

Universal Prediction Applied to Stylistic Music Generation

Gérard Assayag (Ircam) , Shlomo Dubnov (Ben Gurion Univ.)

Abstract

Capturing a style of a particular piece or a composer is not an easy task. Several attempts to use machine learning methods to create models of style have appeared in the literature. These models do not provide an intentional description of some musical theory but rather use statistical techniques to capture regularities that are typical of certain music experience. A standard procedure in this approach is to assume a particular model for the data sequence (such as Markov model). A major difficulty is that a choice of an appropriate model is not evident for music. In this paper, we present a universal prediction algorithm that can be applied to an arbitrary sequence regardless of its model. Operations such as improvisation or assistance to composition can be realised on the resulting representation.

1. Introduction

Machine learning is the process of deriving a set of rules from data examples. Being able to construct a music theory from examples is a great challenge, both intellectually, and as a means for a whole range of new exciting applications. Such models can be used for analysis and prediction, and, to a certain extent, they can generate acceptable original works that imitate the style of their masters, recreating a certain aspect of music experience that was present in the original data set. The process of composition is a highly structured mental process. Although it is very complex and hard to formalise, it is not completely random. The task of this research is to try to capture some of the regularity apparent in the composition process by applying information theoretic tools to this problem.

Mind-reading Machines

In early 50's at Bell Labs David Hagelbarger has built a simple 8 state machine, whose purpose was to play the "penny matching" game. The simple machine tried to match the future choices of a human player over a long sequence of random "head" or "tail" choices. Mind-reading was done by looking at similar patterns in opponent's past sequence that would help predict the next guess. The achieved rate of success was greater than 50%, since human choices could not be completely random and analysing patterns of previous choices could help foretell the future.

Inspired by Hagelbarger's success, Shannon has built a different machine with improved performance. An account of Shannon's philosophy on mind-reading machines can be found in [Shannon 1953].

It is important to note that if the model of the data sequence was known ahead of time, an optimum prediction could be achieved. The difficulty with most real situations is that the probability model for the data is unknown. Therefore one must use a predictor that works well no matter what data model is. This idea is called "universal prediction".

Music Generation and Style Replication

Generative theory of music can be constructed by explicitly coding music rules in some logic or formal

grammar (Cope 1987, Ebicoglu 1986, Lidov & Gambura 1973). This approach is sometimes called "expert system" or "knowledge engineering approach". A contrasting approach is the statistical learning or empirical induction approach. Several researchers have used probabilistic methods, notably Markov models, to model music (Pinkerton 1956, Brooks Jr. et al. 1993, Conklin and Witten 1995). Pinkerton used a small corpus of diatonic major-key nursery rhyme to learn a Markov model, which he later used to generate nursery rhymes. Because he used a small alphabet (seven symbols of the diatonic scale and a tied note symbol), he was able to use a high-order (long context) Markov model up to order eight. Conklin and Witten (1995) used trigrams¹ to generate chorale melodies from parameters on a corpus of Bach chorale melodies. A more recent Markov model experiment was done by Brooks Jr. et al. (1993). Like Conklin and Witten, 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). To capture similarities between pieces in different keys (but the same mode), all pieces were into C. The experiment showed that at very low orders (e.g., unigram), generated strings do not recognisably resemble strings in the corpus, while at very high orders, strings from the corpus are just replicated.

An interesting "compromise" between the two approaches is found in more recent works of Cope (1991). Cope uses grammatical generation system combined with what he calls "signatures", 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.

Predictive Theories in Music

Following the work of Meyer (1957) it is commonly admitted that musical perception is guided by expectations based on the recent past context.

¹ An n-gram is a sequence of symbols of length n. The first n-1 of these are the context.

Predictive theories are often related to specific stochastic models which estimate the probability for musical elements to appear in a given musical context, such as Markov chains mentioned above. If one is dealing with a data sequence whose probabilistic model is known, then one can optimally predict the next samples in the sequence. If one does not know the model, there are two possible solutions. One is to estimate the model first and use it for prediction. The second approach is to use a predictor that works well for every model or at least works as good as any other predictor from a limited class of prediction methods.

In music applications the model is unknown. Considering the context or the past samples for prediction, one of the main problems is that the length of musical context (size of memory) is highly variable, ranging from short figurations to longer motifs. Taking a large fixed context makes the parameters difficult to estimate and the computational cost grows exponentially with the size of the context. In order to cope with this problem one must design a predictor that can deal with arbitrary observation sequence and is competitive to a rather large class of predictors, such as Finite State Machine Predictors and Markov Predictors. Philosophically, we take an agnostic approach: do the best we can relative to a restricted class of strategies.

Finite State Prediction

In order to describe the theory of prediction for a completely arbitrary data model we need to define the concept of finite-state predictor. Let us define a set S and two functions $f : S \times A \rightarrow A$ and $g : S \times A \rightarrow S$, such that the predictions \hat{x}_i for a sequence x_1, x_2, \dots, x_n are generated by the following mechanism:

$$\begin{aligned}\hat{x}_i &= f(s_i) \\ s_i &= g(s_{i-1}, x_{i-1})\end{aligned}$$

The initial state s_0 is given as well. In the finite state (FS) predictor the predicted value depends only on the current state s_i according to the prediction function f . For each new observation the machine moves to a new state according to the transition rule g . The error between a sequence of predictions and the actual data is defined by

$$d_n(x_1^n, \hat{x}_1^n) = n^{-1} \sum_{i=1}^n \delta(x_i, \hat{x}_i)$$

where $\delta(x, \hat{x})$ is the error count, i.e a Hamming distance function that equals 0 iff $x = \hat{x}$ and 1 otherwise. The minimal fraction of errors for an S-state predictor is called ‘‘S-state predictability’’ and is denoted by $\pi_S(x_1^n)$. If we want to consider the performance of FS predictor for increasing S, the

length of the sequence must be increased. Growing n first and S second, FS predictability is defined as

$$\pi(x) = \lim_{s \rightarrow \infty} \limsup_{n \rightarrow \infty} \pi_s(x_1^n).$$

FS predictors are examined in detail in Feder et al. (1992). They consider the problem of constructing a universal predictor that performs as well as any finite state predictor. By definition, $\pi(x)$ depends on the particular sequence x . The surprising result is that a sequential predictor can be found that does not depend on x and yet achieves asymptotically FS predictability $\pi(x)$. Similarly, when the class of FS predictors is further confined to Markov predictors² then the corresponding prediction performance measure is called Markov predictability. It is further shown by Feder et al. (1992) that the finite-state predictability and the Markov predictability are always equivalent, which means that is is sufficient to confine attention to markov predictors in order to achieve the finite-state predictability. For a treatment of nonparametric universal prediction theory the reader is invited to consult also additional references Blackwell (1954), Hannan (1957).

In our work we present a dictionary-based prediction method, which parses an existing musical text into a lexicon of phrases/patterns, called *motifs*, and provides an inference method for choosing the next musical object following a current past *context*. The parsing scheme must satisfy two conflicting constraints. On the one hand, one wants to maximally increase the dictionary to achieve better prediction, but on the other hand, enough evidence must be gathered before introducing a new phrase, so that a reliable estimate of the conditional probability is obtained. The secret of dictionary-based prediction (and compression) methods is that they cleverly sample the data so that most of the information is reliably represented by few selected phrases. This could be contrasted to Markov models that build large probability tables for the next symbol at every context entry. Although it might seem that the two methods operate in a different manner, it is helpful to understand that basically they employ similar statistical principles.

Predictability and Compression

The preceding discussion might seem needlessly complicated to someone current in compression and coding methods. It is widely known that prediction serves as the basis for modern data compression and it seems just natural that an opposite analogy would exist, i.e. a good compression method would be also useful for a good predictor. A standard measure for compression quality is coding redundancy or how close the entropy of the coded sequence approaches the entropy of the data source. Intuitive link between

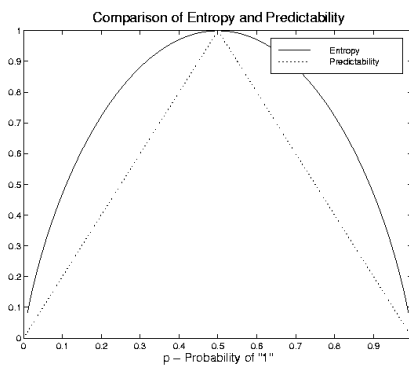
² Markov predictor of order k is FS predictor with 2^k states where $s_i = (x_{i-k}, \dots, x_{i-1})$.

predictability and entropy is easy to establish. Entropy (also sometimes called "uncertainty") measures the minimal number of bits needed to describe a random event. For a completely random, i.i.d binary sequence, one must transmit all bits in order to describe the sequence. If the probability for ones is greater than for zeros (or vice versa), one can devise a scheme where long sequences of ones are assigned to short codewords, thus saving on the total number of bit, i.e. achieving on the average less than one bit per symbol. The entropy function $H(p)$ for a sequence with probability p to see "1" is given by

$$H(p) = -\{p \log p + (1 - p) \log(1 - p)\}$$

Predictability on the other hand measures the minimum fraction of errors that can be made by some prediction machine over long data sequences. For instance, optimal single state predictor employs counts $N_n(0)$ and $N_n(1)$ of zeros and ones occur along the sequence x_1^n . It predicts "0" if $N_n(0) > N_n(1)$ and "1" otherwise. The predictability of this scheme is $\pi(x_1^n) = \frac{1}{n} \min\{N_n(0), N_n(1)\}$,

where $N_n(x)$ $x \in \{0,1\}$ is the joint count of ones and zeros occurring along the sequence x_1^n . Comparing the behaviour of prediction to entropy is best demonstrated in the following graph:



H and π drawn as a function of the probability P .

The predictability is related to the error probability in guessing the outcome of a variable, while the compressibility is related to its entropy. It can be further shown that a lower limit to predictability exist in terms of the entropy. For the binary case discussed above, it can be shown that $\rho / 2 \geq \pi \geq h^{-1}(\rho)$, where ρ is the compressibility, π is the predictability and $h(\cdot)$ is the binary entropy function. While the two quantities are not functionally dependent, it is evident that they do coincide on the extreme points.

Predictability and Complexity

We will terminate this long introductory section by a brief discussion of relations between predictability and some other complexity measures. As we stressed in the beginning, one of the great advantages of the universal method is its applicability to arbitrary sequences, including deterministic sequences. The complexity of sequences that are not governed by a probabilistic model (sometimes called "individual" sequences) can be considered in terms of the Solomonoff-Kolmogorov-Chaitin complexity. This measure defines complexity of a sequence as the length of a shortest program for a universal Turing machine that outputs the sequence. In the same spirit we have a complexity definition by Lempel Ziv who considered the shortest code needed to reproduce an individual sequence by an FS encoder. Their well-known Lempel-Ziv algorithm (LZ'78) has been shown to achieve finite-state compressibility for every sequence. The details of the LZ incremental parsing algorithm, that will serve as the basis for our prediction method, will be discussed below. Feder et al. (1992) prove that in a similar manner to the compression property of the incremental parsing method, a predictor which uses the conditional probabilities induced by the LZ scheme attains Markovian predictability and this FS predictability for any individual sequence.

2. Dictionary-based prediction

As we have explained above, we use dictionary based methods for assessing the probability of the next sample given its context. In the following sections we will describe in detail the parsing algorithm and its application to stylistic music generation.

Incremental Parsing

We chose to use an incremental parsing (IP) algorithm suggested by Lempel and Ziv [LZ78]. IP builds a dictionary of distinct motifs by sequentially adding every new phrase that differs by a single next character from the longest match that already exists in the dictionary. For instance, given a text {ababaa...}, IP parses it into {a,b,ab,aa,...} where motifs are separated by commas. The dictionary may be represented as a tree (see last section).

Probability Assignment

Assigning conditional probability $\hat{p}^{LZ}(x_{n+1} | x_1^n)$ of a symbol x_{n+1} given x_1^n as context is done according to the code lengths of the Lempel Ziv compression scheme. Let $c(n)$ be the number of motifs in the parsing of an input n -sequence. Then, $\log(c(n))$ bits are needed to describe each prefix (a motif without its last character), and 1 bit to describe the last character (in case of a binary alphabet). For example, the code for the above sequence is (00,a),(00,b),(01,b),(01,a) where the first entry of each pair gives the index of the prefix and the second

entry gives the next character. Ziv and Lempel have shown that the average code length $c(n)\log(c(n))/n$ converges asymptotically to the entropy of the sequence with increasing n . This proves that the coding is optimal. Since for optimal coding the code length is $1/\text{probability}$, and since all code lengths are equal, we may say that, at least in the long limit, the IP motifs have equal probability.

Thus, taking equal weight for nodes in the tree representation, $\hat{p}^{LZ}(x_{n+1} | x_1^n)$ will be deduced as a ratio between the cardinality of the subtrees (number of subnodes) following the node x_1^n . As the number of subnodes is also the node's share of the probability space (because one codeword is allocated to each node), we see that the amount of code space allocated to a node is proportional to the number of times it occurred.

In our example, the probability on the arc from the root node to {a} is 3/4, root to {b} is 1/4, probability from node {a} to {aa} is 1/2 and from {a} to {ab} is 1/2.

Seen in the bin representation, the probabilities are simply the relative portion of counts of characters $N_c(x)$, $x \in \{a, b\}$ appearing in bin with label c ,

$$\frac{N_c(x)}{N_c(a) + N_c(b)}$$

Sometimes a corrected count is preferred, considering the probability for a next symbol x to

$$\frac{N_c(x) + 1}{N_c(a) + N_c(b) + 2}$$

enter a current bin, giving This is equivalent, in the tree representation, to adding the count of a current node to cardinality of the subtrees in every direction. For large counts, the two probabilities are very close.

Growing the context in IP v.s. Markov models

An interesting relation between Lempel-Ziv and Markov models was discovered by [WIL91] when considering the length of the context used for prediction. In IP every prediction is done in the context of earlier prediction, thus resulting in a "sawtooth" behavior of the context length. For every new phrase the first character has no context, the second has context of length one, and so on. In contrast, the Markov algorithm makes predictions using a totally flat context line determined by the order of the model. Thus, while a Markov algorithm makes all of its prediction based on 3- or 4-character contexts, the IP algorithm will make some of the predictions from lower depth, but very quickly it will exceed the Markov constant depth and use a better context. To compensate for its poor performance in the first characters, IP grows a big tree that has the effect of increasing the average length of the phrase

so that beginnings of the phrase occur less often. As the length of the input increases to infinity, so does the average length, with the startling effect that at infinity it converges to the entropy of the source. In practice though, the average phrase length does not rise fast enough to provide for reliable short-time predictions. On the other hand, it behaves surprisingly well for long sequences. Our experiments show that this IP scheme, along with the appropriate linear representation of music, provides with patterns and inferences that successfully match musical expectation.

Another important feature of the dictionary-based methods is that they are "universal". If the model of the data sequence was known ahead of time, an optimum prediction could be achieved at all times. The difficulty with most real situations is that the probability model for the data is unknown. Therefore one must use a predictor that works well no matter what the data model is. This idea is called "universal prediction" and it is contrasted to Markov predictors that assume a given order of the data model. Universal prediction algorithms make minimal assumptions on the underlying stochastic sources of musical sequences. Thus, they can be used in a great variety of musical and stylistic situations. Our IP based predictor is one such example of universal predictor. This differs also from knowledge-based systems, where specific knowledge about a particular style has to be first understood and implemented [COP96].

3. The Incremental Parsing (IP) algorithm

The *IPMotif* function computes an associative dictionary (the *motif dictionary*) containing motifs discovered over a text.

```

Parameter text, a list of objects
dict = new dictionary
motif = ()
While text is not empty
    motif = motif ! pop (text)
    If motif belongs to dict
        Then value(dict,motif)++
        Else add motif to dict with value
1
    motif = ()
return dict

```

dict is a set of pairs (key, value) where the keys are motifs and values are integer counters. text and motif are ordered lists of untyped objects (we don't restrict to characters). value(dict,motif) retrieves the value associated with motif in dict. W!k notates the list obtained by right-appending object k to list W. Pop(var) returns the leftmost element from the list pointed to by var and advances var by one position to the right.

The text is processed linearly from left to right, object after object, without any backtracking or look-ahead. At any current time, the variable motif contains the current motif W being discovered and the variable text contains the remaining text, beginning just after W. Now a new object k is

popped from the text and appended to the right of `motif`, which value changes to `W!k`. If `W!k` is not already in the dictionary, it is added to it and `motif` is reset to an empty list `()`, thus being prepared to receive the next motif. The LZ78 compression algorithm would, at that time, output a codeword for `W`, depending on `W`'s index in the dictionary, along with the object `k`. Compression would occur because `W`, which must have been previously encountered, is now output as a simple code. But since we are not concerned with compression, we do nothing more. If `W!k` is already in the dictionary, we increment the counter associated with it and iterate. By doing this, we compute for each motif `W!k` the frequency at which object `k` follows motif `W` in the text. It is an IP property that, if motif `W` is in the dictionary, then all its left prefixes are there. So, if for instance motifs `ABC`, `ABCD`, `ABCE`, `ABCDE`, are discovered at different places, the frequency of `C` following `AB` will be equal to 4. Another way to look at it is to consider that, for each motif `W` in the dictionary, for which there exists other motifs `W!ki` in the dictionary, we will easily get the (empirical) conditional probability distribution $P(k_i|W)$ (probability of occurrence of `ki` knowing that `W` has just occurred).

In order to achieve this, we have to transform the motif dictionary into another one, called a *continuation dictionary*, where each key will be a motif `W` from the previous dictionary, and the corresponding value will be a list of couples `(. (k, P(k|W)) .)` for each possible `k` in the object alphabet, representing in effect the empirical distribution of objects following `W`.

The *IPContinuation* function computes a continuation dictionary from a motif dictionary.

```

Parameter dict1, a dictionary
dict2 = new dictionary.
For each pair (W!k, counter) in dict1
  If W belongs to dict2
    Then value(dict2,W) =
      value(dict2,W) !(k counter)
    Else add W to dict2
      with value ( (k counter) )
Normalize (dict2)
Return dict2

```

The function `Normalize` turns the counters in every element of `dict2` into probabilities.

Example

```

Text=(abababcabdabcbabce)
Motif dictionary = { ((a) 6) ((b) 1) ((a b) 5) ((a b c)
3) ((a b d) 1) ((a b c d) 1) ((a b c e) 1) }
Continuation dictionary = { ((a) ((b 1.0))) ((a b) ((c
0.75) (d0.25)) ((a b c) ((d 0.5) (e 0.5)) )

```

As can be seen in the previous example, a single pass IP analysis on a short text is not sufficient to detect a significant amount of motifs. There is no information on continuations for motif `b` or motif `ba`. Due to the asymptotic nature of IP, these motifs will eventually appear when analyzing long texts. Another way to increase redundancy and to detect more motifs is to parse several times the same text using the same

motif dictionary, rotating each time the text to the left by one position.

The *IPGenerate* function generates a new text from a continuation dictionary. Suppose we have already generated a text `(a0 a1 ... an-1)`. There is a parameter `p` which is an upper limit on the size of the past we want to consider in order to choose the next object.

1. Current text is `(a0 a1 ... an-1)`
context = `(an-p ... an-1)`.
2. Check if context is a motif in the continuation dictionary.
3. If found, its associated value gives the probability distribution for the continuation. Make a choice with regard to this distribution and append the chosen object `k` to right of text.
text = text ! k. Iterate in 1.
4. If context is not found in dictionary, shorten it by popping its leftmost object.
context = `(an-p+1 ... an-1)`. If motif becomes `()` generate a *failure* otherwise iterate in 2.
5. Upon failure either stop or append a random object to text, then iterate in 1.

4. Resolving the polyphonic problem

The *IPGenerate* algorithm works on any linear stream of objects. It was successfully tested on linear streams of midi pitches from solo pieces or isolated voices of polyphonic pieces. In order to be able to process polyphony, thus fully capturing rythmical, countrapuntal and harmonic gestures, we had to find a way to linearize multivoice midi data in a way that would musically make sense and take advantage of the IP scheme. The best results were achieved by using a variant of the superposition languages defined by Chemillier & Timis [CHE90].

To understand this, take the 2-voice example shown below.



Only the rhythm is notated. Pitch, as well as other relevant information are coded with letters a through h. If we slice time with respect to the common time unit (the gcd of the durations, i.e. the eighth note) we may code the sequence using 2 parallel words:

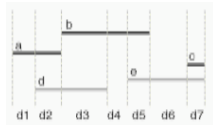
```

aabcdd
effggh

```

where the letter `x` in bold means the continuation of the previous (contiguous) letter `x` (which is either a beginning symbol or itself a continuation). In order to linearize, we go from the normal alphabet, augmented by continuation symbols, $S = \{a, b, c, \dots, \mathbf{a}, \mathbf{b}, \mathbf{c}, \dots\}$ to the cross-alphabet $S \times S$. Now the sequence is: `(a,e) (a, f) (b, f) (c, g) (d, g) (d, h)`.

In order to cope with any arbitrary time structure and to optimize the parsing, we use the following variant.



Time is sliced at each event boundary occurring in any voice. A set of durations $D = \{d_1, \dots, d_7\}$ is thus built. Using the cross alphabet $S \times S \times D$ we build the linear triplet sequence: $(a, -, d_1)$ (a, d, d_2) (b, d, d_3) $(b, -, d_4)$ (b, e, d_5) $(-, e, d_6)$ (c, e, d_7) , where $-$ denotes the empty symbol (musical rest).

These triplets can easily be packed into 3 bytes numbers if we code only the pitches along with the durations. In order to optimize the duration alphabet, we quantize the original durations into a reasonable set of discrete rhythmic values. The idea is then easily generalized to n -voice polyphony.

5. Experiments

Once a multi-voice midi file is transformed into a linear text based on the cross alphabet, it is presented to the IPMotif/IPcontinuation algorithm. The resulting continuation dictionary can then be randomly walked by IPGenerate to build variants of the original music.

The cross-alphabet representation used has proven to fit decisively into the IP framework. In particular, the continuation symbols encode the fact that certain notes, in certain contexts, have a certain probability of being sustained while other notes are playing on other voices. The result is that counterpointal gestures, as well as harmonic patterns, tend to be generated in a realistic way with regard to the original. Another characteristic of IP is that if not only one text but a set of different texts are analyzed using the same motif dictionary, the generation will "interpolate" in a space constituted by this set. This interpolation is not a geometrical one, but rather goes randomly from one model to another when there exists a common pattern of any length and a continuation from the second model is chosen instead of the first one.

IPGenerate has been tested, in normal and interpolation mode, over the set of 2-voices Bach Inventions, normalized for tonality and tempo. While the lack of overall harmonic control does not favor consistent harmonic progression in the resulting simulations, these should be seen as "infinite" streams where very interesting subsequences, show original and convincing counterpoint and harmonic patterns.

On the Bach material, we have established empirically that 0 rotation of the original text would lead to a poor, unusable, continuation dictionary; 3-4 rotations are optimal, in that whole phrases from the original may be generated; more rotations do not improve the generation quality. This is certainly due

to the way phrases are built from combination of small motifs in this style of music.

In the Jazz domain, a new piece by Jean-Remy Guedon, *miniX*, has been created recently at Ircam by the French "Orchestre National de Jazz" with the assistance of Frederic Voisin. In this 20 mn piece, about half of the solo parts were IPGenerated and transcribed on the score.

These experiments were carried-out using OpenMusic, a Lisp-based visual language for music composition [ASS99]. Some results are available at: <http://www.ircam.fr/equipements/repemus>.

References

- [ASS99] Assayag, Agon, Laurson, Rueda. Computer Assisted Composition at Ircam: PatchWork & OpenMusic. Computer Music Journal, to come, 1999.
- [CHEM90] Chemillier, M, Structure et methode algebriques en informatique musicale. Doctorat, LITP 90-4, Paris VI, 1990.
- [COP96] Experiments in Musical intelligence. Madison, WI: A-R Editions, 1996.
- [LZ78] Ziv J, Lempel A, "Compression of individual sequences via variable rate coding", IEEE Trans. Inf. The., 24:5, pp.530-536, 1978.
- [WIL91] Williams, R.N, "Adaptive Data Compression", Kluwer Academic Publishers, Norwell, Massachusetts, 1991.
- D.Blackwell, "Controlled Random Walk", in Proceedings of the 1954 International Congress of mathematics, Vol. III, pp. 336-338, Amsterdam, Holland.
- J.F.Hannan, "Approximation to Bayes Risk in Repeated Plays", in Contributions to the theory of Games, 39, pp: 97-139, Princeton 1957.
- M.Feder, N.Merhav and M.Gutman, "Universal Prediction of individual sequences", IEEE transactions on Information Theory, vol. 38, pp. 1258-1270, 1992.
- Cope, D. (1991). Computers and musical style. Oxford University Press.
- Conklin, D. and Witten, I. (1995). Multiple viewpoint systems for music prediction, Interface, 24, 51- 73.
- Pinkerton, R. (1956). Information theory and melody. Scientific American, (194), 76- 86.
- 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.
- Cope, D. (1987) An expert system from computer-assisted composition, in Computer Music Journal, 11(4): pp. 30-46.
- Ebicoglu, K. (1986), An Expert System for Harmonization of Chorals in the style of J.S.Bach, PhD Dissertation, Department of Computer Science, SUNY at Buffalo.
- Lidov, D. and Gambura, J. (1973), A melody writing algorithm using a formal language model, in Computer Studies in Humanities, 4(3-4), pp. 134-148.