Statistical learning of harmonic movement

Dan Ponsford[*] Geraint Wiggins[†] Chris Mellish[‡]
Department of Artificial Intelligence
University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN

Abstract

We explore the application of statistical techniques, borrowed from natural language processing, to music. A probabilistic method is used to capture and generalise from the local harmonic movement of a corpus of seventeenth-century dance music. The probabilistic grammars so generated are then used for experiments in generation (composition).

The corpus is preprocessed in a novel way, automatically converting the harmonies into a normal form to capture the underlying harmonic similarities between pieces. It is then automatically marked up with constituent boundaries (beginnings and ends of pieces, phrases and bars), to enable the learning process to capture some of the higher-level structure of the music.

The experiment is promising, and a sample of the results are given. We discuss the limitations of the approach, and how they might be overcome.

1 Introduction

This paper is about deriving grammars from musical data using statistical techniques. These grammars are then used for generation (composition).

Much work has been done on natural language processing (NLP) using probabilistic techniques, and parallels exist between natural language and music and between NLP and music processing. In this paper some ideas and techniques are taken from NLP and applied to music processing.

The aim is to develop a grammar-learning system which uses a corpus (collection) of music examples to learn significant characteristics of the music. The grammar that is learned can then be used to generate new music in the same style. It should be possible to specify the type, key and length of a piece and have the system compose a piece of music to that specification.

Like natural language, music may occur as sound or in written form. There are parallels too between the relative simplicity of text and notated music compared with the complexity of speech and acoustic music.

This work models written (notated) music. Notation simplifies and idealises the acoustic music it represents. It treats note lengths as having round-number values, notes of chords as being sounded precisely together, notes in melodies as being perfectly contiguous and non-overlapping, and leaves the exact timbre and pitch to the player. The present system makes most of these simplifications also.

The *desiderata* guiding this work are:

- extensibility to other corpora – we do not wish to produce an *ad hoc* system suited only to this data

[*]Dan Ponsford no longer works at Edinburgh. He may be reached *via* the other authors.
[†]Email: `geraint@ed.ac.uk` (correspondence should be sent to this address in the first instance)
[‡]Email: `chrism@dai.ed.ac.uk`

- minimal use of music knowledge – again, we wish to be general and in particular to test the extent to which a linguistic technique can be applied to music
- fidelity to compositional method where music knowledge is used – in those areas where domain-specific knowledge is required, we require it to be well-founded and justifiable

Specifically, the aim of this research was to capture the *harmonic movement* of a particular style of tonal music, in order to determine whether the automated methods used were adequate to such a task. For our purposes here, harmonic movement is defined as the beat-by-beat progression of chord structures found (explicitly or otherwise) in a piece of music, and we include the cadence as a particular formalisation of harmonic movement.

The aim, then, is to be able to write down a set of grammar rules which can be used to generate (*i.e.*, compose) music in the selected style, so that the results can be judged by comparison with the original data. The reader can see these results for him- or herself in Section 12.

The rest of the paper is structured as follows. Section 2 presents some of the background issues in statistical approaches to language learning. Section 3 describes grammar-learning techniques in some detail. Section 4 relates these techniques to existing work in music processing. Section 5 describes the corpus chosen for the present work, and section 6 describes the design of the grammar-learning system. Sections 7–8 give details of our experiments in generation and present the results. Finally, some conclusions are drawn and further work is identified. Several examples of the generated output are given in Section 12.

2 Background

2.1 Grammar learning

Grammar learning (also *grammar induction* or *grammar inference*) is the process of deriving a grammar (set of rules) from examples of a language. If the examples are adequate to represent the language as a whole, and the grammar-learning procedure is suitable, then the resulting grammar will be a grammar of the language. In practice, the set of training examples used to find the grammar is usually incomplete. A system that aims at wide coverage must allow for this.

Grammar learning may be motivated by a wish to imitate human grammar learning (cognitive modelling) or it may just be a practical way of building an evaluator (system for assigning probabilities to strings) or a generator.

2.2 Language type

If a language consists of a set of symbols and a set of rules for combining those symbols, then music is a language. This is the view taken here. Rules may also be termed *regularities* or *patterns.* It is regularities and patterns that make music appealing. For instance, without repetition of some feature such as rhythm or melody, music becomes difficult to listen to.

Music, like many spoken languages, such as English, is really a set of languages. A musical style may be regarded as a language; so may a musical form. For present purposes, the music of each composer is a separate language. Any language with well-defined rules can be modelled by a computer program.

Chomsky classified languages into four types according to the power of the rules needed to describe them. The two types that have received most attention in computing and natural language processing are regular (finite-state) and context-free languages. These types have rules that are more constrained (and less powerful) than the other types, context-sensitive and general rewrite. In this work, an attempt is made to model music using a finite-state grammar (type 3 in the Chomsky hierarchy), computationally the simplest type. See Hopcroft and Ullman (1969) for more details.

The languages modelled here are languages of individual composers and more broadly the language of a school of composers – in other words, a style, applied to a particular musical form.

2.3 Evaluation and generation

Grammars have two main uses – recognition and generation. Recognition means determining whether a string belongs to a language or not. Generation involves creating new strings.

Probabilistic grammars may be designed in such a way that they will recognise *any* string, but assign low probabilities to strings that are unlikely to be in the language. In this case, it is inappropriate to speak of recognition. Instead the term *evaluation* is used to mean assigning a probability to a string.

Much work has been done in NLP on generation of natural language text. To generate text, you need to have some meaning to convey, and that meaning must be represented in some unambiguous way. When generating music, strings can be generated using syntactic rules alone, as there is no meaning, as such, to convey. In this paper we concentrate on generation (also referred to in the literature as *prediction*).

2.4 Constituents, Phrases and Strings

For our purposes here, a musical phrase is a sequence that reaches a point that gives a sense of pause or arrival. This point is called a *cadence.* A natural length for phrases is four bars (measures). In general, however, phrases can be of any length.

A phrase may consist of subphrases, and these may be analysed further. However, musical phrases are not recursive in the way that natural-language phrases are.

Phrase is closely related to meter. For simplicity, it is assumed here that phrase boundaries coincide with beginnings and ends of bars.

2.5 Terminology

Some terminology is defined here that will be useful later on.

alphabet the set of symbols of a language, *e.g.,* the set of words in a corpus of text, or the set of harmonies in a harmonic corpus

annotation the addition of meta-symbols to strings in order to provide information about them that is not already explicitly present

corpus a set of strings of some language

dissonance here, anything that is not a triad or a seventh

feature features (also *dimensions*) are characteristics distinguishing between the symbols of a language, such as pitch and note-length (duration) in music

harmony vs chord a harmony is an abstract set of notes whose precise pitch and arrangement are not specified; a chord is a set of notes whose pitch and arrangement is specified

learning here, the process of finding regularities and storing information about them

parameter a probability. *Parameter estimation* is the process of calculating probabilities

string a sequence of symbols, especially one that is regarded as in some way complete (such as a natural-language sentence or a complete piece of music)

subcorpus part of a corpus, especially a part that is homogeneous in some way

training the process of learning regularities from examples

3 Statistical grammar learning

Statistical grammar-learning techniques find patterns in data and the frequency with which they occur. These patterns, together with their frequencies, may be regarded as rules and can be used for generation or evaluation of strings. The key ingredients in statistical learning are a *corpus* of data (*i.e.,* a collection of strings), a pattern matcher and a formula for finding probabilities.

The main advantage of statistical techniques over purely symbolic techniques is that statistical rules are ranked according to frequency, whereas with symbolic rules, the only way of saying that rule A is more important than rule B is to include A and exclude B.

The corpus may be created for the purpose of training, or it may already be available, having been created for some other purpose. It is preferable that the corpus be already available, since creating a corpus may involve a lot of work. On the other hand, ‘naturally occurring’ data usually needs some editing to get it into a suitable form. (In NLP, large amounts of text exist and are available for use as training corpora. In the present case, the corpus was transcribed from existing music specifically for the present application.)

Techniques involving statistics have given some useful results in NLP and pattern recognition. In the area of part-of-speech tagging, for instance, statistical taggers are generally accepted as being more successful than non-statistical ones.

The major statistical techniques that have been used in NLP and are n-grams (a class of Markov models), hidden Markov models and probabilistic context-free grammars.

3.1 N-grams

An n-gram is a sequence of symbols of length n. The first $n - 1$ of these are the context. The context is denoted here by $\bar{c}$.

There are two stages involved in using n-grams: parameter estimation, where probabilities are assigned to n-length sequences found in a corpus, based on their frequency and the frequency of their context $\bar{c}$; and using these parameters, for either evaluation or generation. Figure 1 shows a portion of text and one trigram (3-gram) and its context, $\bar{c}$, which is two symbols in length.

$$\ldots\quad a\quad t\quad e\quad x\quad t\quad u\quad a\quad \overbrace{\underbrace{l\quad e}_{\bar{c}}\quad x}^{\text{trigram}}\quad a\quad m\quad p\quad l\quad e\quad \ldots$$

Figure 1: A trigram in some text

n-grams are a very simple language model – the simplest in use. They are based on the idea that selection of the next symbol in a string depends on the preceding symbols in the string. However, instead of taking account of *all* the preceding symbols, which could be computationally very costly and would require vast amounts of data, the next symbol is assumed to depend only on the previous $n - 1$ symbols.

The simplicity of n-grams is appealing – they correspond with a simple kind of finite state model of the language, and they are easily implemented. However, to use them in their pure form is to treat the language as being adequately characterised by n-symbol sequences – a very limited kind of language. The view here, and by most users of n-grams, is that this is just a useful approximation of the way the language works.

The size of n can in theory be anything from 1 upwards. However, certain values are better than others at capturing the characteristics of the language. The larger the value of n, the more context is captured. If n is 1, then there is no context, and the probability of seeing a particular symbol is dependent only on its individual relative frequency in the corpus. In real languages, context is important in determining what symbol comes next. So an n of 2 is better, because here the previous symbol affects what comes next. An n of 3 is better still, as now two previous symbols constrain what comes next. However, there is a cost to having a larger n, as there are more parameters (n-length sequences whose frequency needs to be stored).

If there are 30 distinct symbols in the corpus, then there are 30 possible unigram sequences; $30 \times 30 = 900$ bigram sequences; and $30 \times 30 \times 30 = 27,000$ trigram sequences. Going up to 4-grams, the figure is 810,000. In NLP, where alphabets (sets of words) are large, this exponential growth means that 3 is normally the largest value for n that is computationally possible.

While it would be useful to have n a bit larger than 3, is it not a case of the larger the better. As n grows, it captures more context. Eventually the sequences learned become not just characteristic of the corpus, but the exact sequences in the corpus. If n were too large, then in generation, actual portions of the corpus would be reproduced, instead of some novel string.

What n-grams capture of a language is *taxis* – the way symbols fit together locally. What they do not capture is larger structure and long-distance dependencies. Music has all three things: *taxis* (transitions between notes or harmonies), structure (key structure and phrasing) and long-distance dependencies in the form of repeated themes and motifs.

3.1.1 Parameter estimation

An n-gram *parameter* of a language is the probability of a symbol w_n occurring, given that the previous symbols $w_{i,n-1}$ have just been found. Parameters are estimated (calculated) from a corpus. (Parameter estimation is also referred to as *training.*) A language whose alphabet is A has $|A|^n$ possible n-gram parameters. In reality, however, only a subset of the possible parameters occur. This is because of the rules of *taxis* of the language which exclude certain orderings.

The parameter associated with the final symbol of each n-length sequence $w_{i,n}$ is found as follows. Divide $w_{i,n}$ into the first $n-1$ symbols, $w_{i,n-1}$, and the nth symbol, w_n. The probability is then the number of times $w_{i,n}$ occurs in the corpus divided by the number of times $w_{i,n-1}$ occurs.

$$P(w_n|w_{i,n-1}) = \frac{\mathrm{Freq}(w_{i,n-1}, w_n)}{\mathrm{Freq}(w_{i,n-1})} \tag{1}$$

The probability of a string that has been generated is a measure of how likely the string is to be found in the language.

According to the n-gram model, the probability of a string is the product of the probabilities of each symbol given the $n-1$ symbols preceding it. For trigrams this is

$$P(w_{1,l}) = \prod_{i=1}^{l} P(w_i|w_{i-2}, w_{i-1}) \tag{2}$$

where l is the length of the string. (In fact, at $w_{i=1}$ there is no context, so it is necessary to add $n-1$ dummy symbols to the start of the string to provide context.)

3.1.2 Random generation

Random generation is used to create new strings. To do this, the parameters learned from the corpus are used.

Given a starting context of $n - 1$ dummy symbols, the string is generated by finding all the possible next symbols, randomly generating a number, and translating this number into a selection from the list of possible next symbols, which are weighted according to their share of the probability mass (probabilities sum to 1).

The last $n - 1$ symbols in the sequence generated so far, $\overline{c}$, are used in determining which the possible next symbols are. For trigrams, the length of $\overline{c}$ is 2. Any symbol that occurred in the corpus, preceded by $\overline{c}$, is a possible next symbol.

3.2 Smoothed n-grams

Where an n-length sequence is absent from the corpus, its probability is zero, and the product given above, for any string as part of which it appears, (equation 2) will also be rendered zero. This is undesirable, because it makes the unrealistic assumption that all possible n-length sequences were seen in training. It would be better for all strings to have non-zero probabilities, provided that they use symbols of the language. To achieve this, n-gram probabilities can be mixed (interpolated) with (n–1)-gram probabilities. And where these are also absent, (n–2)-gram probabilities, and so on, until unigram (1-gram) probabilities are used.

This is done by assigning weights to sequences of length n down to 1, and summing the weighted probabilities over each of these lengths to get the probability of an n-length sequence. The probability of a given n-length sequence must be between 0 and 1. So the weights sum to one, as do the probabilities. Larger context is preferred, so longer sequences are assigned higher weights than shorter ones. By convention, the weights are denoted by λ. For instance, interpolated trigram probabilities are calculated as the sum over weighted trigrams, bigrams and unigrams.

$$\lambda_3 P(w_n | w_{n-2}, w_{n-1}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_1 p(w_n) \tag{3}$$

Now every n-length sequence will be assigned a non-zero probability, provided that the nth symbol occurred in training (*i.e.,* the unigram probability is non-zero).

The process of altering probabilities to allow for data that was absent in training is called *smoothing.* Interpolated n-grams, one method for smoothing, are discussed in Jelinek and Mercer (1980).

3.3 Hidden Markov models

The following definition of a hidden Markov model (HMM) is from Church and Mercer (1993).

> A Markov model is a finite state machine with probabilities governing transitions between states and controlling the emission of output symbols. If the sequence of state transitions cannot be determined when the sequence of outputs is known, the Markov model is said to be *hidden.* In practice, the Forward-Backward algorithm is often used to estimate the values of the transition parameters on the basis of corpus evidence.

The forward-backward (or Baum-Welch) algorithm mentioned here is presented in Baum (1972).

Charniak (1993) introduces the idea of a hidden Markov model by discussing the need for smoothing (interpolation) with trigrams (see section 3.2 above). He points out that it would be useful for the weights for different sizes of n (represented by λ in the description above) to be calculated automatically. Hidden Markov models are able to do this.

HMMs have been used widely in speech recognition and other pattern recognition applications. They are more powerful than n-gram based methods; we choose n-grams here by way of Ockham's razor.

3.4 Probabilistic context-free grammars

HMMs are equivalent to statistical regular grammars. Further up the Chomsky hierarchy are context-free grammars. Probabilistic context-free grammars (PCFGs), also called stochastic context-free grammars (SCFGs) have been used in NLP, and the inside-outside algorithm (Lari and Young, 1990) exists for inferring them from data.

It is appropriate to use PCFGs where the language worked on is itself context-free. The present corpus is not context-free. A key feature of context-free languages is recursion: symbol A rewrites as other symbols, and somewhere down the line, one of these other symbols, or their children, rewrites as A. This does not occur in the harmonic progressions or phrase structures modelled here, so PCFGs would be unnecessarily powerful.

3.5 Corpora

A corpus is used in probabilistic learning to find patterns. For patterns to be found the corpus needs two properties: it must be large enough and the patterns in it must be repeated. The first requirement is fulfilled by including a large number of items (here, pieces of music). The second requirement is fulfilled by choosing similar items – pieces of a similar style, possibly pieces by the same composer and of the same type.

Ideally, a corpus used for n-gram estimation should be large enough to contain all possible n-grams. This is very unlikely to happen, though. One test for whether the corpus had reached such a 'saturation point' is to hold out some strings, estimate n-gram parameters on the rest and then see if the held out strings are evaluated with probabilities that are similar to those of the strings the parameters were actually trained on. For a discussion of *held-out estimation*, see Jelinek and Mercer (1985).

For use with n-grams, the minimum corpus size will depend on a) the size of n, b) the size of the alphabet, and c) the stringency of the tactical rules of the language. Section 10 suggests one piece of music knowledge that could be used for testing the adequacy of a corpus. A rule of thumb for corpus size is that a corpus can never be too big!

For discussion of issues in corpus linguistics, see Manning and Schütze (1999), Krenn and Samuelsson (1996) and McEnery and Wilson (1996).

4 Previous work in statistical music processing

This section presents existing work in the area of statistical music processing. Techniques borrowed from statistical NLP are introduced in detail in Section 6.

Several researchers have used probabilistic methods, notably Markov models, to model music. Usually it is melody that is modelled. Barbaud (1965, page 106) used digrams to model harmony. Among those who have used Markov models are Pinkerton (1956), Brooks Jr. *et al.* (1993) and Conklin and Witten (1995). Pinkerton used Markov models to generate nursery rhymes. He used a small corpus of diatonic major-key nursery rhymes. He augmented his seven-symbol alphabet (one symbol for each note of the diatonic scale) with a symbol representing a tied note. He simplified some of the rhythms in his corpus.

Because he used a small alphabet, Pinkerton was able to use a high-order Markov model (one using a comparatively large amount of context). He experimented with orders up to eight. Beginnings and ends of pieces were not marked, with the result that some pieces he generated began with tied notes.

A more recent Markov model experiment was done by Brooks Jr. *et al.* (1993). Like Conklin and Witten, below, they worked with chorale melodies, and like Pinkerton, they experimented with orders up to eight. Their corpus was of 37 hymn tunes (giving perhaps 5,000 note transitions). Again, to capture similarities between pieces in different keys (but the same mode), all pieces were transposed into C. The experiment showed that at very low orders (*e.g.,* unigram), strings generated do not resemble strings in the corpus recognisably, while at very high orders, strings from the corpus are just replicated.

Random generation was used to generate strings (as in the present work – see Section 6). Strings that are generated but do not meet certain constraints are rejected. For instance, strings that did not end on the desired note or contained rhythms that conflicted with common-time meter were rejected.

Conklin and Witten (1995) used trigrams to generate chorale melodies from parameters estimated on a corpus of Bach chorale melodies. They made the point that highly predictive theories will produce reasonable estimates of a musical language, and will generate original, acceptable works.

Ames (1989) describes his compositional method using Markov models and surveys Markov composition up to 1989. Two important distinctions emerge from Ames's discussion: first, between approaches where transition probabilities (see Section 3 for a technical explanation) are decided on by the researcher and approaches where the probabilities are derived from a set of musical examples; and second, between approaches where the output of the model is taken as being a composition in itself, and those where it is treated as material for human composition.

Ames's own approach was to set the transition probabilities himself and to 'cut and paste' the output into a composition that is to his liking, which makes the results difficult to analyse. By contrast, in the present work, transition probabilities are derived objectively from a corpus of example pieces of music, and the output from generation is treated as a finished product.

Another well-known approach to stylistic replication is that of Cope (1991). Cope uses a grammatical generation system based on a "style dictionary". In particular, the system (EMI) uses what Cope calls "signatures", which are melodic micro-gestures common to individual composers. By identifying and reusing such signatures, Cope is able to reproduce the style of past composers in reportedly impressive ways.

With respect to the current approach, while there is an apparent surface similarity between Cope's signatures and n-grams (as applied to individual instrumental parts – recall that we consider *harmony*, in the abstract, here), it is not clear whether Cope's system uses a related kind of statistical analysis. Generation in EMI is by means of Augmented Transition Networks, which are a significantly more powerful mechanism than that required for the grammars produced by the n-gram technique. As such, we may suppose that even if a genuinely statistical analysis is carried out in EMI, it is combined with other forms of processing. So it is still useful to examine the potential of a given NLP technique (*viz.,* n-grams) in isolation.

The current approach is novel in that we have abstracted out a significant amout of noise from our data, and focussed sharply on the *underlying* harmonic movement of a number of pieces in a given style, rather than the *implementation* of that style in a particular piece. We expect, on this basis, to achieve rather more abstract, general results than those achieved in the work outlined above. In particular, our focus is more on the suitability of the n-gram technique than on the results *per se*.

5 The corpus

5.1 Corpus choice

The corpus needed to be of a music that is sufficiently simple for it to be possible to model at least the most important characteristics within the short time available. At the same time, the music had to be of a type that was interesting, and not overly simple. It would be useful too if there existed a theory for the music.

Various kinds of seventeenth-century music were considered, and in particular dance forms. These tend to be relatively simple. They are strongly harmonic, and modelling harmony alone would generate slightly more satisfactory pieces of music than modelling melody alone. It is useful to to be able to limit the number of characteristics looked at. *Chaconne* and *passacaglia* – two kinds of dance – were considered, because as well as having the simplicity of other seventeenth-century music, they also have the property that their harmony consists of repeated four-bar sequences. However, it was decided that this music would be too simple. Instead, the *sarabande* was chosen. The *sarabande* is a dance in 3-time. It has a binary structure. Enough sarabandes are available in written form to make up a usable corpus.

Behind this music is a fair amount of theory, providing ideas about how to model the music.

The corpus consists of 84 seventeenth-century French *sarabandes.* The composers are shown in Table 1. Of the 84 pieces, 25 are by Chambonnières, and 19 by Louis Couperin. The rest are spread among the 13 other composers. The corpus is divided into subcorpora by composer and mode. The subcorpora are shown in Table 2 and discussed in Section 5.3.

The music of the corpus represents a school or style, and similarities exist between the music of the various composers, such as the series of keys through which the music passes. Most pieces are between 20 and 32 bars long, the commonest length being 24 bars.

Composer	Dates
Louis Couperin	1626–1661
Nicolas Lèbegue	1631–1702
Jean-Henri D'anglebert	?1635–1691
Jacques Champion de Chambonnières	1601–1672
Elisabeth Jacquet de la Guerre	1666/7–1729
François Couperin	1668–1733
Louis-Nicolas Clérambault	1676–1749
Jean-Baptiste Lully	1632–1687
Denis Gaultier	1603–1672
Germain Pinel	d. 1661
Estienne Richard	*c.* 1621–1669
Marin Marais	1656–1728
René Mézangeot	d. 1638
Louis Marchand	1669–1732
François Dagincour	1684–1758

Table 1: Composers represented in the corpus

5.2 Corpus entry

The corpus was created by recording the 84 pieces using a keyboard and EZVision (Macintosh music editing software). Using EZVision's editing facilities, wrong notes were corrected and beginnings of notes were rounded to the nearest semiquaver. Each piece was then then saved as a MIDI file (see Rothstein (1992) for a discussion of MIDI format).

5.3 Subcorpora

Nine subcorpora were extracted from the corpus and were used for separate experiments. Their characteristics are shown in Table 2.

	Subcorpus content	Bars	Beats
1	all major- and minor-key pieces	2000	6000
2	all major-key pieces	1052	3156
3	all minor-key pieces	948	2844
4	major-key pieces by Chambonnières	352	1056
5	minor-key pieces by Chambonnières	240	720
6	major-key pieces by L. Couperin	296	888
7	minor-key pieces by L. Couperin	144	432
8	major-key pieces not by Chambonnières or L. Couperin	404	1212
9	major-key pieces not by Chambonnières or L. Couperin	564	1692

Table 2: Subcorpora

Subcorpus 1 was used in only one experiment. For most of our experiments, subcorpora 2 and 3 were the largest. When compared to the million-word corpora used for NLP, these are tiny. However, the alphabet of symbols in the present corpus is correspondingly small.

Major and minor subcorpora are processed separately. For major-key subcorpora, the maximum size of the alphabet is 17 symbols (*i.e.*, 17 different harmonies that are possible, represented by degree of scale). For minor-key subcorpora the maximum is 20.[1] Using trigrams, this means a maximum of $17^3 = 4,913$ trigrams in the major mode and $20^3 = 8,000$ trigrams in the minor. These are maxima. The actual numbers of parameters found for major-key pieces is 882 and for minor-key pieces 1191. The difference is at least partly accounted for by rules of *taxis*. Similarly with 4-grams, there is a big difference between the number of possible and actual 4-grams. Possible numbers are 34,391 and 160,000 for major and minor respectively, while actual numbers of 4-grams recorded were 1,826 and 2,397.

All of these figures are tiny when compared with the number of possible trigrams for a piece of text using, say, a modest vocabulary of 2,000, where the maximum[2] would be $2,000^3 = 8 \times 10^9$.

6 The system

This section describes the implementation of the grammar-learning system. The n-gram technique was chosen as the grammar-learning technique – specifically, 3-grams (trigrams) and 4-grams. Preprocessing and annotation of the corpus are described, as are methods for random generation of strings.

[1]The figures of 17 and 20 are the numbers of harmonies in the keys in the tonic and the four most closely related keys for each mode. For major keys, this these are the diatonic harmonies of the tonic, supertonic, subdominant, dominant and submediant keys; and for major keys, the diatonic harmonies of the tonic, mediant, subdominant, dominant and subtonic (seventh) keys.

[2]A solution to the huge number of n-gram parameters for a natural-language corpus is to assign syntactic or semantic classes to words in the corpus and do parameter estimation at the level of classes. This was proposed by Brown *et al.* (1992).

6.1 Features modelled

The most important features of the corpus are harmony and key structure. Rameau (1722) wrote that “it appears that Melody arises from Harmony”, and “it is from harmony that the rules of melody must be derived”. Pieces of seventeenth-century music tended to be harmonic/metrical frameworks that needed some degree of completion (ornamentation or more) by the performer. A minimum of harmonies plus a bass line were supplied by the composer.

Therefore, harmony was chosen for modelling. Key structure (the progression through different keys), phrasing and meter are also captured by the model, due to the annotation of the corpus described in section 6.5.

The aim is to generate pieces consisting of sequences of harmonies. The particular texture and arrangement of notes are left unspecified.

A successful grammar would be one that generated realistic modulations (key changes), phrase and cadence structure and harmonic rhythm.

Major- and minor-key pieces are treated separately. The symbols of the corpus are harmonies. There are effectively two alphabets – one for the major key pieces (17 symbols) and one for the minor (20 symbols).

With harmony changing at least once every two beats, there are around 1,500 harmony transitions in the major-key subcorpus and around 1,400 in the minor-key subcorpus. It is these transitions that are learned by the system.

6.2 Rules of harmony

The system combines the use of musical knowledge in preprocessing and the statistical n-gram technique for learning harmony. Estimation of n-gram parameters, evaluation and most steps in generation are independent of language. That is to say, they are not specific to music input, but can be used for other corpora, such as natural-language text.

The following harmonic knowledge is defined in a logic program: root position, first inversion and second inversion triads; major, minor, diminished and augmented triads; seventh chords; incomplete triads; consonance and dissonance; intervals; inversion of chords; the set of harmonies for each mode; the set of harmonies for each related key in each mode. This knowledge is used in preprocessing.

6.3 Representation

Pieces in the corpus are in 12 different keys – 6 major and 6 minor. It is assumed here that all major keys behave alike and so do all minor keys. (In fact, this is known to be an approximation, but for present purposes it is a useful one.) So to capture the similarities, some representation needs to be used that is neutral with respect to key. To achieve this, two representations were considered. One transposes everything so that it is centred on the note C; the other dispenses with note names and represents notes by their degree of scale. Neither system makes explicit reference to mode (major and minor), and both have the same size of alphabet.

The second system was chosen because it is easier to think in neutral scale-degree terms rather than mentally transposing between keys. There is no difference from the point of view of the grammar-learning process.

6.3.1 Pitch

In the scale-degree representation, each pitch is represented as a pair of (scale degree, modifier). The range of values for each of these features is given in Figure 2.

scale degree $\in \{1,2,3,4,5,6,7\}$
modifier $\in$ {`common`, `major`, `minor`, `raised`, `lowered`}

Figure 2: Representation of pitch

`common` means that the pitch is the same for major and minor scales. Where the two scales differ, `major` and `minor` are used to differentiate. For pitches outside the major and minor scales, `raised` and `lowered` are used. `raised` means that the `common` pitch is raised by a semitone, and `lowered` means that it is lowered by a semitone.

Three other attributes are needed to represent a note: a unique identifier, a start (onset) time and a note-length (duration). Onset and duration are represented as integers. Sometimes it is convenient to translate integer durations into mnemonics, such as `quaver`.

6.3.2 Harmony

Harmony can be thought of in terms of precise pitches. Alternatively, it can be treated abstractly as sets of notes, whose precise pitch is not specified, and where there is no duplication of notes.

The latter is how harmony is treated here. By treating harmonies as sets, it is possible to find more similarities between one piece and another than if they are treated as bags, containing arbitrary duplication. Of course, there must be some basis for this, and there is: music theory and practice view harmonies as sets. For instance, the practice of playing from a figured bass, where each harmony is specified by a bass note and numbers abstractly indicating which other notes should be played, treats harmonies as ordered sets. The figures indicate what ordering should be placed on the set, modulo the octave.

Brooks Jr. *et al.* (1993) chose not to model harmony on the grounds that it was too complex. However, as this work shows, harmony can be modelled fairly simply if it is treated as one-dimensional and subject to normalisation (see section 6.4.5).

Harmonies are represented initially as lists of notes.

((2 `common`) (5 `common`) (7 `minor`))

This one is a second inversion minor triad on the 5th degree of the scale. In C, this would be as shown in Figure 3.

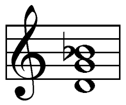


Figure 3: Example of chord representation

6.3.3 Pieces

Pieces of music (here, *sarabandes*) are represented as lists of harmonies.

6.4 Preprocessing

Preprocessing has two functions: conversion to Prolog, the programming language used; and conversion of individual notes to harmonies. The harmonies are then converted to a normal form, in order to reduce the number of different symbols in the corpus and capture commonalities. The steps involved are as follows:

- convert to Prolog
- convert pitches to scale-degree notation
- round note lengths
- sample harmony at quaver intervals
- convert harmonies to root position form (RPF)
- elide identical harmonies

Natural language texts are often preprocessed before being used for information extraction, grammar learning, etc. Preprocessing usually involves *tokenising*, where a text is broken into segments (tokens) – words and punctuation symbols. Tokens may be further segmented in a process called *lemmatising*. Here, unimportant differences between word tokens, such as verbal or plural suffixes, are removed in order to reveal the underlying similarity. Where differences exist that are unimportant, it is useful to remove them. This effectively reduces the size of the alphabet, and so reduces the amount of data needed.

Sampling harmony is equivalent to tokenising, where an input stream is segmented into discrete units, and converting harmonies to a normal form is equivalent to lemmatising.

Preprocessing algorithms are given at the end of this section.

6.4.1 Conversion to Prolog

The data starts off in MIDI format (Rothstein, 1992, for example). Notes are treated as *events*, consisting of a pitch, a start (onset) time and an end (offset) time. These three elements are represented as integers. The MIDI data is converted into a form that can be used by Prolog, using a C program written by Stephen Watkinson[3]. Each note event is assigned a unique identifier.

Once in Prolog format, the data consists of lists of terms of the form

```
event(NoteID, Pitch, OnsetTime, Duration)
```

for instance, a piece by Chambonnières is represented as

```
piece(  [event(cham1_0, 70, 0, 737),
         event(cham1_1, 69, 720, 252),
         event(cham1_2, 43, 0, 979),
         event(cham1_3, 55, 480, 518),
         ...
         event(cham1_143, 43, 26880, 526)] ).
```

This representation of note events as quadruples of (identifier, pitch, onset time, duration) conforms to the CHARM specification, suggested by Smaill *et al.* (1993).

[3]Stephen Watkinson is a PhD student at the Department of Computer Science, University of York.

6.4.2 Conversion to scale-degree notation

This step converts pitches from their MIDI representation as integers to a representation in terms of scale-degree. In this representation, notes are labelled relative to the tonic note. So if the tonic of the piece is C and the pitch to be converted is also a C, then the scale degree will be 1. If, however, a D from the same piece is converted, the scale degree will be 2.

As explained in section 6.3 above, the purpose of using a scale-degree representation is to remove key differences, in order to capture the commonalities between music in different keys of the same mode. In this simple case, the tonic of the scale used is the tonic of the whole piece, which is an input parameter.

6.4.3 Rounding note lengths

Note lengths are rounded to some rounding unit (quaver, semiquaver, etc.). The purpose of this is to ensure that notes which were held too long when the corpus was recorded are shortened to their intended length, so that they do not interfere with the following harmony. The algorithm is given in Figure 4.

In the present work, the rounding unit was chosen to be a semiquaver, because semiquaver is the shortest note value in the corpus.

```
let $r$ be the chosen rounding unit;
$h = r/2$;
for each note $n$ do
  $d = duration(n)$;
  $\delta = d \bmod r$;
  if $\delta \geq h$ then
    $rounded = d + (r - \delta)$;
  else
    $rounded = d - \delta$;
  end if
end for
```

Figure 4: Algorithm for rounding note lengths

6.4.4 Sampling harmony

Sampling harmony involves finding all the notes that are sounding during (all or part of) a specified time interval. The size of this time interval is called the sampling rate. The notes occurring during this interval are treated as forming a harmony. For present purposes, the sampling rate is chosen to be of quaver-length, because it is at the level of quavers that the quickest changes of harmony occur in the present corpus. The algorithm is given in Figure 14..

Sometimes the window allows in notes that do not belong to a harmony. This happens when there are short notes, such as passing notes. For the purposes of the current work, these are included in the harmony that is formed; they are not a big problem for the present corpus. However, section 10 suggests a way of treating the problem in a way that would cater for other corpora.

6.4.5 Conversion to root position form (RPF)

A triad is a three-note chord which satisfies one of the following two conditions:

1 the lowest note forms a 3rd (major or minor) with the middle note, and the middle note forms a 3rd (major or minor) with the highest note

2 a permutation of the chord satisfies condition 1

Chords which satisfy condition 1 are in root position. In music theory, root position is the canonical form for triads. Sequences of harmonies that are all in root position are in *root position form* (RPF). Harmonies not in root position form are converted to RPF, in the following way.

The first step in converting to RPF is to remove duplication of notes at the octave (*i.e.,* pitch classes), so that the harmonies become sets, rather than bags. An example is shown in Figure 5.

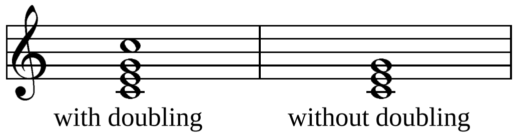


Figure 5: Removal of doubled notes

Other harmonies contain fewer than three notes. These incomplete triads are completed by adding notes (Figure 6). An incomplete triad can be completed in more than one way. So heuristics are used that give preference to certain completions over others. Completions that form root position triads, first inversion triads and second inversion triads are favoured in that order. This is based on the observation that the lowest note in a chord tends to be the root; less often the 3rd degree; and least often the 5th degree. (This holds for Western harmonic music generally.) The completion of a triad that is missing its 3rd degree depends on the mode of the piece. First, an attempt is made to complete to a member of the set of diatonic harmonies (those in the 'home' key of the piece). If this fails, the incomplete triad is completed to a member of the set of harmonies in all related keys.

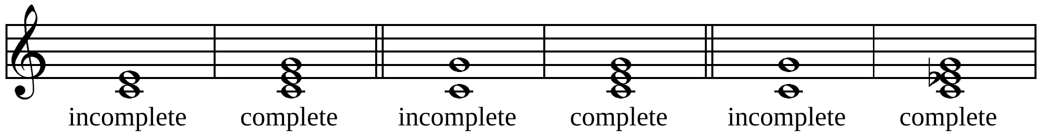


Figure 6: Completion of incomplete triads

Many harmonies contain more than three different notes. These, such as seventh and ninth chords, are reduced to triads by removing notes (Figure 7). A distinction is made between sevenths and other dissonances, for two reasons: first, because contemporaneous music theory does; and second, because sevenths may resolve onto dissonances, but dissonances must resolve to a consonance.

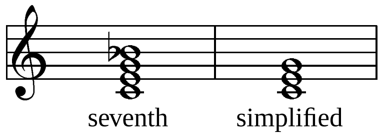


Figure 7: Simplification of sevenths

Finally, dissonant harmonies that resolve to consonant ones are resolved. Where a dissonant harmony A' can be made the same as the consonant harmony A which follows it, the sequence is normalised as AA. In other words, the dissonance is resolved. An example is given in Figure 8.

In summary, converting to triads involves four main passes:

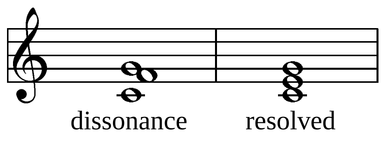


Figure 8: Resolution of dissonances

- converting off-beat non-triads
- simplifying sevenths
- completing incomplete triads
- resolving dissonances

The algorithms for these passes are given in Figures 9, 10, 11, 12 and 13.

```
let $s$ be a sequence of harmonies;
**while** more symbols in $s$ **do**
    let $i$ be current symbol;
    let $j$ be next symbol;
    let $p$ be position of $i$;
    **if** $i$ is consonance $\wedge$ $j$ is seventh $\wedge$ $p$ is odd **then**
        $j = i$;
    **else if** $i$ is consonance $\wedge$ $j$ is dissonance $\wedge$ $p$ is odd **then**
        $j = i$;
    **end if**
**end while**
```

Figure 9: Algorithm for converting off-beat non-triads

```
let $s$ be a sequence of harmonies;
**while** more symbols in $s$ **do**
    let $i$ be current symbol;
    let $j$ be next symbol;
    **if** $i$ is seventh $\wedge$ $j$ is consonant subset of $i$ **then**
        $i = j$;
    **else if** $i$ is dissonance $\wedge$ $j$ is consonant subset of $i$ **then**
        $i = j$;
    **end if**
**end while**
```

Figure 10: Algorithm for simplifying sevenths/dissonances

Conversion to RPF is robust. If a harmony cannot be converted, it is left as it is, adding a non-standard symbol to the language. Provided that conversion is successful in most cases, unconvertible harmonies are not statistically important.

To give an idea of the difference made by normalisation, trigram parameters were estimated on both RPF and non-RPF input (for major and minor subcorpora). The big difference in numbers of parameters is evident in Table 3.

```
let s be a sequence of harmonies;
while more symbols in $s$ do
    let $i$ be current symbol;
    let $j$ be next symbol;
    if $i$ is last symbol in sequence then
        if $i$ is incomplete triad then
            complete $i$;
        end if
    else if $i$ is incomplete triad $\wedge$ $j$ is triad then
        if $i$ is subset of $j$ then
            $i = j$;
        else
            complete $i$;
        end if
    else if $i$ is triad $\wedge$ $j$ is incomplete triad then
        if $j$ is subset of $i$ then
            $j = i$;
        else
            complete $j$;
        end if
    else if $i$ is incomplete triad $\wedge$ $j$ is incomplete triad then
        if $i = j$ then
            complete $i$;
            $j = i$;
        else
            complete $i$;
            complete $j$;
        end if
    end if
end while
```

Figure 11: Main loop for completing incomplete triads

```
let $i$ be incomplete triad to complete;
let $m$ be mode of piece;
let $d$ be diatonic harmonies for tonic key in $m$;
let $s$ be set of harmonies of all related keys for $m$;
if $d_1 \in d$ can be added to $i$ to form root position triad then
    do so;
else if $d_2 \in d$ can be added to $i$ to form first inversion triad
then
    do so;
else if $d_3 \in d$ can be added to $i$ to form second inversion
triad then
    do so;
end if
```

Figure 12: Algorithm for completing the incomplete triads themselves

6.4.6 Eliding identical harmonies

Once the harmonies have been converted to RPF, the sequence of harmonies contains pairs of identical quaver-length harmonies. These can be elided to form crotchet-length harmonies. Getting rid of

```
let $s$ be a sequence of harmonies;
**while** more symbols in $s$ **do**
    let $i$ be current symbol;
    let $j$ be next symbol;
    **if** $i$ is dissonance $\wedge$ $j$ is consonance **then**
        resolve $i$ to $j$;
    **end if**
**end while**
```

```
let $d$ be dissonance to resolve;
**if** $n \in d$ can be moved by one degree to $n'$ **then**
    do so;
**end if**
```

Figure 13: Algorithm for resolving dissonances

Subcorpus	Normalisation	Trigrams
all major-key pieces	RPF	1060
all major-key pieces	non-RPF	3758
all minor-key pieces	RPF	1405
all minor-key pieces	non-RPF	4554

Table 3: Number of trigram parameters with and without normalisation

redundant duplication in this way means that n-gram learning captures twice as much context than it would otherwise do, and the sequence of harmonies becomes doubly informative. It is valid to do this because, in harmonic terms, a sustained harmony is equivalent to a repeated one, so a repeated quaver-length harmony $quaver_h, quaver_h$ is equivalent to a single crotchet-length harmony, $crotchet_h$.

This is effectively another sampling pass, but instead of using the sampling algorithm in Figure 14, all that is needed is to select alternate harmonies and discard the rest.

This leaves us with a sequence of symbols corresponding one-to-one with the beats in the piece. So there is no need to represent the length of beats in our notation: the harmonic motion is represented as on a graph with time on the x-axis.

```
let $s$ be the sampling rate;
let $e$ be the end time of the last note in the piece;
$t_1 = 0$;
$t_2 = t_1 + s$;
**while** $t_2 \leq e$ **do**
    find note events whose onset time $< t_2$ and offset time $> t_1$;
    store these note events as a harmony of length $s$;
    $t_1 = t_2$;
    $t_2 = t_2 + s$;
**end while**
```

Figure 14: Sampling algorithm

6.5 Annotation

Simple n-grams are known not to capture high-level structure. To learn structural characteristics such as phrasing and modulation between keys, n-gram learning must be guided. One way of doing this is to annotate the corpus with symbols marking boundaries between constituents. In an experiment to infer a probabilistic context-free grammar (PCFG), Pereira and Schabes (1992) used a corpus of English-language text in which some phrasal constituents had been hand-marked with brackets. The grammar-learning system was then required to find productions (grammar rules) that were consistent with the bracketing.

Similarly, the present corpus is annotated with constituent boundaries. Here, the constituents are a) pieces, b) phrases and c) bars.

A certain amount of annotation must be used. In order to provide context for n-gram learning and generation, a minimum of $n - 1$ symbols of some kind must be added to the start of each string in the corpus, for instance, `start start`. And to provide a means of terminating generation, it is necessary to use an end-of-string marker – here, `end`.

Apart from being necessary, the addition of $n - 1$ start symbols and a single symbol at the end of the string also has the effect of capturing something of the way pieces begin and end. In generation, this can be used to constrain the way generated pieces begin and end, to make them more realistic than they would otherwise be.

Annotation need not be restricted to whole pieces – constituents may be annotated too. Phrase boundaries can be marked to indicate where phrases begin and end (for instance, using a `phrase` symbol), which again will allow generated phrases to be constrained to resemble phrases in the corpus. Bar boundaries too can be annotated, by inserting a symbol such as `bar` after every three symbols, as shown in Figure 15.

$$\ldots\overbrace{harm_i \ \texttt{bar} \ harm_j}^{\text{trigram}}\ldots$$

Figure 15: A trigram including a bar symbol

So far the annotation has treated phrases and bars as context-free – that is to say, the harmonies in a bar or in a phrase do not depend on where in the piece they come. In real music, phrases and bars are not context-free. Rather, they depend on what has gone before, and where they come in the overall design of the piece. There is a *progression* of keys, starting in the tonic and ending in the tonic, and taking a planned route through other keys on the way.

To capture this behaviour, we can use distinct annotation symbols for different bars or phrases. This can be done by numbering each phrase-boundary symbol.

However, `phrase_6` in a six-phrase piece would not have the same meaning as `phrase_6` in an eight-phrase piece. So we need to make a distinction. We can do this by including in the boundary symbol the length of the piece. With this system, the above two symbols become `phrase_6_of_6` and `phrase_6_of_8`.

Numbering phrases effectively makes the data quite a lot sparser, as n-grams at phrase boundaries effectively operate over different subcorpora, depending on the length of the piece they are from.

Potentially, bar boundaries could be numbered, as phrase boundaries are, to capture the precise sequences of harmonies at different positions in a piece. However, this would make data much sparser still. It is expected that simple n-grams, combined with numbered phrase annotation will be sufficient to capture the nature of harmony transitions.

Where the end of a bar coincides with the end of a phrase, it is unnecessary to mark this point with both bar- and phrase-boundary symbols. Moreover, inserting both would put some of the neighbouring harmonies out of trigram range, and important phrase-boundary sequences would not be captured. Therefore, bar-boundary symbols are not inserted at phrase boundaries. Simlarly, where the end of a phrase is also the end of a piece, only a piece-boundary symbol is inserted.

In music at large, phrases may be of variable length, though there is a strong tendency for phrases to be multiples of two or four bars. In NL, which is driven by meaning rather than meter, phrases may be of any length.

Pereira and Schabes use a manually annotated corpus (the Air Travel Information System (ATIS) corpus, prepared by the Penn Treebank project), whereas the present corpus is annotated automatically, so there is no significant overhead involved.

The annotation symbols effectively become symbols of the language, and take part in trigram estimation just as the harmonies do. The result of including these meta-symbols in the language should be that the sequences of harmonies occurring at beginnings and ends of pieces, phrases and bars should be learned. So in effect, the system learns some structure.

The annotated corpus has the general form shown in Figure 16, where each piece is a bracketed sequence. The addition of extra annotation symbols to the alphabet increases the number of possible n-gram parameters.

```
(start  phrase_1_of_l...bar...bar...bar...
        phrase_2_of_l...bar...bar...bar...
        ...
        phrase_x_of_l...bar...bar...bar...end)
(start  phrase_1_of_m...bar...bar...bar...
        phrase_2_of_m...bar...bar...bar...
        ...
        phrase_y_of_m...bar...bar...bar...end)
...
(start  phrase_1_of_n...bar...bar...bar...
        phrase_2_of_n...bar...bar...bar...
        ...
        phrase_z_of_n...bar...bar...bar...end)
```

Figure 16: Corpus annotation

6.6 Parameter estimation

Parameter estimation is done on each of the nine subcorpora (see section 5 for details of these). The procedure is as follows. The set of n-gram sequences in the subcorpus is found, and for each n-gram in this set, the number of occurrences is found. To get the probability of the last symbol in the sequence given the context, the count is divided by the number of times the first two symbols of the sequence occur together. This information is stored in a table.

Once estimation is done, another table is prepared, containing for each (n-1)-gram sequence, the set of possible next symbols and their probabilities. This is not part of the algorithm, but is useful for speeding up generation.

7 Random generation

Two types of random generation are implemented; one almost without constraints, the other constrained more. With the first kind, the generated string is initialised with a dummy context of $n-1$

annotation symbols and generation stops when an `end` symbol is randomly selected as the next symbol. Potentially this could result in a piece consisting of just one harmony since there is no length constraint. In fact, because pieces of music often begin and end on the same harmony, this scenario is quite likely. The result would be a very unsatisfactory piece of music! (A piece consisting of two identical harmonies was generated this way. See section 12.) Also, this scheme is very unlikely to generate fixed-length bars and phrases. Both are important to the music. This approach is used as a control.

The other type of generation, the mainstay of the experiments, is generation constrained by a fully annotated template. The annotation of the template is the same as the annotation of the corpus: `start` and `end` mark the beginning and end of the piece and there are `phrase_x_of_y` symbols at four-bar intervals and `bar` symbols after each bar. The general form of a template is shown in Figure 17. The number of phrases in the template is parameterised, so that one can specify that a piece of, say, five phrases should be generated.

```
(start  phrase_1_of_l...bar...bar...bar...
        phrase_2_of_l...bar...bar...bar...
        ...
        phrase_x_of_l...bar...bar...bar...end)
```

Figure 17: Template

Generation of both types works as follows. Given the context of the most recently generated $n-1$ symbols, the set of possible next symbols is obtained from the table mentioned in Section 6.6 above. The probabilities of these possible next symbols sum to 1. A random number r between 0 and 1 is generated. The probabilities of the possible next symbols are summed until r is exceeded. The symbol whose probability was added last is selected as the next symbol in the string.

Example: Possible next symbols (harmonies) and their probabilities are ((`g_major` 0.3), (`a_minor` 0.22), (`c_major` 0.28), (`e_minor` 0.2)). A random number is generated – say 0.6. The list is summed until 0.6 is exceeded. This happens when 0.28 is added to the sum. Therefore `c_major` is selected and added to the string.

7.1 Generate and test

Most strings that are randomly generated by the method described above will not match the template. These are rejected. In effect, we are saying that n-gram generation alone will not give us what we want.

The longer the template, the more unlikely it is that a randomly generated string will match the template. So if a long piece is specified, it may take a long time to generate a string that matches the template. As an implementational way round this, each phrase is generated separately. If a string is generated that is consistent with the annotation of the phrase template, the phrase is kept, and the next phrase template is instantiated, otherwise the string is discarded and another is generated. Each phrase has its own template and is instantiated separately. To achieve continuity between phrases, once a template has been instantiated, its last $n-1$ symbols are added to the beginning of the next template to provide context for the first symbol in the next template. Generating phrase by phrase in this way is equivalent to generating the whole string in one go.

After generation, annotation symbols are removed, leaving just the newly composed sequence of harmonies.

8 Experiments

This section describes a set of experiments in generation that were carried out using 3-gram (trigram) and 4-gram parameters trained on the nine subcorpora.

8.1 Testing

Several experiments in generation (composition) were carried out. A number of experimental variables were tested:

- different kinds of annotation

  piece boundaries only

  piece and phrase boundaries

  piece, phrase and bar boundaries

  piece, numbered phrase and bar boundaries

- size of n
- different sizes of subcorpus

Some experimental variables that were *not* tested were:

- non-RPF input to n-gram estimation
- length of piece

A control experiment was carried out, where training was done on a minimally annotated corpus with only beginnings and ends of pieces marked, and generation was done without using a template. Generation stopped when an end symbol was selected.

Generated pieces were assessed subjectively by how closely they were judged to resemble the music of the corpus.

The music of the corpus is in one of two modes: major or minor.

In all experiments, learning was done separately on subcorpora of major- and minor-key pieces.

Before running the experiments, it was expected that the best results would be obtained with full annotation (*i.e.,* piece, *numbered* phrase and bar boundaries all marked), an n of 4 (*i.e.,* 4-grams), and with the larger subcorpora. In the corpus pieces, harmony tends to change at the beginnings of bars. Marking bars should force this to happen in generation. Phrases end with particular sequences of harmonies. Again, these sequences should be learned if phrase annotation is used.

The experiments that were carried out are summarised in Table 4. For each generation experiment, ten pieces were generated. A selection of the generated pieces are shown in Section 12.

The formula 2, the probability of a string, is fine as long as all the strings that are worked with are of the same length. However, if the lengths differ, shorter strings will almost always be assigned higher probabilities than longer ones (as can be seen starkly in the pieces generated without templates in 12.1 on page 31). This is inappropriate. Typically, pieces in the present corpus are 24 bars long (62% of them). The shortest is 16 bars and the longest 40 bars, and the length is always a multiple of four. There are no pieces of 6 bars; nor would there ever be, so to assign a high probability to a 6-bar sequence would be inappropriate.

What is needed is either some way of removing string length from the equation or of favouring strings whose lengths are those found in the corpus.

The first effect can be achieved by taking the lth root of the string probability (*i.e.*, the geometric mean of all the probabilities comprising the string). For trigrams, this is

$$M(w_{1,l}) = \sqrt[l]{\prod_{i=1}^{l} P(w_i|w_{i-2}, w_{i-1})} \tag{4}$$

where l is the length of the string.

This removes length of string from the equation. However, the result is no longer the probability of the string. Rather, it is the probability of a transition in the string assuming that the probabilities of all transitions are equal.

In the absence of a sound model that takes into account string length as found in the corpus, experiments in generation were done with strings of uniform length (24 bars, since this is the most common length in the corpus).[4] This meant that probabilities could be compared meaningfully.

Training corpus	Annotation	n
whole (both modes)	phrasing	3
whole (both modes)	phrasing, barring	3
whole (major)	phrasing	3
whole (major)	phrasing, barring	3
whole (minor)	phrasing	3
whole (minor)	phrasing, barring	3
Chambonnières (major)	phrasing	3
Chambonnières (major)	phrasing, barring	3
Chambonnières (minor)	phrasing	3
Chambonnières (minor)	phrasing, barring	3
L. Couperin (major)	phrasing	3
L. Couperin (major)	phrasing, barring	3
L. Couperin (minor)	phrasing	3
L. Couperin (minor)	phrasing, barring	3
whole (major)	numbered phrasing, barring	3
whole (minor)	numbered phrasing, barring	3
whole (major)	numbered phrasing, barring	4
whole (minor)	numbered phrasing, barring	4

Table 4: Experiments in generation

Strings generated from parameters trained on a particular corpus will be typical of that corpus (at least if the training corpus is big enough). So strings generated using parameters trained on all the major-key pieces should be typical seventeenth-century major-key *sarabandes*; and strings generated using Chambonnières' minor-key parameters should be typical of Chambonnières' minor-key *sarabandes.*

8.2 Results

In interpreting the probabilities of generated pieces, probabilities have to be compared *within* a subcorpus, not across subcorpora, because parameters were trained on different amounts of data. Pieces

[4]Exceptions to this were pieces generated without using templates. Here, generation stops just when an `end` symbol is selected.

trained on large amounts of data, where there were more parameters, generally have lower probabilities because the probability mass is split more ways.

In all, around 200 pieces were generated. For reasons of space, we concentrate here on the results obtained from the larger subcorpora – the one containing all major-key pieces and the one containing all minor-key pieces. In each case, the highest probability examples are chosen for discussion. The generated pieces are included in section 12.

If the model is accurate, the pieces should be typical of seventeenth-century *sarabandes*, major and minor.

Questions to be answered about the generated pieces are:

- does barring make a difference?
- are beginnings and ends of pieces learned?
- are harmony transitions learned?
- are phrases learned?
- are cadences learned?
- is key progression learned?
- what kinds of modulations occur?

Generating without templates, the highest probability piece, piece 1, consists of just two major triads! This illustrates the need for length to be treated more realistically. Interestingly, piece 2, a much longer piece of much lower probability, contains some fine transitions and, being unconstrained by meter, is reminiscent of the free *préludes* that were written at the time of this music. Perhaps we have hit upon a way of generating *préludes* – a bonus.

Piece 3 contains plausible phrases and transitions. The final cadence is weak. The preceding phrases mix imperfect and perfect cadences, which is appropriate. Piece 4 also has plausible phrases with good endings. It has a fairly strong final perfect cadence. This is by chance, though, since the length of the cadence (almost the whole phrase) is well outside the range of trigrams. The use of templates and generate and test ensures that the final cadence is in the right key.

Piece 5 has some nice modulations to the dominant key (the key on the 5th degree). But the last cadence is rather weak. The first two phrases of piece 6 are convincing. They have good transitions and go to appropriate keys. However, the final plagal cadence would not happen. Again, the cause is the limited range of trigrams.

Pieces 7, 8, 9 and 10 are all generated using numbered phrases. This is intended to force a suitable progression of keys. The four examples here are not sufficient to judge whether this works in general. However, inspection of the whole set of results shows that more plausible progressions do result from numbering phrases.

Pieces 9 and 10 are generated with 4-grams, and therefore capture more context than the pieces generated with 3-grams. In general, this has a good effect. However, when annotation is used, as it must be to capture structure, 4 is not always a big enough context. Piece 10 finishes with a weak sequence that does not resemble anything in the corpus, because the context was not large enough to force a perfect cadence.

Generally, transitions between one harmony and the next resemble the transitions in the corpus pieces. Often, neighbouring harmonies are different only by one note, which is appropriate.

Barring does help to group harmonies into multiples of 3 (*i.e.,* into metrical units) – typically AAA or AAB groupings.

Pieces almost always begin and end in the same key. This is always true of the corpus pieces.

Generally, phrases end in appropriate cadences (either an imperfect cadence or a perfect cadence). The final cadence should always be a perfect one, and usually is. However, 4-grams never see more than three harmonies at a time when annotation is used, so there can be no guarantee that an appropriate cadence will be generated, though one often is. Minimally, a perfect cadence consists of a triad on the 5th degree, followed by one on the 1st. But to be a strong cadence, these should be closely preceded by a triad on either the supertonic or the subdominant.

Individually, phrases tend to be convincing. They begin in one key and modulate to another, ending in a cadence. Collectively they are more convincing when numbered phrase-boundary symbols are used than when a general phrase-boundary symbol is used throughout. When phrases are numbered, pieces generate pass through a suitable series of keys. When phrases are not numbered they are not constrained to do this, and there may be several consecutive phrases ending in the same way, with no progression. This is boring, and is avoided in well-written pieces. The difference is one of treating phrases as context-free – which they're not – and treating their key as dependent on the position in the piece.

The numbering of phrases does create a problem of its own, though, as it effectively makes the data more sparse, making it necessary to increase the size of the corpus.

A feature of generated pieces that does not occur in real pieces is enharmony, where distantly related harmonies are bridged by a shared harmony (especially a diminished triad). If the aim is to imitate style, this is bad news, as music of this period usually avoids sudden modulations to remote keys. The cause is again the small size of the context. However, enharmony does give interesting results.

9 The Grammars

An obvious question to arise at this stage is "What do the grammars look like?". In standard Chomskian terms, this is a non-trivial question to answer, because the n-gram parameters are expressed simply as symbols and probabilities associated with a pair of existing symbols, as explained above. Further, the generation system we use above applies a strong meta-level constraint to its output: that it has to be of the right length and bar structure. For a Chomskian-style grammar to be at all comprehensible in this context, this constraint needs to be brought into the grammar itself.

However, it is possible[5] to translate the parameters into grammars, and, in so doing, perform a process of "partial evaluation" of the meta-level constraint, so that the only rules produced are ones which lead to an output which is valid according to the constraint. The procedure is explained in the next section.

9.1 The Conversion Procedure

9.1.1 A Finite State Transition Network

The first step of converting the n-gram parameters is to view the parameters as labelled arcs in a Finite State Transition Network (FSTN). It is known that FSTNs are equivalent to Chomsky type 3 (or Finite State) grammars (Hopcroft and Ullman, 1969), so this is a step towards a Chomskian grammar.

An FSTN is formally a finite collection of states connected by directed arcs. Each arc is associated with a symbol (in our case, a harmony). There is a distinguished *start* state and a distinguished *end* state.

To use the network, we begin at the *start* state, and proceed to other states, either reading or generating their associated symbol as we go. If we reach the *end* state, we have read or generated a

[5]This idea was developed in collaboration with Michael Walker, an MSc student in the School of Cognitive Science at Edinburgh, to whom the authors are grateful.

well-formed string in the language defined bythe FSTN. An FSTN for the two sentences “The man bites the dog.” and “The dog bites the man.” is shown in Figure 18. Note that this FSTN is also capable of reading/generating “The man bites the man.” and “The dog bites the dog.”.

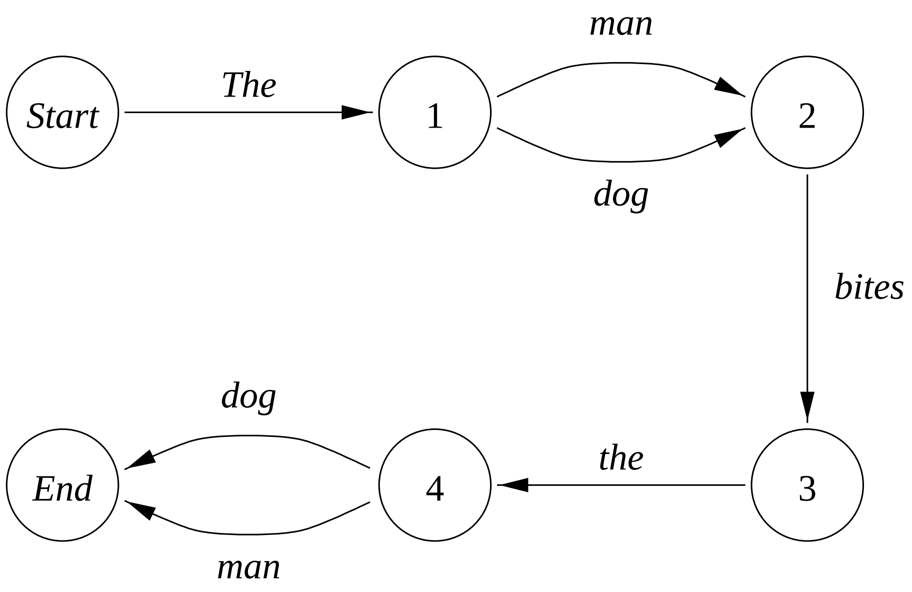


Figure 18: An example Finite State Transition Network

We extend the representation to allow each arc to be associated also with a probability parameter, taken from the n-gram probability, so each arc is associated with a ⟨harmony,probability⟩ pair.

Each state in our FSTN is represented as an m-tuple of harmonies, where m is $n - 1$. In the discussion here, we use the output from our 3-gram experiments, so m is 2. Each n-gram parameter is then converted into an arc between two states. The first state is the named after the context of the parameter, in order of occurrence. The second is named after the new context which will be current *after* the parameter – *viz.* the second of its context harmonies paired with the parameter of this arc. The state named *start-start* is then the *start* state, and any state name *X-end* where X is any harmony is an *end* state.

Consider, for example, the following 3-gram parameter.

```
p( [ ii, I ], V, 0.25 ).
```

This states that in a harmonic context of *ii* followed by `I`, the likelihood of a *V* harmony occurring is 0.25. In our FSTN representation, it is represented by the arc shown in Figure 19.

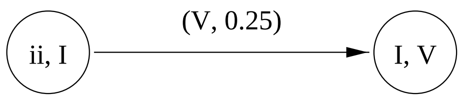


Figure 19: An example FSTN arc

In the first instance, we can straightforwardly produce a network which captures the parameter set. However, this parameter set may contain arcs such as the one shown in Figure 20.

In this instance, it is clearly possible for a repeated sequence of *I* harmonies to be produced. Since we also have `bar` symbols in our language, which will therefore appear as arc labels in the

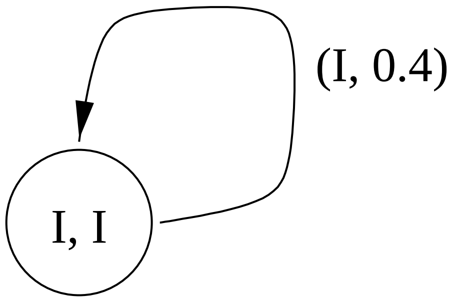


Figure 20: A looping FSTN arc

FSTN, it is clearly possible for an FSTN containing the arc of Figure 20 to generate bars of arbitrarily many beats, which is unacceptable. The next step, therefore, is to unfold any such looping arcs, and control them in such a way that loops are not possible.

9.1.2 Generating an extended FSTN

To perform this step, we need to extend our state representation slightly further. In principle, unfolding a loop would simply lead to an infinite number of identical non-looping arcs. We will pick a subset of these, but it is important during the generation process to know *which* of the subset we are working with. This will become clearer in the example below. Therefore, we extend our state representation to include the length of the sequence generated up to each state. (In fact, the generality is that each beat of the generated piece must be distinct, but this is a neat way of implementing it.) Subsequently, an application of the arc in Figure 20 at symbol 2 would look like Figure 21(a), while an application at beat 6 would look like Figure 21(b). This means that there is no possibility of looping, and that the network can be restricted to producing `bar` and `phrase` symbols where appropriate.

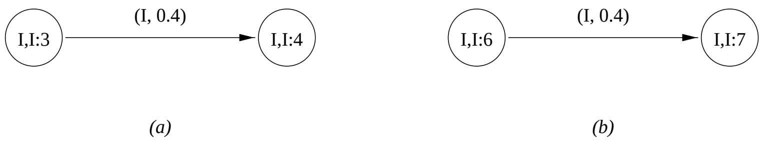


Figure 21: Two states in our extended FSTN

In order to perform this new labelling, it is necessary to elaborate the search space defined by the network. This is a drawback, as the search space is potentially very large, and we have to consider its entirety. However, we are saved in this instance by the restricted size and strict form of our output – many possible branches in the tree can be pruned early on, to save wasted effort.

The algorithm works by exploring the tree beat by beat, insisting that every fourth symbol be `bar` (or `phrase`, where appropriate). As each arc from our initial FSTN is successfully traversed, according to this restriction, it is added to a representation of a new FSTN in the extended representation described above. This means that arcs in the original FSTN are constrained to appear only at certain places in the pieces generated by the extended representation version.

Another consequence of applying the barring constraint is that some paths through the search tree will never reach an *end* state. Subsequently, therefore, it is necessary to trace backwards through the resulting network, removing states which are dead ends. This action is performed repeatedly until no dead ends are left, and at this stage the network is complete.

Now, we could translate our FSTN directly (and trivially) into a Type 3 grammar, or, more naturally for a musician, into a Type 2 grammar, but without using recursion. However, the grammars produced are simply too large to be presented here, except in excerpt – see Figure 22 for a flavour – and so they are made available at the following Web site, where they may be freely downloaded and used for research purposes:

`http://www.dai.ed.ac.uk/~geraint/music_grammars/`

$S_0 \longrightarrow$‘I’ S_1	(1.000)
$S_1 \longrightarrow$‘I’ S_2	(0.933)
$S_1 \longrightarrow$‘IV’ S_3	(0.067)
$S_2 \longrightarrow$‘I’ ‘bar’ S_4	(0.772)
$S_2 \longrightarrow$‘II’ ‘bar’ S_5	(0.002)
$S_2 \longrightarrow$‘IV’ ‘bar’ S_6	(0.093)
$S_2 \longrightarrow$‘V’ ‘bar’ S_7	(0.038)
$S_2 \longrightarrow$‘ii’ ‘bar’ S_8	(0.057)
$S_2 \longrightarrow$‘vi’ ‘bar’ S_9	(0.038)
$S_3 \longrightarrow$‘I’ ‘bar’ S_{10}	(0.285)
$S_3 \longrightarrow$‘IV’ ‘bar’ S_{11}	(0.429)
$S_3 \longrightarrow$‘V’ ‘bar’ S_{12}	(0.143)
$S_3 \longrightarrow$‘ii’ ‘bar’ S_{13}	(0.143)
$S_4 \longrightarrow$‘I’ S_{14}	(0.241)
$S_4 \longrightarrow$‘I9’ ‘V’ ‘V’ ‘bar’ S_{15}	(0.017)
$S_4 \longrightarrow$‘II’ S_{16}	(0.017)
$S_4 \longrightarrow$‘IV’ S_{17}	(0.242)
$S_4 \longrightarrow$‘IV7’ S_{18}	(0.017)
$S_4 \longrightarrow$‘V’ S_{19}	(0.294)
$S_4 \longrightarrow$‘ii’ S_{20}	(0.120)
$S_4 \longrightarrow$‘vi’ S_{21}	(0.052)

Figure 22: Fragment of Type 3 grammar for the Chambonnières corpus

Figure 22 shows the first few beats of our Chambonnières major mode corpus, rendered as a Type 3 grammar. Symbols in ‘’ are terminal symbols, and non-terminal symbols are of the form S_i, S_0 being the start state. Probabilities are shown in () after the corresponding rule. Note that these probabilities are adjusted to account for the fact that some of the possible arcs in the raw learned grammar are rendered infeasible by the barring process described above, and that rules introducing certain symbols, which are not readily represented in standard notation, are omitted – however, these rules are included in the full FSTN which can be obtained from our Web site. The archived data contains learned probabilities, not adjusted ones, so choices must be made by roulette wheel selection.

10 Conclusions

The first conclusion is that this task *can* be done – it is possible to infer harmonic frameworks using n-grams, normal form harmony and annotation. Sequences of harmonies are produced that are characteristic of the training corpus in terms of harmony transitions, the way in which pieces, phrases and bars begin and end, modulation between keys, and the relation between harmony change and meter.

A declared aim was extensibility. Parameter estimation and generation are language-independent, and so are necessarily usable for other music. Annotation is extensible provided that heuristics can be developed that will find phrase boundaries (*i.e.*, find cadences). These heuristics do not need to be totally reliable; as long as the corpus is big enough, there is room for error in a system based on counting.

Sampling of harmony is appropriate for any harmonic music – a vast range. Conversion of harmony to a normal form is reasonable in any harmonic music where harmony is treated as abstract sets of notes (most Western musics). The current base of harmonic knowledge is applicable to many other kinds of harmonic music besides the music of the corpus. It would need to be augmented to cater for other musics, but not replaced.

It is necessary to use 4-grams in order to stand a better chance of capturing cadences. With 3-grams, the most that can be seen at the end of a phrase is the last two symbols (because one symbol is occupied by an annotation symbol). With 4-grams, the whole last bar of the phrase is seen. This does not guarantee that even the final two harmonies will be seen, but it does make this more likely.

In general, alterations to the basic scheme, such as numbering phrases or increasing n to 4, makes the data more sparse, and needs to be accompanied by an increase in corpus size.

In some cases, pieces assigned low probabilities by the model seem to a human judge to be better than some of the higher probability pieces. For instance, piece 11 is a more convincing example of a minor-key *sarabande* than piece 10, but is assigned a lower probability. The model's rating should accord with the human's. Therefore, some work is needed to improve the model.

The current approach is conceptually simple, especially once preprocessing has been done. There is only one learning component – the n-gram technique, and musical knowledge is restricted to the use of annotation symbols to mark constituents in the corpus.

Length of string remains an issue. Currently, strings of any length can be generated, but it makes no sense to compare the probabilities of strings of different length, as there is a bias towards shorter strings.

11 Further work

Some refinements to the current system would improve the model, and extra work on bass lines would extend coverage to the remaining fundamental feature of the music.

11.1 Developing preprocessing

Currently, some complex harmonies are not converted to RPF. These harmonies are formed at the sampling stage when notes of shorter length than the sampling rate are bundled together.

The problem is a conflict between the sampling rate and the shortest note value of the piece. Where there are many notes that are shorter than the sampling rate, this causes there to be a lot of complex harmonies which will not be convertible. The sampling rate is chosen as the note length at which significant harmonic movement occurs. This is not flexible. However, it is possible to simplify rhythms involving notes that are shorter than the sampling rate. Equivalence rules such as

$$dotted\_quaver_{n1}, semiquaver_{n2} \rightarrow quaver_{n1}, quaver_{n2}$$

$$quaver_{n1}, semiquaver_{n2}, semiquaver_{n3} \rightarrow quaver_{n1}, quaver_{n3}$$

will do this.

11.2 A more general normal form

On the whole, conversion to RPF works well, capturing the underlying harmonic similarities between pieces. In some ways RPF goes too far, with the result that characteristic features, like sevenths near cadences, are missed. It would be straightforward to develop RPF as a normal form that allowed sevenths, ninths and possibly other non-triads, but still converted everything to root position, to eliminate some of the variation that obscures trends.

11.3 A more general way of identifying phrases

The next step would be to develop cadence-finding heuristics, to allow the system to be used for music with variable phrase length.

11.4 Developing annotation

As suggested in Section 6, the idea of numbering constituents could be taken further, by giving bar-boundary symbols unique numbers. Annotation of a 24-bar piece would then be as shown in Figure 23. This would make the data more sparse than numbering phrases does, and would certainly need to be accompanied by a big increase in corpus size.

```
(start  bar_1_of_24...bar_2_of_24...bar_3_of_24...bar_4_of_24...
        bar_5_of_24...bar_6_of_24...bar_7_of_24...bar_8_of_24...
        ...
        bar_22_of_24...bar_23_of_24...bar_24_of_24...end)
```

Figure 23: Annotation with numbered bars

11.5 A larger corpus

The switch from a context-free phrase to a context-dependent one effectively reduces the amount of data. To compensate for this, the corpus ought to be made bigger. Enough material exists for this to be possible without moving to a different music.

11.6 Smoothing

Enlarging the corpus is one way of coping with sparse data. Another is to use smoothing. Smoothed n-grams were discussed in section 3. An HMM could alternatively be used.

11.7 Adding a bass

Originally the plan was to model harmony, key scheme and bass line, since this in the seventeenth century was the basis of a piece of music.

Masson (1705) recommended composing a bass before the upper parts and wrote that “la baze [the bass] [est] le fondement des autres Parties”. Rameau (1722) wrote on the subject of “How to know which Chords must be given to the Bass Notes in any progression”, suggesting that the bass even came before the harmony.

Figured bass, a very common seventeenth- and eighteenth-century way of sketching a piece, consisted of a bass line and harmonies, indicated by figures.

Limited time meant that it was not possible to model bass lines. However, to add a bass line to our sequences of harmony, we could exploit the parallel between this task and the task of assigning part-of-speech tags to a sequence of words in a text. Church (1988) computes the probability of a tag sequence, given a sequence of words, as the maximised product of tag trigrams and lexical (word) frequencies.

The quote from Rameau, above, suggests that a bass might be composed before the harmony, in which case we could reformulate the task as assigning harmonies to bass notes.

11.8 Accounting for length

The final problem is the problem of length. Shorter strings are assigned high probabilities and longer strings low probabilities, regardless of the quality of the transitions (regardless too of the actual length of strings in the corpus).

What is needed is some formula that introduces a bias in favour of the string lengths found in the corpus.

12 Pieces generated

This section contains *sarabandes* that were randomly generated using n-gram parameters. For ease of reading, all pieces have been converted into C (C major for major-key pieces and C minor for minor-key pieces) and are represented using stave notation.

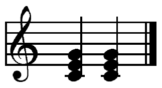
Each piece has a label, indicating the training corpus, type of annotation and size of n used.

All pieces generated using templates are 6 phrases in length (*i.e.*, 72 beats). This is the most common length of string in the corpus.

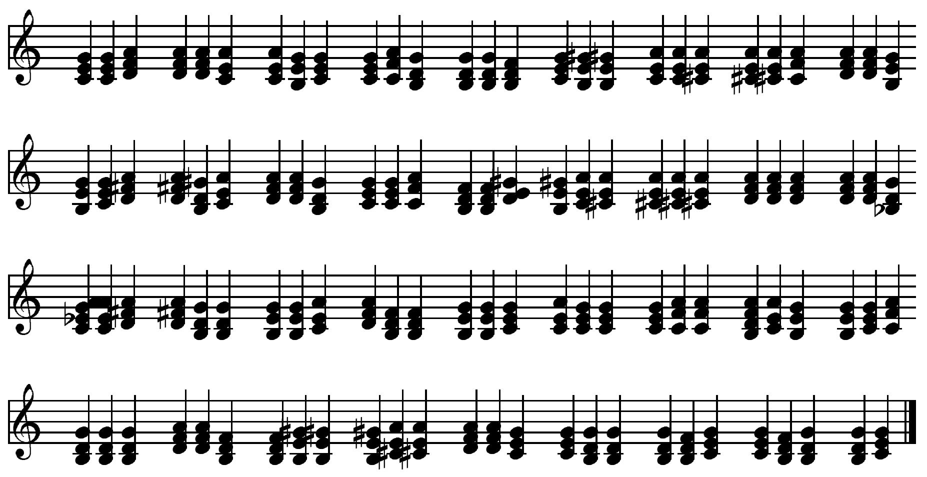
The output of generation is a sequence of abstract harmonies whose precise arrangement (register, inversion) is unspecified. For the purposes of displaying the results of generation, it was necessary to choose a specific arrangement for the notes of each harmony. This was done by starting with a root position chord and subsequently arranging chords so as to minimise movement of parts. Note that accidentals are shown in the modern convention: a note with no accidental is a natural in all cases, and accidentals are shown wherever they occur, even if twice in one bar.

12.1 Generation without templates (3-grams)

Piece 1 *Major-key pieces, start and end markers only.*

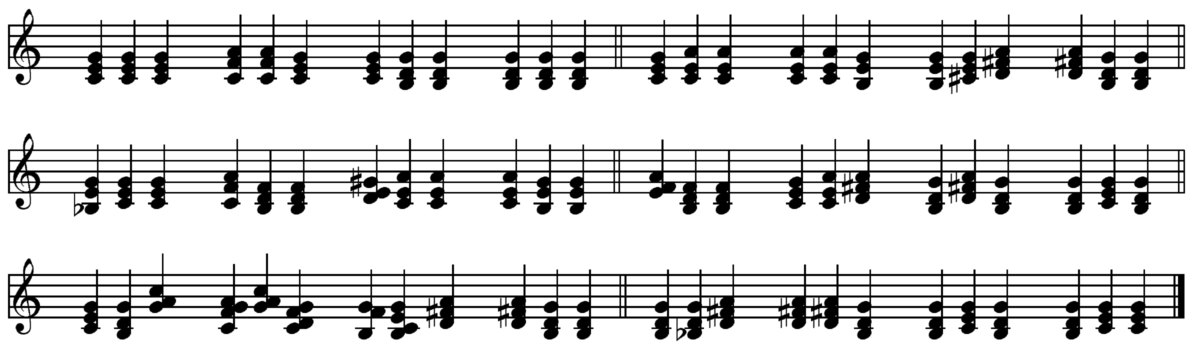


Piece 2 *Major-key pieces, start and end markers only.*

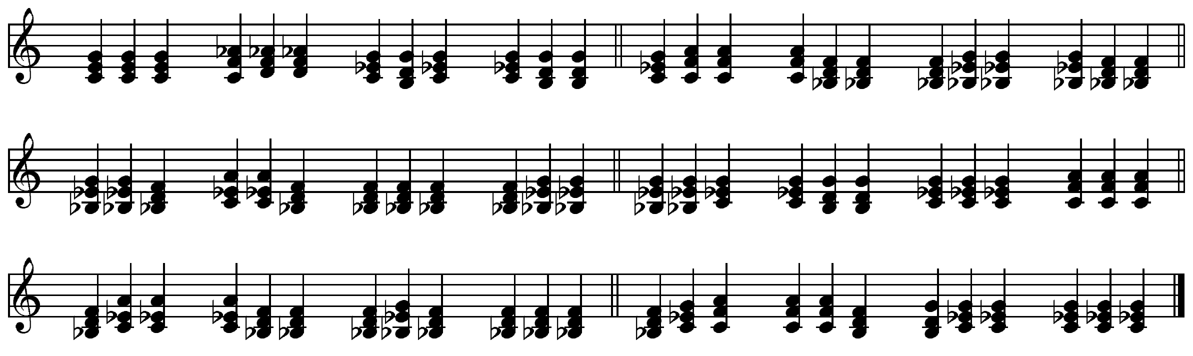


12.2 Generation with phrase markers (3-grams)

Piece 3 *Major-key pieces, phrase markers.*

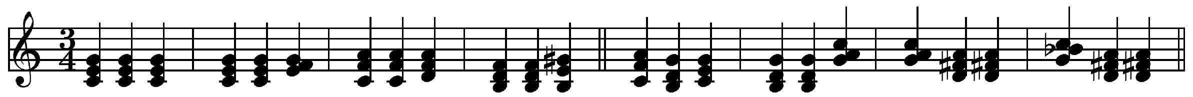


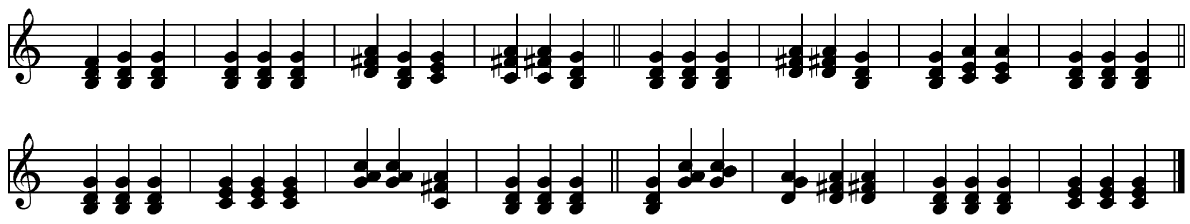
Piece 4 *Minor-key pieces, phrase markers.*



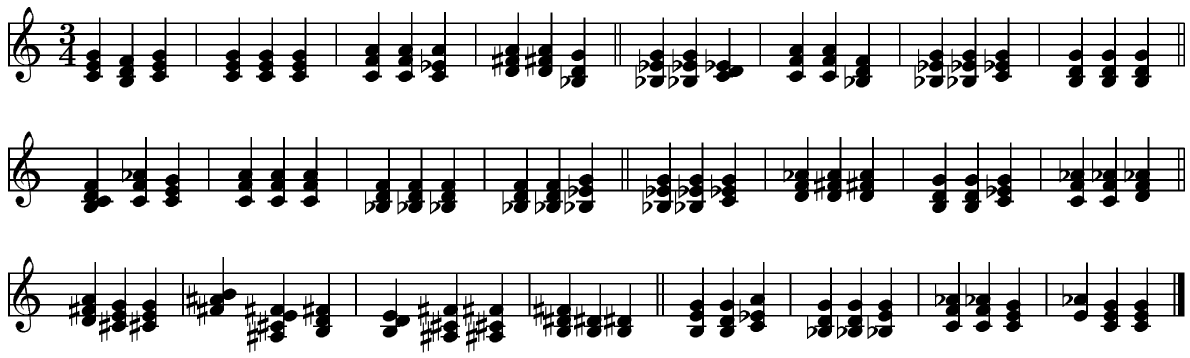
12.3 Generation with phrase markers and barring (3-grams)

Piece 5 *Major-key pieces, phrase markers and barring.*



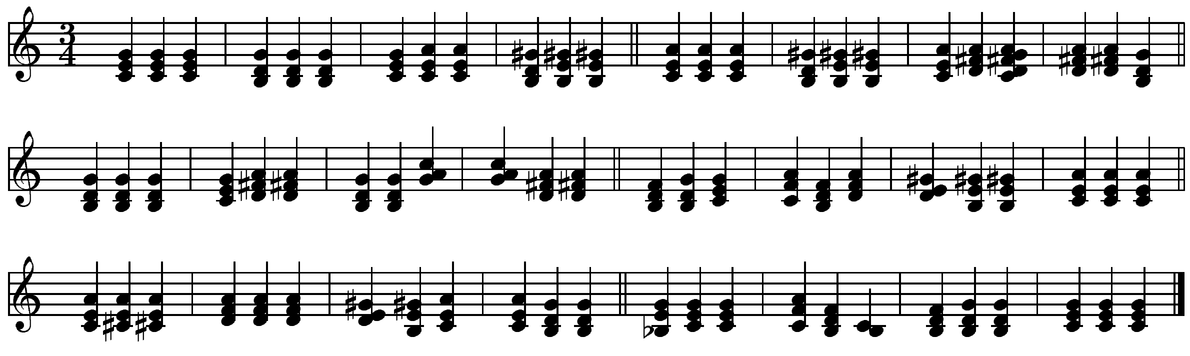


Piece 6 *Minor-key pieces, phrase markers and barring.*

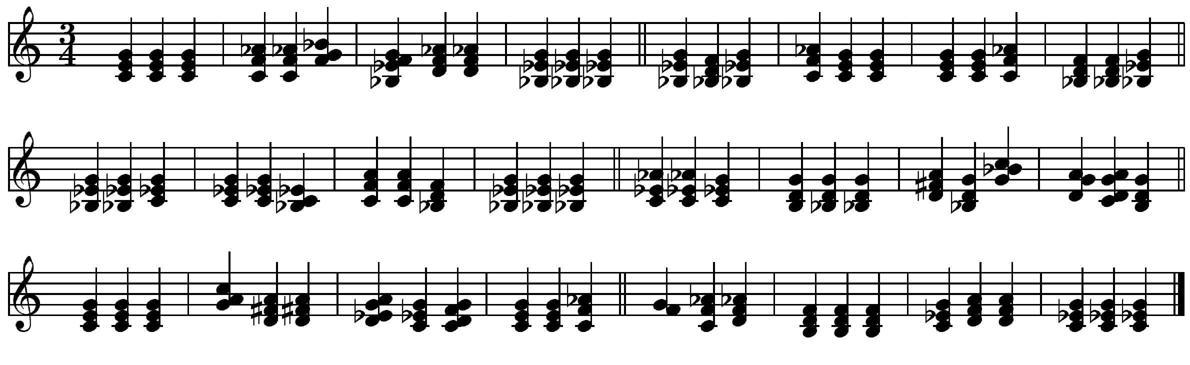


12.4 Generation with numbered phrase markers and barring (3-grams)

Piece 7 *Major-key pieces, numbered phrase markers and barring.*

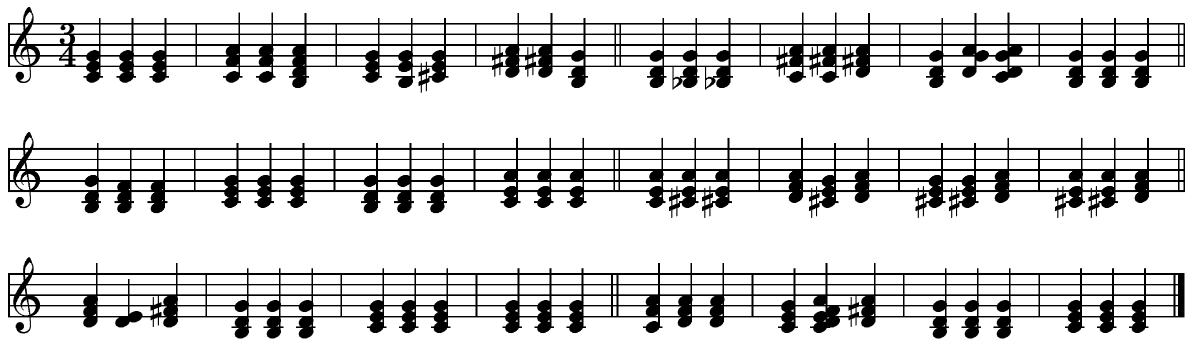


Piece 8 *Minor-key pieces, numbered phrase markers and barring.*

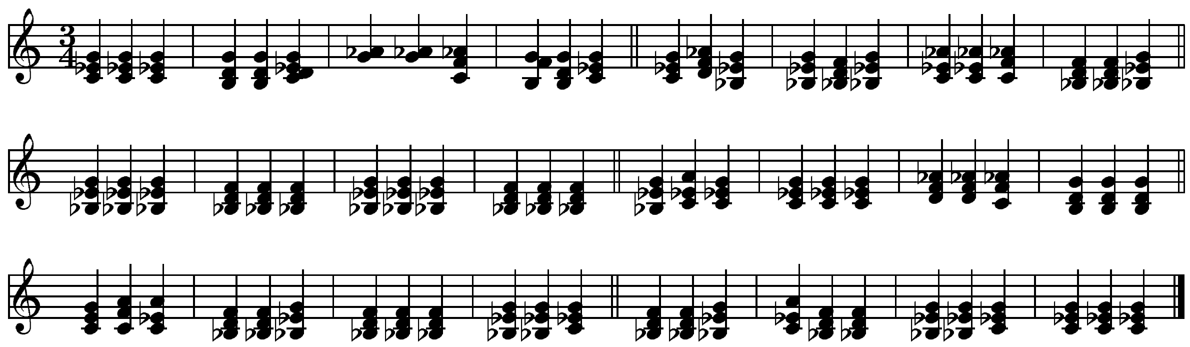


12.5 Generation with numbered phrase markers and barring (4-grams)

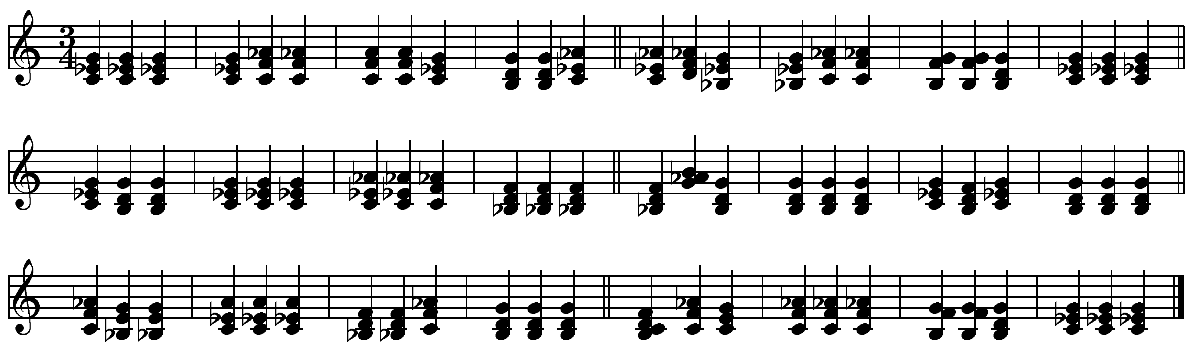
Piece 9 *Major-key pieces, numbered phrase markers and barring, using 4-grams.*



Piece 10 *Minor-key pieces, numbered phrase markers and barring, using 4-grams.*



Piece 11 *Minor-key pieces, numbered phrase markers and barring, using 4-grams.*



Acknowledgements

We would like to thank Chris Brew for his advice on statistical NLP techniques and Stephen Watkinson for the use of his program. Dan Ponsford's work was supported by studentship number 96417425 from the UK Engineering and Physical Sciences Research Council.

References

Ames, C. (1989). The Markov process as a compositional model: A survey and tutorial. *LEONARDO*, **22**(2), 175–87.

Barbaud, P. (1965). *Initiation à la composition musicale automatique.* Dunod, Paris.

Baum, L. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities*, **3**, 1–8.

Brooks Jr., F., Hopkins Jr., A., Neumann, P., and Wright, W. (1993). An experiment in musical composition. In Schwanauer and Levitt, editors, *Machine Models of Music*, pages 23–40. MIT Press.

Brown, P., Della Pietra, V., de Souza, P., Lai, J., and Mercer, R. (1992). Class-based n-gram models of natural language. *Computational Linguistics*, **18**(4), 467–79.

Charniak, E. (1993). *Statistical Language Learning.* MIT Press.

Church, K. (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing.*

Church, K. and Mercer, R. (1993). Introduction to the special issue on computational linguistics using large corpora. *Computational Linguistics*, **19**(1), 1–24.

Conklin, D. and Witten, I. (1995). Multiple viewpoint systems for music prediction. *Interface*, **24**, 51–73.

Cope, D. (1991). *Computers and musical style.* Oxford University Press.

Hopcroft, J. and Ullman, J. D. (1969). *Formal Languages and their relation to automata.* Addison-Wesley.

Jelinek, F. and Mercer, R. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Pattern Recognition*, pages 381–97. North Holland.

Jelinek, F. and Mercer, R. (1985). Probability distribution estimation from sparse data. Technical Report 2591–94, IBM Technical Disclosure Bulletin.

Krenn, B. and Samuelsson, C. (1996). *The Linguist's Guide to Statistics.* Available on the WWW.

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

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing.* MIT Press, Cambridge, MA.

Masson, C. (1705). *Nouveau traité des regles pour la composition de la musique.* Republished by Da Capo Press, New York (ed. I. Horsley), 1967.

McEnery, T. and Wilson, A. (1996). *Corpus Linguistics.* Edinburgh University Press.

Pereira, F. and Schabes, Y. (1992). Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the Fifth DARPA Speech and Natural Language Workshop.*

Pinkerton, R. (1956). Information theory and melody. *Scientific American*, (194), 76–86.

Rameau, J.-P. (1722). *Treatise on Harmony.* Republished by Dover, New York (ed. and trans. P. Gossett), 1971.

Rothstein, J. (1992). *MIDI: A Comprehensive Introduction.* Oxford University Press.

Smaill, A., Wiggins, G., and Harris, M. (1993). Hierarchical music representation for composition and analysis. *Computers in the Humanities*, **27**, 7–17.