

Multiple Viewpoint Systems for Music Prediction *

Darrell Conklin

Department of Computing
City University, London
Northampton Square
London EC1V 0HB

Ian H. Witten

Dept. of Computer Science
The University of Waikato
Hamilton, New Zealand

Abstract

This paper examines the prediction and generation of music using a *multiple viewpoint system*, a collection of independent views of the musical surface each of which models a specific type of musical phenomena. Both the general style and a particular piece are modeled using dual short-term and long-term theories, and the model is created using machine learning techniques on a corpus of musical examples.

The models are used for analysis and prediction, and we conjecture that highly predictive theories will also generate original, acceptable, works. Although the quality of the works generated is hard to quantify objectively, the predictive power of models can be measured by the notion of entropy, or unpredictability. Highly predictive theories will produce low-entropy estimates of a musical language.

The methods developed are applied to the Bach chorale melodies. Multiple-viewpoint systems are learned from a sample of 95 chorales, estimates of entropy are produced, and a predictive theory is used to generate new, unseen pieces.

*This paper appeared in *Journal of New Music Research*, 24(1):51–73, 1995. Here with corrections (see Acknowledgements), 2002.

1 Introduction

This paper is concerned with machine learning and evaluation of music theories. A theory of music is an intensional description of some musical language. Theories are evaluated according to the predictions they make about particular pieces of music. In addition to explaining known pieces, a theory of a musical style is expected to generate new pieces that are acceptable — even creative — specimens of the style. Musical styles are vast and complex languages; we have begun our work by tackling the problem of constructing theories for melody, a necessary prerequisite to more advanced issues such as polyphony and harmony. The Bach chorale melodies were chosen as the object of analysis due to their abundance, simplicity and general display of good melodic form.

There are two approaches to the construction of a generative theory for a musical language. The first is the knowledge engineering approach, where rules and constraints are explicitly coded in some logic or grammar (Cope, 1987; Ebcioğlu, 1986; Lidov and Gabura, 1973; Hiller, 1970; Baroni and Jacobini, 1978). The second is the empirical induction approach, where a theory is developed through an analysis of existing compositions (Brooks et al., 1956; Kohonen, 1989; Conklin and Cleary, 1988; Schwanauer, 1993). The knowledge engineering approach was discarded after careful consideration. There are too many exceptions to any logical system of musical description, and it will be difficult to ensure the completeness of an intuited theory. The system will always exclude some valid pieces. The generations of a theory are bound to reflect the biases of an engineer; this is the only way they might be called creative. This paper uses the empirical induction approach to generative theory construction.

Predicting plausible continuations in a musical style presents difficulties not encountered in other domains. In addition to capturing general stylistic rules, a prediction system must also capture sequential structure, pattern, and repetition within a particular work (Brown and Dempster, 1989). The presence of these two interacting levels of description makes music a unique problem domain. A theory of a musical style needs to represent two concepts simultaneously — that of “piece in a style,” as well as “next event in a piece.” The combined concept being acquired here is that of “next event in a chorale melody.”

This research is motivated by the goal of learning generative theories for the chorale genre. The evaluation of generative theories will always be subjective, varying with individual listeners. To assess this research, it is necessary to have a more precise and replicatable method for evaluating music theories. The conjecture of this paper is that highly predictive theories for the chorales will also generate original, acceptable works. The predictiveness of a theory can be precisely measured using entropy.

This paper is primarily about the new formalism of multiple viewpoint systems for machine learning and prediction of music, secondarily about its use on the Bach chorale melody data set, and only peripherally with the generation of new music from learned theories. The paper is structured as follows. Section 2 develops the formal basis for prediction, entropy estimation, and generation of music. It reveals the idea of *non-exclusive* music theories which do not view pieces as being “true” or “false,” but rather regard all representable musical surfaces as possible. Section 3 presents a general framework for learning about sequences, and describes the model used to represent and learn sequence-generating rules. This framework, derived from research on data compression and grammatical inference, is sub-

stantially extended in Section 4, where multiple viewpoint systems are introduced. These are a novel distributed knowledge representation scheme, that can concurrently represent and reason about multiple properties of the musical surface. Section 5 shows the chorales to be amenable to a multiple viewpoint representation, and develops an appropriate set of viewpoints. Finally, Section 6 presents some experimental results, which indicate that the entropy of the chorale genre is probably not greater than 1.87 bits/pitch. An example of a generated chorale is presented.

2 Prediction, entropy, and generation of music

In his paper “Prediction and Entropy of Printed English,” Shannon (1951) outlined the idea of a predictive theory of a language as a transducer of letters into compressed codes. This transduction is reversible, so that it is also possible to exactly reconstruct a letter from a compressed code. Figure 1 outlines this prediction scheme for general sequences. A context — a sequence of events — and an event are shown to a theory T , which is in “predict” mode and produces a compressed code. When the code and original context are shown to a theory — which is now in “generate” mode — the original event is reproduced.

The prediction scheme of Figure 1 forms the basis for modern data compression theory (Bell et al., 1990). It can be shown that if the probability of an arbitrary event e , given a context c and theory T , is $p_T(e | c)$, the code produced cannot have a length of less than

$$-\log_2 p_T(e | c) \tag{1}$$

bits. Furthermore, there are coding techniques that can almost achieve this lower bound (Witten et al., 1987).

Let (e_1, \dots, e_n) be a subsequence from an element of the language L . The sequence $c = (e_1, \dots, e_{n-1})$ is the context, and $e = e_n$ is the event to be predicted. The *entropy* of the language L , with respect to a probabilistic predictive theory T , is the minimal expected code length of an event. It can be estimated as

$$\frac{-\sum_{i=1}^n \log_2 p_T(e_i | c_i)}{n}, \quad (2)$$

where n is the number of subsequences used in the estimation. As n grows, statistically more reliable estimates of the entropy are produced. Theories which minimize the entropy estimate in Expression 2 are more predictive of the language under investigation.

Meyer (1957) has found a striking relationship between musical experience and entropy. Instability in music results from temporarily or permanently blocked expectations on the part of a listener. A piece of music which moves according to expectation is “neutral with respect to meaning,” where high expectation corresponds to high probability and low information. Conversely, high information may be the result of the delay of a consequent, or the ambiguity of an antecedent. Meyer claims that intraopus uncertainty is either *systemic*, where the particular work asserts its individuality, or *designed*, where the composer introduces elements of originality or surprise into a piece. These points of deviation are of interest in identifying important structural events in the music. In order to identify such instabilities, a predictive theory must be formed which creates expectations, so that their breakdown may be detected.

Predictive theories can also be used to generate new music. If a theory T minimizes

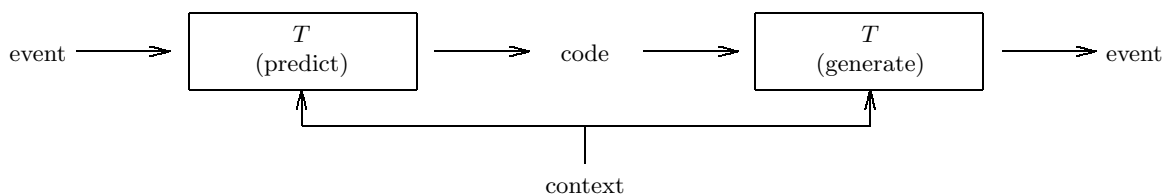


Figure 1: Shannon’s formulation of a predictive theory as a compressor of events.

the entropy estimate of the language, the codes produced will be incompressible, and hence highly random, bit strings. Thus in addition to evaluating a theory according to its predictive power, it is possible to inspect the music generated by a theory when provided with random codes.¹ A conjecture of this research is that highly predictive theories will also be good generative theories. Thus the goal of developing generative theories, which is a subjective process, can be replaced by the one of developing highly predictive theories. Furthermore, predictive theories of a musical style can be learned from examples of the style, as the next section will discuss.

There is an important point to note regarding the prediction scheme. Since $\log 0$ is undefined, it is necessary that $p_T(e | c)$ be non-zero for all contexts and events representable in the underlying symbol set. This means that, according to a theory T , no sequence of events is impossible, however unlikely it may be. Adopting Rahn’s (1989) term, we say that predictive theories of music must be *non-exclusive*. This means that a theory could conceivably generate any sequence of events.

Prediction is not the same thing as generation of music. In the above scheme, all predictive theories are generative. However, there are generative theories which are not predictive.

¹In the implementation, we generate music by random sampling from a cumulative probability distribution. This has the same effect as sending the generative theory random codes.

Consider, for example, various music composition techniques of *ars combinatoria* (Ratner, 1970), including musical dice games, fractal music, and so on. Even though these techniques may generate reasonable pieces, they are almost always exclusive and are not readily incorporated into a prediction scheme.

3 Machine learning of context models

Machine learning is concerned with improving performance at a specified task. Here the task is music prediction. This section presents the machinery we use to represent and learn about sequences. The power of this representation will be substantially enhanced in the next section, where multiple viewpoint systems are introduced. For this exposition, it is necessary to introduce a bit of notation. We will sometimes use the notation $\overline{e_n}$ as an abbreviation for a sequence (e_1, \dots, e_n) . The set of all representable events will be called the *event space*, and denoted by the symbol ξ . The set of all sequences that can be built from elements of a set S will be denoted by S^* . The catenation of an event e onto a sequence c will be denoted by $c :: e$. The cardinality of a set S will be denoted by $|S|$.

3.1 Context models

Here we describe *context models*, a subclass of the probabilistic finite-state, or Markov,² class of grammars. Context models have a conceptually simple induction procedure, and many modifications to their basic structure can be made without sacrificing ease of induction. A context model has three features: 1) a database of sequences over an event space, 2) a

²See Ames (1989) for a good overview of Markov modelling of music.

frequency count attached to each sequence, and 3) an inference method which is used to compute the probability of an event in context.

Deductive inference from a context model can be performed as follows. The conditional probability $p_T(e | c)$ of an event e given a context c is the number of times the sequence $c :: e$ occurs in the database divided by the number of times the context c occurs. The problem with this simple method is that the frequency of the context c could be 0, and $p_T(e | c)$ will then be undefined. An innovative solution to this problem is the partial match algorithm (Cleary and Witten, 1984). The conditional probability $p_T(e_n | (e_1, \dots, e_{n-1}))$ is computed by *blending* — computing a weighted linear combination of — the quantities

$$\begin{aligned}
 & p_T(e_n | (e_1, \dots, e_{n-1})), \\
 & p_T(e_n | (e_2, \dots, e_{n-1})), \\
 & \qquad \qquad \qquad \dots, \\
 & p_T(e_n | ()).
 \end{aligned}$$

Higher weights are given to terms appearing earlier in the above enumeration. It is possible that the conditional probability is still undefined if the last quantity above is undefined. This will occur when the one-event sequence (e_n) does not appear in the database. To solve this, the final probability measure is blended with $1/|\xi|$. The final result of blending is a probability distribution over all events in the event space ξ , given a context.

3.2 Induction of context models

Induction of context models is incremental. The initial theory of the concept is the most general theory possible, assigning equal probability to any tuple. Each example that arrives specializes the theory, since after incorporating the example, the theory will give higher probability to it. As such, induction of context models can be viewed as a hill-climbing search of a specialization hierarchy of probabilistic theories (Gennari et al., 1989; Buntine, 1988).

For a sequence \overline{e}_n comprising n events, the machine is given n (context, next event) tuples

$$\begin{aligned} & ((), e_1), \\ & ((e_1), e_2), \\ & ((e_1, e_2), e_3), \\ & \dots, \\ & ((e_1, \dots, e_{n-1}), e_n). \end{aligned}$$

Suppose an example $(\overline{e}_{n-1}, e_n)$ is given. Induction processes the sequence \overline{e}_n as follows. If \overline{e}_n is not in the database, add it, set its frequency count to 1, and recursively process the sequence (e_2, \dots, e_n) . If \overline{e}_n is in the database, increment its frequency count, and recursively process the sequence (e_2, \dots, e_n) . The operation terminates after the null sequence $()$ is processed. We call the first type of probabilistic specialization — the addition of a new sequence — *structural*, and the second type — the incrementing of frequency counts — *statistical*.

():10	(A):3	(AB) : 1	(ABA):1
	(G):4	(AG) : 1	(AGG):1
	(D):1	(GA) : 1	(GAB):1
	(B):2	(GG) : 2	(GAB):1
		(GD) : 1	(GGA):1
		(DB) : 1	(GDB):1
		(BA) : 2	(DBA):1
			(BAG):1

Table 1: A small context database after incorporation of the first ten pitch classes (sequence GGDBAGGABA) from chorale 1.

The database would quickly become unwieldy if we simply added to it every sequence that was encountered. This is because up to $n + 1$ sequences can be stored for a tuple $(\overline{e_{n-1}}, e_n)$. We impose the restriction that if a sequence is longer than a certain length h , it is not added. That is, for a tuple $(\overline{e_{n-1}}, e_n)$, only the sequences (e_{n-h+1}, \dots, e_n) , (e_{n-h+2}, \dots, e_n) , \dots , $()$ are processed. Nothing is done if $n < h$. We also use a trie data structure to store the sequences. A trie is a hierarchical data structure allowing very efficient access to sequence data.

As an example of the context model induction procedure, refer to Table 1, which shows the state of a context database (with $h = 3$) after incorporating the first ten pitch classes from chorale 1 (see Figure 4). This table simply presents an enumeration of database contents, and the set of sequences is actually represented much more efficiently using a trie.

The quantity $h - 1$ is called the *order* of the context model. A well-known problem of fixed-order context models is that very low order models are too general, and do not capture enough structure of the concept; very high order models are too specialized to the examples from which they were constructed, and do not capture enough statistics of the concept. The

induction scheme described above, using the partial match inference method, will tend to avoid such extreme overgeneralization or overspecialization.

3.3 Short and long term models

In sequence prediction using context models, two forces contribute to the sequence-generating rule. One is provided by long-term effects, and is governed by structure and statistics induced from a large corpus of sequences from the same genre. The other is provided by short-term effects, by structure and statistics particular to the sequence being predicted. This research explicitly represents the short and long-term forces as different context models. The short-term model is *transitory* in the sense that it is discarded after a particular sequence is predicted, and *dynamic* in the sense that it adapts to a particular sequence.

More precisely, suppose a tuple (c, e) is encountered. A short-term model of this tuple is a context model of the sequence c . Short-term models are also non-exclusive. The final probability of an event is a combination of its probability according to the short-term and long-term models. Figure 3 depicts the process of short/long term model combination.

4 Multiple viewpoint systems

The main problem with basic context models, which makes their use for music unsatisfactory, is that they require an exact match of supplied contexts to contexts in the database. They were not meant to deal with domains, such as music, where events have an internal structure (e.g., musical events have pitch, durations, and start-times) and are richly representable in languages other than the basic event language. To handle this problem, we introduce a

unique distributed problem solving method for sequences, called *multiple viewpoints*.

Multiple viewpoint systems address various weaknesses with standard context models. First, they are adaptive, in the sense that a model of a particular piece will change as that piece progresses. They support models which “look back” any number of events in a sequence to extract a context. The mathematical formalism of multiple viewpoints, described in this section, precisely guides the development of many different styles of context models.

4.1 Derived types

The central idea behind viewpoints is to use background domain knowledge to derive new ways of expressing events in a sequence. A *type* is an abstract property of events, such as scale degree or its melodic interval with its predecessor. For every type τ , there exists an associated partial function Ψ_τ which maps sequences of events to elements of type τ . We use the notation $[\tau]$ to denote the set of all syntactically valid elements of type τ , and thus $[\tau]^*$ is the set of all sequences representable using elements of type τ . A *viewpoint* comprises 1) a partial function $\Psi_\tau : \xi^* \rightarrow [\tau]$, and 2) a context model of sequences in $[\tau]^*$. For convenience, we will often refer to a viewpoint by the type it models, and we ask the reader to keep in mind that all viewpoints have an underlying context model. A collection of viewpoints forms a *multiple viewpoint system*.

The problem of context models has now been inverted; a system of viewpoints models no correlation between any basic types in an event. Such a system will have a limited representational and predictive power. A solution to this problem is to model interactions explicitly; the resulting model is called a *linked viewpoint*. The idea of linked viewpoints was

motivated by Lewin’s (1987) *direct product* systems.

A *product type* $\tau_1 \otimes \dots \otimes \tau_n$ between n constituent types τ_1, \dots, τ_n is itself a type τ , where elements of the product type are elements of the cross product of the constituents, that is, $[\tau] = [\tau_1] \times \dots \times [\tau_n]$. For a product type $\tau = \tau_1 \otimes \dots \otimes \tau_n$, $\Psi_\tau(\overline{e_k})$ is undefined if $\Psi_{\tau_i}(\overline{e_k})$ is undefined for any $i \in \{1, \dots, n\}$, else it is a tuple $\langle \Psi_{\tau_1}(\overline{e_k}), \dots, \Psi_{\tau_n}(\overline{e_k}) \rangle$.³

The complete space of product types forms a set partially ordered on the subset relation among constituents. Figure 2 displays the lattice of such a set for three primitive types τ_1 , τ_2 and τ_3 . On the bottom of the lattice is the empty type; on the top, the product type between all three primitive types. The second level of the lattice contains all 2-constituent product types, and the third level contains all primitive types. A multiple viewpoint system can be viewed as a set of points on this lattice. For example, $\{\tau_1, \tau_3\}$, $\{\tau_2, \tau_1 \otimes \tau_3\}$, and $\{\tau_2, \tau_1 \otimes \tau_2, \tau_3\}$ are multiple viewpoint systems.

For a system of n primitive types, the number of possible primitive multiple viewpoint systems that can be formed is $O(2^n)$. Once linked viewpoints with any number of constituents are allowed, this number increases to $O(n^n)$. A heuristic, such as degree of correlation of constituent viewpoints, might be used to guide the search for the best possible system of linked viewpoints. This constructive formation of new models during the learning process is recognized as one of the harder problems in sequence learning (Dietterich and Michalski, 1986), and will not be explored in this paper.

We supply a multiple viewpoint system with an informal semantics. Whereas $[\tau]$ represents the set of syntactically valid elements of type τ , $\llbracket \tau \rrbracket$ represents the semantic domain

³For convenience, if a product type has only one constituent, the angled brackets around a tuple will be omitted.

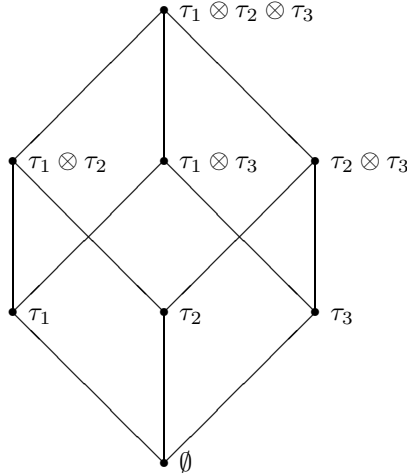


Figure 2: The lattice of product types for three primitive types.

of interpretation, or the set of possible meanings for type τ . For primitive types, this is a set. For a product type $\tau_1 \otimes \dots \otimes \tau_n$, $[[\tau_1 \otimes \dots \otimes \tau_n]]$ is a set of tuples $[[\tau_1]] \times \dots \times [[\tau_n]]$. The notation $[[\cdot]]_\tau$ denotes the meaning of a typical element from $[\tau]$, that is, $[[\cdot]]_\tau$ is a function from $[\tau]$ to $[[\tau]]$. We will leave off the subscript τ when it is evident from the context.

A sequence can be viewed as a set of derived sequences, one for each primitive type used in a multiple viewpoint system. This set of derived sequences is represented using a structure called a *solution array* (Ebcioğlu, 1986). A solution array for n primitive viewpoints τ_1, \dots, τ_n and a basic event sequence \overline{e}_k is an $n \times k$ matrix, where location (i, j) holds the value $\Psi_{\tau_i}(\overline{e}_j)$, or the symbol \perp if $\Psi_{\tau_i}(\overline{e}_j)$ is undefined. Product types do not need a row in the matrix, as they can be derived from their constituent rows. Section 5 will present and discuss the solution array for a Bach chorale.

4.2 Inference using viewpoints

Given a context c and an event e , each viewpoint τ in a multiple viewpoint system must compute the probability $p_\tau(e | c)$. This cannot be done directly, as a viewpoint τ is a context model over sequences in $[\tau]^*$, not in ξ^* . Thus it is first necessary to convert the surface string $c :: e$ in ξ^* to a string in $[\tau]^*$. The viewpoint then predicts a distribution over $[\tau]$, using the inference method outlined in Section 3.1. The conversion of the surface string is done using the Φ_τ function, where $\Phi_\tau : \xi^* \rightarrow [\tau]^*$ is defined inductively as

$$\begin{aligned} \Phi_\tau(()) &= (), \\ \Phi_\tau(\overline{e_k}) &= \begin{cases} \Phi_\tau(\overline{e_{k-1}}) :: \Psi_\tau(\overline{e_k}) & \Psi_\tau(\overline{e_k}) \text{ defined,} \\ \Phi_\tau(\overline{e_{k-1}}) & \text{otherwise.} \end{cases} \end{aligned}$$

Computing the probability of an event using a viewpoint is a bit more complicated than for normal context models, because the mapping Φ_τ is, in general, many-to-one. That is, the sequence $\Phi_\tau(c :: e)$ could represent many sequences of events other than the sequence $c :: e$, and the probability must be divided by the number of all such sequences.

Inductive inference of a multiple viewpoint system is similar to induction of regular context models. Suppose an example (c, e) is seen. For all viewpoints τ , if $\Psi_\tau(c :: e)$ is defined, the sequence $\Phi_\tau(c :: e)$ is added to the database for the viewpoint, in exactly the manner described in Section 3.2. It is necessary to make the check that $\Psi_\tau(c :: e)$ is defined, since otherwise the sequence in $[\tau]^*$ could be incorporated into the database multiple times for the same example. This follows from the definition of Φ_τ given above; if $\Psi_\tau(\overline{e_k})$ is undefined, then $\Phi_\tau(\overline{e_k}) = \Phi_\tau(\overline{e_{k-1}})$.

4.3 Inference using multiple viewpoints

Figure 3 summarizes the architecture of a multiple viewpoint system. This architecture is certainly not the only one possible, but is the one chosen for this study. The final probability of an event, given a context, is a function of many independent context models. On the left part of the figure, all short term models for every viewpoint combine into a prediction: on the right side all long term models. At the bottom part of the figure, predictions from the short term and long term models are combined into a final prediction.

There are numerous ways to combine predictions from viewpoints and short/long term models, including weighted linear combinations (Hamburger, 1986), Dempster-Schafer theory (Garvey et al., 1981) and fuzzy set theory (Dubois et al., 1992). We do not claim to have found a fully satisfactory solution, but have used a weighting scheme for viewpoints, and a Dempster-Schafer scheme to combine short/long term models, with reasonable success. The weighting given to a viewpoint is designed so that viewpoints that are very uncertain about the outcome are given lower weight. Uncertainty is measured using Shannon's entropy function. A viewpoint combination approach which also seems to work reasonably well first converts the viewpoint distributions to ranked lists, combines these rankings, and finally transcribes the ranked list back into a probability distribution. Further research on context model combination schemes is necessary.

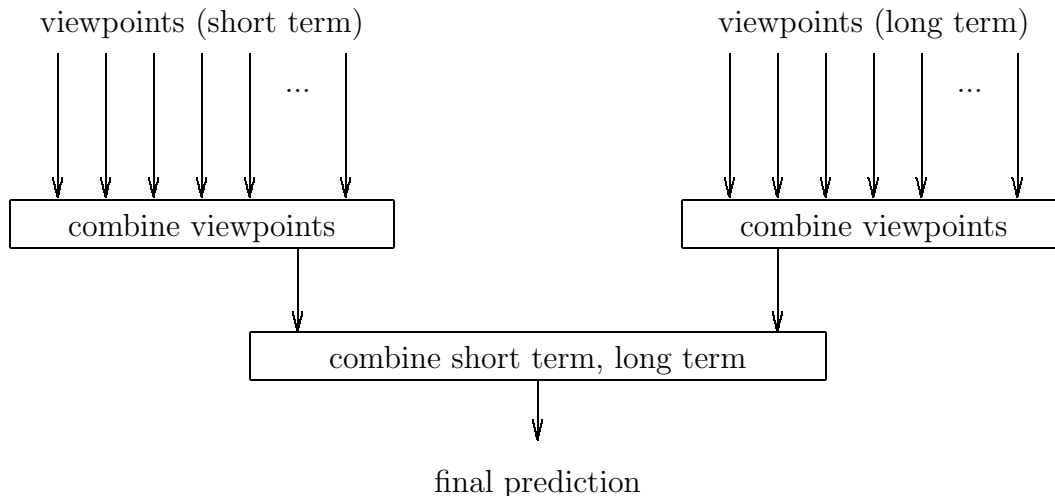


Figure 3: The architecture of a multiple viewpoint system.

5 Musical viewpoints

As an example of all types that will be discussed in this section, refer to Figure 4, which shows the first two phrases from chorale 1, and the solution array (see Section 4) for the fragment. Table 3 shows some applications of the functions Ψ and Φ (see Section 4) using some of the types discussed below and the chorale fragment of Figure 4. Note that the result of Ψ_τ is an element of $[\tau]$, whereas the result of Φ_τ is an element of $[\tau]^*$.

5.1 Basic types

A chorale is represented as a discrete event sequence, that is, all events in a chorale have discrete types, start-times and durations. Time and key signatures are included as basic types of an event. This choice ensures that no special circumstances have to be constructed for their prediction. It ensures homogeneity, and allows key and time signature to be linked

with other types, using the viewpoint linking technique described in Section 4.1.

The event space for the chorales is:

$$[\text{pitch} \otimes \text{keysig} \otimes \text{timesig} \otimes \text{fermata} \otimes \text{st} \otimes \text{duration}],$$

where `pitch` is the pitch of the event, `keysig` and `timesig` are the key signature and time signature of the chorale, `fermata` is a boolean type that indicates whether an event is under a fermata, `st` is the start-time, and `duration` is the duration of the event

The fundamental unit of time is the sixteenth note; all start-times and durations are measured as multiples of this unit. The longest value in `[[duration]]` is a whole note. The start-time of any chorale is 0, the zero time point representing the beginning of the first bar, complete or incomplete, in the chorale. Due to upbeat effects, the first event in a chorale may have a non-zero start-time (as in the chorale in Figure 4). The semantic domain of interpretation `[[pitch]]` ranges from $C\flat_4$ (middle C) to $G\sharp_5$ (19 semitones above middle C). Pitches are integer-encoded in a twelve tone system. The integer representation for pitch is the MIDI standard — `[pitch]` ranges from 0 to 127, `[[60]] = C\flat_4`, `[[72]] = C\sharp_5`, and so on. As a result, there is no notion of enharmonic inequivalence; for example, $C\sharp_5$ is semantically equivalent to $D\flat_5$. Rests are not events; they are modelled by a difference in time between the end of one event and the start of the next, as discussed in more detail below. Repeated sections are not expanded. Ties over a bar line are also not explicitly represented; only one event with the cumulative duration is encoded.

The key signature only states how many sharps or flats the chorale has, and says nothing about its mode or tonic. Since sharps and flats cannot occur together in the signature, and

their number cannot exceed 7, the key signature can be uniquely encoded as a number in the set $\{-7, \dots, 7\}$: $\llbracket -7 \rrbracket$ means “seven flats”, and $\llbracket 7 \rrbracket$ means “seven sharps”, and so on. The time signature is measured in terms of sixteenth notes per bar; for example, $\llbracket 12 \rrbracket = 3/4$ time, and $\llbracket 16 \rrbracket = 4/4$ time. Furthermore, $\llbracket v \rrbracket$ means that bar lines occur at multiples of v time units.

Information about phrases is notated in a consistent manner throughout all chorales using fermatas. This information provides very strong clues about the properties of the next event — for example, many phrases in the chorales begin on $\hat{1}$, $\hat{3}$, or $\hat{5}$, few end on $\hat{7}$, and so on. Phrase endings are represented on the score by means of a fermata symbol. These symbols can be represented by a boolean type `fermata`: $\llbracket T \rrbracket$ means that an event is under a fermata. The beginning of a phrase is assumed to be the event immediately following an event under a fermata.

The syntactic domain of any basic type must contain all elements of the type that could be encountered in the chorales. The syntactic domains were discovered by a simple analysis of 100 chorales. For example, the syntactic domain `[duration]` is not $\{1, \dots, 16\}$ but $\{1, 2, 3, 4, 6, 8, 12, 16\}$, and only 9 of the possible 15 key signatures are actually encountered in the chorale melodies.

5.2 Derived types

About twenty different derived types for the chorales have been implemented. This section presents these derived types, categorizing them according to the primary basic type from which they are derived. Table 2 summarizes all derived types that will be encountered. The

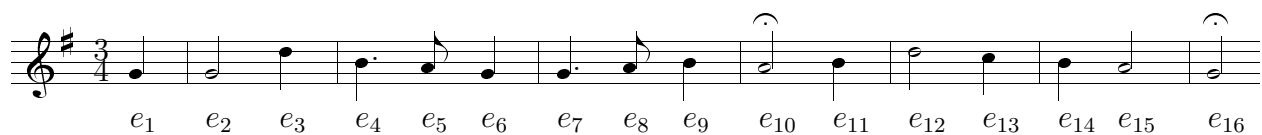
τ	$[[\cdot]]_{\tau}$	$[\tau]$	Derived from
st	start-time of event	$\{0, 1, 2, \dots\}$	st
pitch	pitch, in $\{C\sharp 4, \dots, G\sharp 5\}$	$\{60, \dots, 79\}$	pitch
duration	quarter note, eighth note, etc.	$\{1, 2, 3, 4, 6, 8, 12, 16\}$	duration
keysig	1 sharp, 1 flat, etc.	$\{-4, \dots, 4\}$	keysig
timesig	3/4 time, 4/4 time	$\{12, 16\}$	timesig
fermata	event under / not under fermata?	$\{T, F\}$	fermata
deltast	rest, no rest	$\{0, 4\}$	st
gis221	difference in start-time	$\{1, \dots, 20\}$	st
posinbar	position of event in bar	$\{0, \dots, 15\}$	st
fib	first / not first in bar	$\{T, F\}$	st
seqint	sequential melodic interval	Z	pitch
contour	rising, falling, static	$\{-1, 0, 1\}$	pitch
referent	referent of piece	$\{0, \dots, 11\}$	keysig
intfref	vertical interval from referent	$\{0, \dots, 11\}$	pitch
inscale	in / not in scale	$\{T, F\}$	pitch
intfib	interval from first event in bar	$[\text{seqint}]$	pitch
intfip	interval from first event in piece	$[\text{seqint}]$	pitch
intphbeg	interval from phrase beginning	$[\text{seqint}]$	pitch
thrbar	seqint at bars	$[\text{seqint}] \times Z^+$	pitch, st
lphrase	length of phrase	Z^+	fermata, st
thrph	seqint at phrases	$[\text{seqint}] \times Z^+$	pitch, st
thrqu	seqint at quarters	$[\text{seqint}] \times Z^+$	pitch, st

Table 2: The basic and some primitive derived types for the chorales.

first column of the table gives the symbolic name of the type. The second column informally gives its semantic valuation function. The third column details the syntactic domain of the type, and the last column shows the basic types the type is derived from, and hence is capable of predicting. The symbol Z^+ denotes the positive integers $\{1, 2, 3, \dots\}$. The top part of Table 2 shows all basic types. The bottom part shows some *threaded* derived types. These types inspect a variable number of previous events in a chorale. Not all types of Table 2 will form useful viewpoints; some are used only to simplify the expression of others.

5.2.1 Start-time

If all events in a sequence followed each other immediately, there would be no need for a **st** type; it could be calculated if necessary by summing the durations of all previous events.



Type	Event number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
st	8	12	20	24	30	32	36	42	44	48	56	60	68	72	76	84
pitch	67	67	74	71	69	67	67	69	71	69	71	74	72	71	69	67
duration	4	8	4	6	2	4	6	2	4	8	4	8	4	4	8	8
keysig	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
timesig	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
fermata	F	F	F	F	F	F	F	F	F	T	F	F	F	F	F	T
deltast	⊥	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
gis221	⊥	4	8	4	6	2	4	6	2	4	8	4	8	4	4	8
posinbar	8	0	8	0	6	8	0	6	8	0	8	0	8	0	4	0
fib	F	T	F	T	F	F	T	F	F	T	F	T	F	T	F	T
seqint	⊥	0	7	-3	-2	-2	0	2	2	-2	2	3	-2	-1	-2	-2
contour	⊥	0	1	-1	-1	-1	0	1	1	-1	1	1	-1	-1	-1	-1
referent	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
intfref	0	0	7	4	2	0	0	2	4	2	4	7	5	4	2	0
inscale	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
intfib	0	0	7	0	-2	-4	0	2	4	0	2	0	-2	0	-2	0
intfip	0	0	7	4	2	0	0	2	4	2	4	7	5	4	2	0
intphbeg	0	0	7	4	2	0	0	2	4	2	0	3	1	0	-2	-4
thrbar	⊥	0 ₄	⊥	4 ₁₂	⊥	⊥	-4 ₁₂	⊥	⊥	2 ₁₂	⊥	5 ₁₂	⊥	-3 ₁₂	⊥	-4 ₁₂
lphrase	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	40	⊥	⊥	⊥	⊥	⊥	28
thrph	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	4 ₄₈	⊥	⊥	⊥	⊥	⊥
thrru	⊥	⊥	7 ₈	-3 ₄	⊥	-4 ₈	0 ₄	⊥	4 ₈	-2 ₄	2 ₈	3 ₄	-2 ₈	-1 ₄	-2 ₄	-2 ₈

Figure 4: Solution array, chorale 1, phrases 1 and 2, for a collection of basic and primitive derived types.

Function	Parameter	Result
Ψ_{st}	$\overline{e_7}$	36
Ψ_{pitch}	$\overline{e_{12}}$	74
$\Psi_{\text{intfref} \otimes \text{seqint}}$	$\overline{e_{10}}$	$\langle 2, -2 \rangle$
$\Psi_{\text{gis221} \otimes \text{seqint}}$	$\overline{e_1}$	undefined
$\Psi_{\text{gis221} \otimes \text{seqint}}$	$\overline{e_{13}}$	$\langle 8, -2 \rangle$
$\Psi_{\text{intfref} \otimes \text{fib}}$	$\overline{e_7}$	$\langle 0, \text{T} \rangle$
Φ_{st}	$\overline{e_7}$	(8, 12, 20, 24, 30, 32, 36)
$\Phi_{\text{gis221} \otimes \text{seqint}}$	$\overline{e_1}$	()
$\Phi_{\text{gis221} \otimes \text{seqint}}$	$\overline{e_7}$	($\langle 4, 0 \rangle, \langle 8, 7 \rangle, \langle 4, -3 \rangle, \langle 6, -2 \rangle, \langle 2, -2 \rangle, \langle 4, 0 \rangle$)

Table 3: Some example applications of the functions Ψ and Φ .

In music, however, we must deal with the phenomenon of the rest: although these are not common in the chorales, they are nonetheless present. For the `deltast` type, $\llbracket v \rrbracket$ means that the difference in time between the start-time of an event and the end-time of its predecessor is v time units. The `gis221` viewpoint (after Lewin’s, 1987, generalized interval system 2.2.1), assumes a value that is the difference in start-time between an event and its predecessor. For the `gis221` viewpoint, $\llbracket v \rrbracket$ means that the difference between the start-time of an event and the start-time of its predecessor is v time units. Note that the phenomena modelled by `gis221` cannot be captured by a product type `duration` \otimes `deltast`.

Another useful way to characterize the start-time of an event is by position in the bar. This is the “position in the bar” (`posinbar`) type: $\llbracket v \rrbracket$ means that the start-time of an event relative to the “start-time” of the bar is v time units. By linking it with some other type, we can capture metrical and position-dependent effects of that type (`posinbar` itself is expressed using the time signature). The “first in bar” type (`fib`) takes on boolean values: $\llbracket \text{T} \rrbracket$ means that an event has a `posinbar` value equal to 0. In long sequences it can model effects such as “number of notes per bar,” and can usefully be linked with `timesig`.

The `fib` type is used in the expression of the “threaded bar” (`thrbar`) type. This is an

example of a *threaded* type: its values are only defined at certain points throughout a chorale. For an event with the `posinbar` value equal to 0, $\llbracket a_b \rrbracket$ means that the melodic interval (next section) between its pitch and the pitch of first note of the previous bar is a . The quantity b is the *timescale* of the value; it is the difference in start-time between the two events. Note that the timescale varies and is not always the same as the time signature; the first event in the previous bar need not have a `posinbar` value of 0. This is the first example of a type which is derived from two basic types — `pitch` and `st`.

A threaded type similar to `thrbar`, the `thrqu` type, works as follows. If an event falls on a quarter note pulse, $\llbracket v \rrbracket$ means that the interval between it and the latest earlier event that falls on a quarter note pulse is v . Note that this type also possesses a timescale; for example, event e_6 of the chorale in Figure 4 occurs 8 time units after the latest earlier event that falls on a quarter note pulse (event e_4).

Another useful way to characterize the start-time of an event is by position in the bar. This is the “position in the bar” (`posinbar`) type: $\llbracket v \rrbracket$ means that the start-time of an event relative to the “start-time” of the bar is v time units. By linking it with some other type, we can capture metrical and position-dependent effects of that type (`posinbar` itself is expressed using the time signature). The “first in bar” type (`fib`) takes on boolean values: $\llbracket T \rrbracket$ means that an event has a `posinbar` value equal to 0. In long sequences it can model effects such as “number of notes per bar,” and can usefully be linked with `timesig`.

5.2.2 Pitch

The abstract mathematical properties of twelve-tone pitch systems are very important to this study because they facilitate the construction of a rich set of derived types. The basic tenet of these systems is that pitch is discrete, corresponding to the equal-tempered semitone scale. Absolute points on the scale may be of interest in some musical theories; in the chorales, however, *relative* pitch is more important.

Lewin (1987) has noted that if a binary operation $+$ induces a mathematical group on a set of musical interval measures $[\tau]$, interesting constructive properties emerge, based on the notion of a *quotient group*. The axioms of a mathematical group $([\tau], +)$ are that 1) the combination of any interval with the interval 0 yields the original interval, 2) for every interval, there exists an inverse which “undoes” its effect, and 3) the operation $+$ is associative. If these axioms hold, it is easy to derive new musical interval measures from a group, given a congruence on $([\tau], +)$. We omit the details; see (Lewin, 1987). The most basic metric is the familiar type `seqint`: $\llbracket \text{seqint} \rrbracket$ is the domain of twelve-tone melodic intervals, and $[\text{seqint}] = Z$, the set of all integers.

There is another interesting descriptor of `pitch`. This is the `contour` type, which measures whether an event presents a rise, fall, or no motion from its predecessor. Pitch contour is an extremely important component of melody. Analytical studies have, for the most part, subordinated contour to other melodic descriptors. Recently, however, the study of contour has gained prominence (Marvin and Laprade, 1987).

So far, all types derived from `pitch` have depended only on relations in a sequential context. Simultaneously, there also exists a harmonic or *vertical* context in terms of the

scale degrees of events as they move through time. As mentioned earlier, however, only the key signature is encoded, and information about the tonic of a piece is not available. Some reasonable approximation to scale degree is necessary. The `referent` type represents the *tonic* of the *major* mode that has a particular key signature; note that `referent` is derived from `keysig`. The referent can be viewed as an unchanging “drone bass.” For the “interval from referent type (`intfref`), $\llbracket v \rrbracket$ means that the interval, modulo 12, between the pitch of an event and the referent is v . The `inscale` type takes on boolean values: $\llbracket T \rrbracket$ means that an event is in a major mode scale constructed on the referent.

Two long-range types derived from `pitch` are included in Table 2. The first is the “interval from first in bar” (`intfib`) type: $\llbracket v \rrbracket$ means that the interval between the first event in a bar and a given event is v . The interval from the first event in the bar and itself is defined to be 0. The second is “interval from first in piece,” (`intfip`): $\llbracket v \rrbracket$ means that the interval between an event and the first event of the piece is v . There is an interesting sort of symmetry between this type and `intfref`; `intfip` views the first note in the piece as the “drone,” and a link between the two may help alleviate the problems with modes discussed above.

5.2.3 Duration

There are few expressive ways to talk about the duration of an event in a chorale other than absolute duration. Lewin (1987) proposes a handful of interval systems that might be considered. One, for example, specifies that one note lasts for a certain fraction of the duration of another. This type can expressively model phenomena such as augmentation of fugue subjects. It is felt that although Lewin’s systems for duration may be useful constructs

for certain musics, their applicability to the chorales is debatable. The set of durations used in the chorales is quite small, and absolute durations are used in a very consistent manner. Although no types are derived from `duration`, linking the basic type `duration` with others gives useful abstractions. For example, a linked viewpoint `duration` \otimes `intfref` can capture the relative durations of various referent-degrees, and a linked viewpoint `duration` \otimes `fermata` can model the duration of events ending phrases.

5.2.4 Fermata

A *periodic* threaded type threads a sequence at periods of a fixed number of events. The predictions of fermatas gives a perfect example of a situation where simple periodic threaded types are unsatisfactory. The non-periodic “length of phrase” (`lphrase`) type is derived from both `fermata` and `st`. If an event ends a phrase, $\llbracket v \rrbracket$ means that its start-time minus the start-time of the first event in that phrase is v time units. Phrase lengths become even more redundant with knowledge of the time signature, pointing towards another potentially useful product type `lphrase` \otimes `timesig`.

The “threaded phrase” type is a sort of hybrid, where if an event starts a phrase, $\llbracket v \rrbracket$ means that the interval between the first note of the previous phrase is v . It is derived from neither start-times nor fermatas, but threads the piece at non-periodic timescales, and is derived from `pitch`. Finally, for the “interval from phrase beginning” (`intphbeg`) type, $\llbracket v \rrbracket$ means that the interval between an event and the first event in the phrase containing it is v . This type may be linked with `contour`, for example, and sequences of the product type would detail the exact contour of a phrase.

6 Experimental results

A machine learning system called SONG/3 (Stochastically Oriented Note Generator)⁴ based on multiple viewpoints has been implemented, including about twenty different viewpoints. One hundred Bach chorales from *The 371 Four-Part Chorales* (Edition Mainous and Ottman) were encoded using the event space described in Section 5. These were divided into 1) a randomly chosen test set comprising 5 chorales, and 2) a training set comprising the other 95. The training set comprises about 4500 (context, next event) tuples: the test set slightly over 200. In the experiment described here, we restrict our attention to the basic type `pitch`. That is, the machine is given the rhythmic skeleton, key signature, time signature, and position of the fermatas for every test chorale.

Table 4 shows some results that were obtained. Each row of the table represents an experiment. The second column defines the multiple viewpoint system used. The orders of the short-term and long-term context models for every viewpoint are fixed at 2 and 3, respectively. These parameters were settled upon after some experimentation with a small training set: changes led to somewhat worse prediction performance. The third column of the table is the entropy estimate given by a particular system. This estimate is obtained as described in Section 3.2; a theory is induced from the training set, and the figure in the third column is the average number of bits required to specify `pitch`. This average is over all (context, next event) tuples in the test set.

⁴SONG/1 (described by Conklin and Cleary, 1988) was implemented in PROLOG, and only produced generative theories. When no common ground existed among viewpoints, backtracking to the last event generated took place. SONG/2 was also implemented in PROLOG, and only considered predictive theories. SONG/3, implemented efficiently in Common LISP, has a facility for generating music from predictive theories.

System	Viewpoints	Result
1	<code>pitch</code>	2.05
2	<code>seqint</code>	2.33
3	<code>seqint</code> \otimes <code>gis221</code>	2.13
4	<code>seqint</code> \otimes <code>gis221</code> , <code>pitch</code>	2.01
5	<code>intfref</code> \otimes <code>seqint</code>	2.12
6	<code>intfref</code> \otimes <code>seqint</code> , <code>seqint</code> \otimes <code>gis221</code>	1.94
7	<code>intfref</code> \otimes <code>seqint</code> , <code>seqint</code> \otimes <code>gis221</code> , <code>pitch</code>	1.92
8	<code>intfref</code> \otimes <code>seqint</code> , <code>seqint</code> \otimes <code>gis221</code> , <code>pitch</code> , <code>intfref</code> \otimes <code>fib</code>	1.87

Table 4: Average entropy of `pitch` for different multiple viewpoint systems.

Table 4 shows, for example, that the primitive system `{pitch}` can represent the basic type `pitch` using an average of 2.05 bits (system 1). The entropy of `pitch` with respect to the `{seqint}` system is 2.33 (system 2). According to the conjecture underlying this research, system 1 should generate better chorales than system 2. When the product type `seqint` \otimes `gis221` is used as a viewpoint, performance improves (system 3). This means that the types `seqint` and `gis221` are correlated in the chorales.

Systems 1 to 3 are single-viewpoint ones. The system `{seqint` \otimes `gis221`, `pitch}` contains two viewpoints (system 4). It is interesting that it performs better than either of its constituent viewpoints (systems 1 and 3). This means that the viewpoint combination method described in Section 4 is effective. Systems 3, 5, and 6 show a similar phenomena. Adding the viewpoint `pitch` to system 6 further improves it (system 7).

It was hypothesized that the scale degree is correlated with the fact that an event does or does not begin a bar. A product type `intfref` \otimes `fib` was added to system 7. The result

(1.92 bits decreases to 1.87 bits) shows that this hypothesis was correct (system 8). The low entropy estimate of 1.87 bits/pitch is interesting, as humans do not seem to do much better at the chorale prediction task (Witten et al., 1993): their entropy estimate is about 1.75 bits/pitch. However, this human estimate is probably unstable as it is averaged over only two test chorales. See (Witten et al., 1993) for a detailed comparison of the computational model presented here with human performance at the chorale prediction task.

A useful way to present the results of prediction is with an entropy profile, where the number of bits needed to specify an event (recall Expression 1) is plotted against the event number in a particular piece. Figure 5 shows the entropy profile for `pitch` for chorale 151, using the last system of Table 4. The peaks show which events the computer finds surprising and very hard to predict, and the troughs show the predictable events. The two high-entropy peaks of chorale 151 correspond to events 6 and 23. The machine predicted *B4* at event 6 with probability 0.97. This is most certainly due to the fact that the short-term viewpoint of `pitch` must revert to an zero-length context to make a prediction; and out of the five pitches so far, three of them have been *B4*. Event 23 is a leap of a minor seventh to begin a phrase. This is a highly unpredictable event, for both computers and people. The machine predicted *G4* at event 23 with probability 0.40: certainly a reasonable prediction.

The predictive theory was used to generate some new pieces. This was done by taking a context, generating an event (recall Section 2), then catenating that event to the context, producing a new context. This process is iterated a fixed number of times. Figure 6 shows one of the generations produced in this manner. The first seven events, and the complete rhythmic skeleton of chorale 151, were supplied to the system.

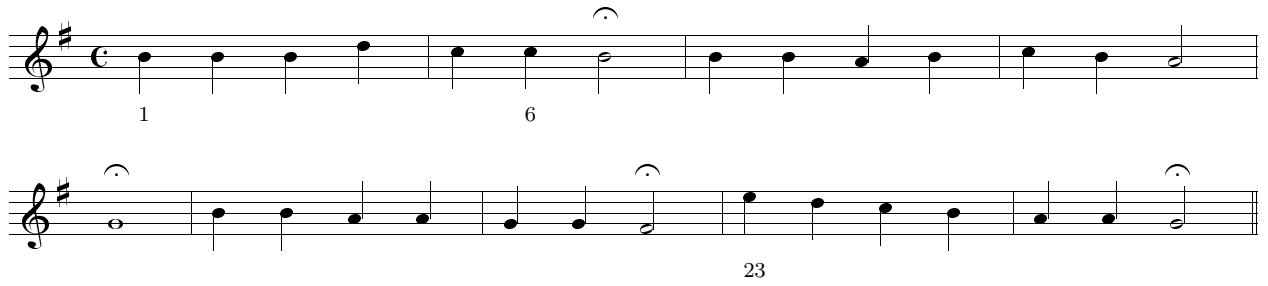
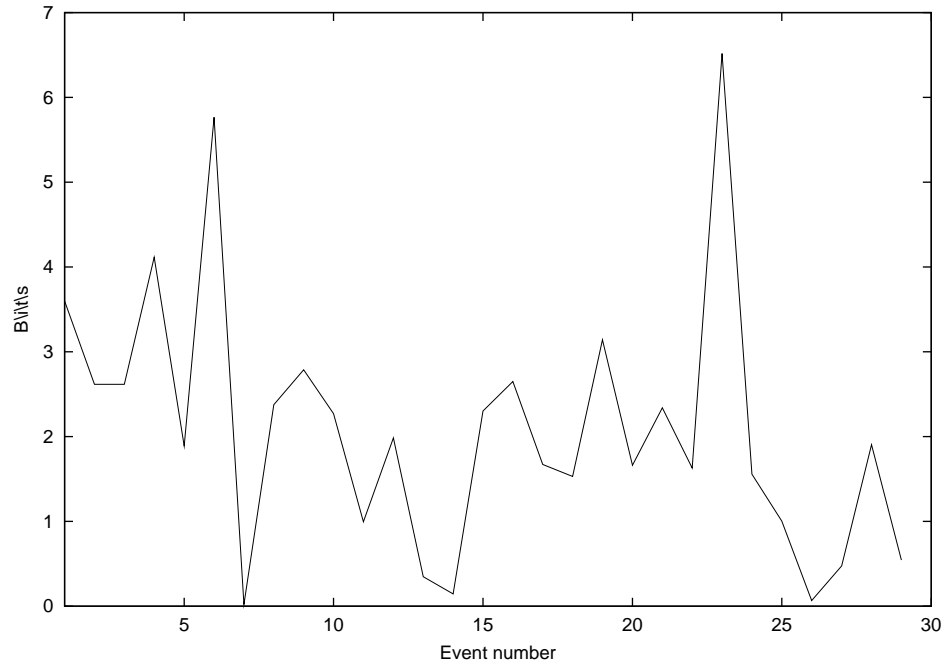


Figure 5: Bottom: chorale 151; Top: entropy profile, chorale 151.

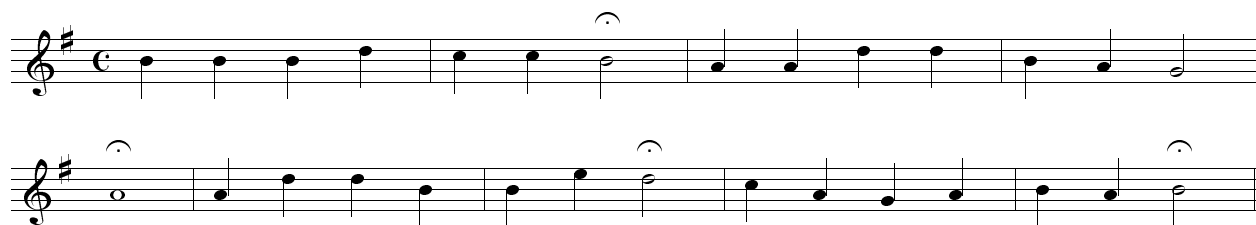


Figure 6: A piece generated by SONG/3. The first seven events of chorale 151 were supplied as a context.

7 Conclusion

This research was motivated by the goal of learning generative theories for the chorale genre. This goal was replaced by one of minimizing the estimated entropy of the genre, due to the conjecture that predictive power is a sufficient criteria for generative capability. The entropy was estimated by a process of analysis and prediction.

The generated chorale presented in the previous section seems to be reasonable, but we cannot claim to have fully answered the conjecture about predictive theories. Of course, theories with a very high measure will likely generate something close to white noise, and it is reasonable to minimize the estimated entropy — to some extent. What we might find, however, is that beyond a certain point further attempts at minimization may not be worth the effort — the gains in generative capability may be so slight and subtle as to be impossible to notice.

This work suggests several areas of future research on prediction and entropy of music. It should be possible to devise better multiple viewpoint systems for the chorales. The

multiple viewpoint formalism could be extended to deal with multiple voice music. With this extension, for example, complete four-part chorales could be predicted and generated. Much work is necessary to determine good alternative architectures and inference methods for multiple viewpoint systems. To be significant as a general-purpose machine learning tool for music, the system should be applied to musical domains wider and more adventurous than the chorale melodies, and should also be widened to include harmonic progressions.

Acknowledgements

Thanks to Neil MacGregor for pointing out (1999) an error (corrected here) in the blending equation in section 3.1 of the original JNMR article, and to Christina Anagnostopoulou for pointing out (2000) an error in Table 3.

References

- Ames, C. (1989). The Markov process as a compositional model — A survey and tutorial. *Leonardo* 22(2):175–187.
- Baroni, M. and Jacobini, C. (1978). *Proposal for a Grammar of Melody*. Les Presses de l’Université de Montréal.
- Bell, T. C.; Cleary, J. G.; and Witten, I. H. (1990). *Text Compression*. Prentice Hall.
- Brooks, F. P.; Hopkins Jr., A. L.; Neumann, P. G.; and Wright, W. V. (1956). An experiment in musical composition. *IRE Transactions on Electronic Computers* EC-5:175–182.
- Brown, M. and Dempster, D. J. (1989). The scientific image of music theory. *Journal of Music Theory* 33(1).
- Buntine, W. (1988). Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence* 36:149–176.
- Cleary, J. G. and Witten, I. H. (1984). Data compression using adaptive coding and partial string matching. *IEEE Trans. Communications* COM-32(4):396–402.
- Conklin, D. and Cleary, J. G. (1988). Modelling and generating music using multiple viewpoints. In *Proceedings of the First Workshop on AI and Music*. The American Association for Artificial Intelligence. 125–137.
- Cope, D. (1987). An expert system for computer-assisted composition. *Computer Music Journal* 11(4):30–46.
- Dietterich, T. G. and Michalski, R. S. (1986). Learning to predict sequences. In Michalski, R.; Carbonell, J.; and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach*, volume II. Morgan Kaufmann.
- Dubois, D.; Lang, J.; and Prade, H. (1992). Dealing with multi-source information in possibilistic logic. In Neumann, B., editor, *Proc. ECAI-92: Tenth European Conference on Artificial Intelligence*. John Wiley and Sons. 38–42.
- Ebcioğlu, K. (1986). *An Expert System for Harmonization of Chorales in the Style of J. S. Bach*. Ph.D. Dissertation, Department of Computer Science, SUNY at Buffalo.
- Garvey, T. D.; Lowrance, J. D.; and Fischler, M. A. (1981). An inference technique for integrating knowledge from disparate sources. In *Proc. International Joint Conference on*

Artificial Intelligence. 319–325.

Gennari, J. H.; Langley, P.; and Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence* 40:11–61.

Hamburger, H. (1986). Representing, combining, and using uncertain estimates. In Kanal, L. N. and Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence*. North–Holland. 399–414.

Hiller, L. (1970). Music composed with computers — a historical survey. In Lincoln, H. B., editor, *The Computer and Music*. Cornell University Press. chapter IV.

Kohonen, T. (1989). A self learning musical grammar. In *Proc. Int. Joint. Conf. on Neural Networks*, Washington, D.C., USA.

Lewin, D. (1987). *Generalized Musical Intervals and Transformations*. Yale University Press.

Lidov, D. and Gabura, J. (1973). A melody writing algorithm using a formal language model. *Computer Studies in the Humanities* 4(3-4):138–148.

Marvin, E. W. and Laprade, P. A. (1987). Relating musical contours: Extensions of a theory for contour. *Journal of Music Theory* 31(2):225–267.

Meyer, L. B. (1957). Meaning in music and information theory. *Journal of Aesthetics and Art Criticism* 15:412–424.

Rahn, J. (1989). Notes on methodology in music theory. *Journal of Music Theory* 33(1):143–154.

Ratner, L. (1970). Ars combinatoria: Chance and choice in Eighteenth–century music. In Landon, H. C. Robbins, editor, *Studies in Eighteenth Century Music*. Allen & Unwin.

Schwanauer, S. M. (1993). A learning machine for tonal composition. In Schwanauer, S. M. and Levitt, D. A., editors, *Machine Models of Music*. The MIT Press. 511–532.

Shannon, C. E. (1951). Prediction and entropy of printed english. *Bell System Technical Journal* 50–64.

Witten, I. H.; Neal, R.; and Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM* 30(6):520–540.

Witten, I. H.; Manzara, L. C.; and Conklin, D. (1994). Comparing human and computational models of music prediction. *Computer Music Journal* 18(1):70–80.