Score generation in voice-leading and chord spaces. In *Proceedings of the international computer music conference* (pp. 593–600), New Orleans, LA.

Granroth-Wilding, M., & Steedman, M. (2012). Statistical parsing for harmonic analysis of jazz chord sequences. In *Proceedings of the International Computer Music Conference* (pp. 478–485), Ljubljana, Slovenia.

Hamanaka, M., Hirata, K., & Tojo, S. (2006). Implementing "A generative theory of tonal music". *Journal of New Music Research, 35*, 249–277.

Hörnel, D. (2004). CHORDNET: Learning and producing voice leading with neural networks and dynamic programming. *Journal of New Music Research, 33*, 387–397.

Johnson, M. J. & Willsky, A. S. (2013). Bayesian Nonparametric Hidden Semi-Markov Models. *Journal of Machine Learning Research, 14*, 673–701.

Juang, B. H. & Rabiner, L. R. (1991). Hidden Markov models for speech recognition. *Technometrics, 33*, 251–272.

Keller, R. M., & Morrison, D. R. (2007). A grammatical approach to automatic improvisation. In *The proceedings of the Sound and music computing conference* (pp. 330–337), Lefkada, Greece.

Kirlin, P. B., & Utgoff, P. E. (2008). A framework for automated Schenkerian analysis. In *The proceedings of the International conference on music information and retrieval* (pp. 363–368), Philadelphia, PA.

Lari, K. & Young, S. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language, 4*, 35–56.

Lerdahl, F., & Jackendoff, R. S. (1996). *A generative theory of tonal music*. Cambridge, MA: The MIT Press.

McCormack, J. (1996). Grammar based music composition. *Complex Systems, 96*, 320–336.

Opatrny, J., & Culik, K., II (1976). Time complexity of recognition and parsing of EOL-languages. In A. Lindenmayer & G. Rozenberg (Eds.), *Automata, languages, development: At the crossroads of biology, mathematics and computer science*. Amsterdam: North-Holland Publishing Company.

Pachet, F. (2002). The continuator: Musical interaction with style. In *Proceedings of the international computer music conference* (pp. 211–218), Gothenburg, Sweden.

Pachet, F. & Roy, P. (2011). Markov constraints: steerable generation of markov sequences. *Constraints, 16*, 148–172.

Ponsford, D., Wiggins, G., & Mellish, C. (1999). Statistical learning of harmonic movement. *Journal of New Music Research, 28*, 150–177.

Prusinkiewicz, P., & Lindenmayer, A. (1990). *The algorithmic beauty of plants*. New York, NY: Springer.

Quick, D. (2014). Kulitta: A framework for automated music composition. New Haven, CT: Yale University.

Quick, D. (2015). Composing with kulitta. In *Proceedings of international computer music conference* (pp. 306–309), Denton, TX.

Quick, D., & Hudak, P. (2013a). Grammar-based automated music composition in Haskell. In *Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling, and design* (pp. 59–70), Boston, MA.

Quick, D., & Hudak, P. (2013b). A temporal generative graph grammar for harmonic and metrical structure. In *Proceedings of the international computer music conference*, Perth, Australia.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77*, 257–286.

Raphael, C. (2001). A hybrid graphical model for rhythmic parsing. In *Proceedings of 17th conference on uncertainty in artificial intelligence*. Morgan Kaufmann.

Raphael, C. & Stoddard, J. (2004). Functional harmonic analysis using probabilistic models. *Computer Music Journal, 28*, 45–52.

Rohrmeier, M. (2011). Towards a generative syntax of tonal harmony. *Journal of Mathematics and Music, 5*, 35–53.

Ron, D., Singer, Y., & Tishby, S. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning, 25*, 117–149.

Roy, P., & Pachet, F. (2013). Enforcing meter in finite-length markov sequences. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence* (pp. 854–861), Bellevue, WA.

Schenker, H. (1954). *Harmony*. Chicago, IL: University of Chicago Press, OCLC 280916.

Smith, N. A. & Johnson, M. (2007). Weighted and probabilistic context-free grammars are equally expressive. *Computational Linguistics, 33*, 477–491.

Smoliar, S. W. (1979). A computer aid for Schenkerian analysis. In *Proceedings of the 1979 annual ACM conference* (pp. 110–115), New York, NY.

Taylan, A. & Peter, C. (2000). Rhythm quantization for transcription. *Computer Music Journal, 24*, 60–76.

Temperley, D. (2009). A unified probabilistic model for polyphonic music analysis. *Journal of New Musi Research, 38*, 2–18.

Temperley, D. (2010). Modeling common-practice rhythm. *Music Perception, 27*, 335–376.

Tymoczko, D. (2006). The geometry of musical chords. *Science Magazine, 313*, 72–74.

White, C. C. (1991). A survey of solution techniques for the partially observed Markov decision process. *Annals of Operations Research, 32*, 215–230.

White, C. (2013a). An alphabet-reduction algorithm for chordal *n*-grams. In *Proceedings of the international conference on mathematics and computation in music* (pp. 201–212).

White, C. W. (2013b). *Some statistical properties of tonality, 1650–1900*. New Haven, CT: Yale University.

Whorley, R. P., Wiggins, G. A., Rodes, C., & Pearce, M. T. (2013). Multiple viewpoint systems: Time complexity and the construction of domains for complex musical viewpoints in the harmonization problem. *Journal of New Music Research, 42*, 237–266.

Winograd, T. (1968). Linguistics and the computer analysis of tonal harmony. *Journal of Music Theory, 12*, 2–49.

Worth, P., & Stepney, S. (2005). Growing music: Musical interpretations of L-systems. *Applications on Evolutionary Computing*, 535–540.

Yi, L., & Goldsmith, J. (2007). *Automatic generation of four-part harmony. UAI applications workshop*.

Yust, J. (2009). The geometry of melodic, harmonic, and metrical hierarchy. In *Proceedings of the Mathematics and computation in music second international conference* (Vol. 38, pp. 180–192), New Haven, CT.