Using Machine-Learning Methods for Musical Style Modeling

Research using statistical and information theoretic tools provides inference and prediction models that, to a certain extent, can generate new musical works imitating the style of the great masters.

Shlomo Dubnov
Ben-Gurion University

Gerard Assayag
Olivier Lartillot
Institut de Récherché et Coordination Acoustique/Musique

Gill Bejerano
Hebrew University

The ability to construct a musical theory from examples presents a great intellectual challenge that, if successfully met, could foster a range of new creative applications. Inspired by this challenge, we sought to apply machine-learning methods to the problem of musical style modeling. Our work so far has produced examples of musical generation and applications to a computer-aided composition system.

Machine learning consists of deriving a mathematical model, such as a set of stochastic rules, from a set of musical examples. The act of musical composition involves a highly structured mental process. Although it is complex and difficult to formalize, it is clearly far from being a random activity.

Our research seeks to capture some of the regularity apparent in the composition process by using statistical and information-theoretic tools to analyze musical pieces. The resulting models can be used for inference and prediction and, to a certain extent, to generate new works that imitate the style of the great masters.

MODELING STYLES

Style modeling implies building a computational representation of the musical surface that captures important stylistic features. The *musical surface* embodies the collection of notes or playing instructions for performing the musical piece. This surface is usually presented to a musician or to musical sequencer software in the form of a written musical score or a musical instruments digital interface (MIDI) file.

The stylistic features that characterize the musical piece are usually hidden from the surface, becoming apparent in the way that higher-level abstractions—such as patterns of rhythm, melody, harmony, and polyphonic relationships—appear in the music, interleaved and recombined in a redundant fashion. Statistical approaches capture some of the statistical redundancies without explicitly modeling the higher-level abstractions. This approach to musical modeling also makes it possible to generate new instances of musical sequences that reflect an explicit musical style.[1]

One of machine learning's main purposes is to create the capability to sensibly generalize. Composers, for example, want to uncover the creative possibilities of certain musical material without necessarily explaining it.

Statistical analysis of a corpus reveals some of the possible recombinations that comply with the constraints or redundancies typically found in a particular style. Interesting applications include style characterization tools for the musicologist,[2] generation of stylistic metadata for intelligent retrieval in musical databases, music generation for Web and game applications, machine improvisation with or without interaction with human performers, and computer-assisted composition.

PREDICTIVE THEORIES IN MUSIC

Building on the work of Leonard B. Meyer,[3] researchers commonly agree that expectations based on recent past context guide musical perception. In music applications, exactly how we make musical

0018-9162/03/$17.00 © 2003 IEEE

Dictionary-based prediction methods sample the data so that a few selected phrases represent most of the information.

predictions—and determine our musical expectations based on it—is unknown. Establishing the right amount of relevant past samples for prediction is difficult because the length of the musical context varies widely, ranging from short figurations to longer motifs.

Taking a large fixed context makes the parameters difficult to estimate, considering that the number of parameters, their associated computational cost, and the data requirements for reliable estimation increase exponentially with the context's size. Coping with this problem requires designing a predictor that can deal with an arbitrary length of observation sequences and be competitive with a rather large class of predictors, such as variable-length Markov predictors.

To design such a predictor, we use two approaches: incremental parsing and prediction suffix trees. IP is based on universal prediction, a method derived from information theory. PST, a learning technique initially developed to statistically model complex sequences, has applications in linguistics and computational biology.

MUSIC GENERATION AND STYLE REPLICATION

A generative theory of music can be constructed by explicitly coding music rules in some logic or formal grammar.[4,5] This approach is sometimes called an expert system or knowledge engineering. Although these methods achieve impressive results, they require extensive exploitation of musical knowledge, often specific to each composer or style. A contrasting approach relies on statistical learning or empirical induction.

Several researchers have used probabilistic methods, notably Markov chains, to model music.[6,7] Markov chains are statistical models of random sequences that assume that the probability for generating the next symbol depends only on a limited past.

First-order Markov chains or models assume dependence on the last symbol only. Higher-order Markov models assume a longer context, so that generating the next symbol depends on several symbols back into the past. Some researchers[6] have shown that at very low orders—such as the second order or so-called bigram—Markov models generate strings that do not recognizably resemble strings in the corpus, while at very high orders, the model simply replicates strings from the corpus.

David Cope's more recent work describes an interesting compromise between these two approaches.[8] Cope uses a grammatical-generation system combined with what he calls *signatures*, melodic micro-gestures typical of individual composers. By identifying and reusing such signatures, Cope reproduced the style of past composers and rendered naturalness to the computer-generated music. Our learning method starts with trying to discover, automatically and in an unsupervised manner, typical phrases or patterns, then it assigns stochastic generation rules for their recombination.

DICTIONARY-BASED PREDICTION

Both the IP and PST methods belong to the general class of dictionary-based prediction methods. These methods operate by parsing an existing musical text into a lexicon of phrases or patterns, called motifs, and provide an inference rule for choosing the next musical object that best follows a current past context.

The parsing scheme must satisfy two conflicting constraints: On the one hand, maximally increasing the dictionary helps to achieve a better prediction, but on the other, enough evidence must be gathered before introducing a new phrase to allow obtaining a reliable estimate of the conditional probability for generation of the next symbol. The "trick" of dictionary-based prediction (and compression) methods is that they cleverly sample the data so that only a few selected phrases reliably represent most of the information.

In contrast to dictionary-based methods, fixed-order Markov models build potentially large probability tables for the appearance of a next symbol at every possible context entry. To avoid this pitfall, more advanced "selective prediction" methods build more complex variable memory Markov models. Although it may seem that the Markov and dictionary-based methods operate in a different manner, they both stem from similar statistical insights.

Generative procedure

We use dictionary-based methods to model the musical (information) source in terms of a lexicon of motifs and their associated prediction probabilities. To generate new instances (messages), these models "stochastically browse" the prediction tree in the following manner:

- Given a current context, check if it appears as a motif in the tree. If found, choose the next symbol according to prediction probabilities.
- If the context is not found, shorten it by removing the oldest (leftmost) symbol and go back to the previous step.

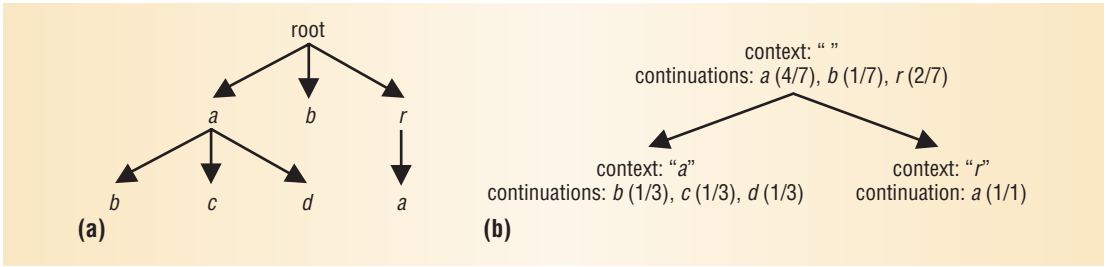

Figure 1. Incremental parsing (IP) example: (a) Analysis tree and (b) its corresponding prediction tree with probability assignments.

These steps iterate indefinitely, producing a sequence of symbols that presumably corresponds to a new message originating from the same source. In some cases, this procedure might fail to find an appropriate continuation and end up with an empty context, or it might tend to repeat the same sequence over and over again in an infinite loop.

Incremental parsing

Jakob Ziv and Abraham Lempel[9] first suggested the IP algorithm in the context of lossless compression. IP builds a dictionary of distinct motifs by making a single left to right traversal of a sequence, sequentially adding to a dictionary every new phrase that differs by a single last character from the longest match that already exists in the dictionary. Using a tree representation, every node is associated with a string, labeled by the characters on the arc that lead from the root to that node.

Each time the parsing algorithm reaches a node, it means the node's string has already occurred from the start of a phrase boundary. When this happens, the system grows a child node that corresponds to the appearance of a new phrase that continues the current string.

Probability assignment

The code lengths of the Ziv and Lempel compression scheme assign conditional probability to a symbol given as context. For the sake of simplicity, let's assume that we compress a binary sequence. Let $c(n)$ be the number of motifs in the parsing of an input n-sequence (the number of nodes in the tree representation). Then, $log(c(n))$ bits are needed to describe each prefix (a motif without its last character), and 1 bit is needed to describe the last character. Since Ziv and Lempel showed that the average code length $c(n)log(c(n))/n$ converges to the entropy of the sequence with increasing n, this provides proof that this coding is asymptotically optimal.

Information theory teaches that in the case of optimal coding, the code length assignment is inversely proportional to the phrase's probability. In the IP case, we use the resulting code lengths, implicitly encoded in the tree, to estimate the probabilities. Since all code lengths in the coding scheme are equal, we consider all IP motifs to have an equal probability. The probability can be deduced now as a ratio between the cardinalities of the different subtrees following the node. As the total number of subnodes is proportional to the number of times the node has occurred, the relative cardinalities of the subtrees from every node correspond to the node's share of the probability space. Thus, for a node labeled by string c, with two possible continuations, the prediction probabilities of the next symbols are taken as the relative portion of counts of number of nodes appearing in all subtrees of the current node c, setting:

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

IP example

The following example demonstrates how we use IP to generate a sequence based on analysis of the simple sequence *abracadabra*. The parsing and generation processes are incremental: The first letter is a continuation of the empty context. Each of the next letters provides a continuation of the longest context that is a suffix of what has been generated already.

In our simple example, if the process first generates an *a* (with probability 4/7), then the next letter is a continuation associated with the context *a* (uniform over *b*, *c*, and *d*), and so on. For example: P(generate "*abrac*") = P(*a*|"")P(*b*|*a*)P(*r*|*ab*)P(*a*|*abr*)P(*c*|*abra*) = 4/7 · 1/3 · 2/7 · 1 · 1/3. Figure 1a shows the analysis tree, and Figure 1b shows the corresponding probability assignments.

SELECTIVE DICTIONARY-BASED PREDICTION

Even though an IP predictor may asymptotically outperform a Markov predictor of a finite order,[10] the music sequences that we model in practice are of a finite length. Moreover, at least in some applications, we do not need to parse the reference set on the fly, as IP does. Using the whole sequence to estimate the statistics may help improve the algorithm's performance for shorter sequences.

Prediction suffix trees

In 1996, Dana Ron and colleagues[11] developed the prediction suffix tree algorithm, named after the data structure used to represent the learned statistical model. A PST represents a dictionary of distinct motifs, much like the one the IP algorithm generates. However, in contrast to the lossless cod-

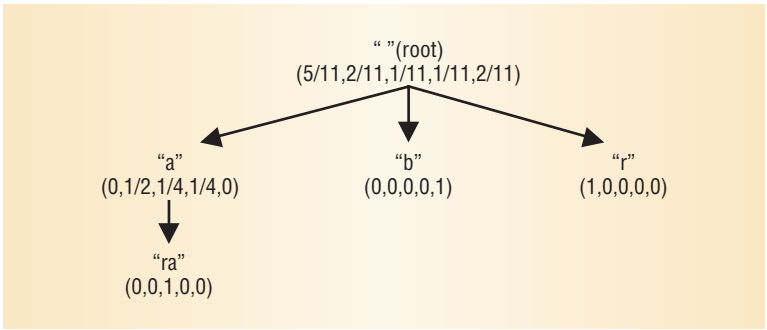

Figure 2. Prediction suffix tree example. The PST analysis assigns probabilities at each node.

ing scheme underlying the IP parsing, the PST algorithm builds a restricted dictionary of only those motifs that both appear a significant number of times throughout the complete source sequence and are meaningful for predicting the immediate future. The framework underlying this approach is efficient lossy compression.

Selective parsing

An empirically motivated variant of the PST algorithm[12] uses a breadth-first search for all motifs that comply simultaneously with three requirements:

- The motif is no longer than some maximal length L.
- Its empirical probability within the given data set exceeds a threshold *Pmin*.
- The conditional probability it induces over the next symbol differs by a multiplicative factor of at least r from that of the shorter contexts it subsumes, for at least one such next symbol.

The search collects only these motifs into a restricted dictionary.

Probability assignment

The empirical probability of a motif is the number of its occurrences in the text divided by the number of times it could have occurred. For example, the empirical probability of the motif "*aa*" within the sequence "*aabaaab*" is 3/6 = 0.5. A conditional probability to observe a symbol after a given context is the number of times this symbol comes immediately after that context, divided by the total occurrences of the context in the text. Thus, the conditional probability of seeing "*b*" after "*aa*" in the sequence "*aabaaab*" is 2/3. Since the conditional probability of seeing "*b*" after "*a*" is 2/5, the multiplicative difference between the conditional probability to observe "*b*" after "*aa*" and that after "*a*" is (2/3)/(2/5) = 5/3.

The PST formulation incorporates the counts in a slightly different manner than in IP. Using the same notation as in the previous example, the probability for a next symbol extension x from a node labeled by string c is given by $\hat{P}_c(x) = (1 - |\Sigma|g)P_c(x) + g$, where $|\Sigma|$ is the size of the alphabet and g is the smoothing factor, $0 < g < 1 / |\Sigma|$. Fractional probabilities are collected from all possible next-symbol extensions in proportion to their actual counts, then they are redistributed equally between all counts to achieve a minimal conditional probability for any continuation $\hat{P}_c(x) \geq g > 0$.

PST example

The PST analysis example using *abracadabra* has the following parameters: $L = 2$, *Pmin* = 0.15, $r = 2$, and $g = 0$. For each node, the analysis associates the list of probabilities that the continuation may be, respectively, a, b, c, d, or r.

Figure 2 shows the results of this analysis. Potential node *ba* failed the Pmin criterion, while potential node *bra* passed that criterion but failed both the L and r criteria. Generation proceeds much like in the IP resulting tree. Now, however,

P(generate "*abrac*") = P(*a*|"")P(*b*|*a*)P(*r*|*ab*) P(*a*|*abr*)P(*c*|*abra*) = 5/11 · 1/2 · 1· 1 · 1.

MUSIC REPRESENTATION

So far we have loosely used the term *sequence* to denote an ordered list of objects or symbols. To capture a significant amount of musical substance, a preanalytic phase cuts the musical data—generally in MIDI format—into slices, with the appearance of new events and the termination of past ones determining the beginnings and endings. Every slice has duration information and contains a series of channels, each of which contains pitch and velocity information and possibly other musical parameters. These slices are objects or symbols, and the set of possible symbols is the alphabet.

Figure 3a, for example, shows the piano roll representation for the beginning of Bach's *Prelude in C*, in which lowercase letters represent the third octave and uppercase letters the fourth octave. Figure 3b shows how the analysis slices this sequence into symbols, a process sometimes called *quantization*. The bold letters represent the beginnings of notes.

ENHANCEMENT FILTERS

In light of some theoretical insights and preliminary tests, we made several improvements to our system's representation, including preanalytic simplification, generative constraints, loop escape, and analysis-synthesis parameter reconstruction.

Preanalytic simplification

Real musical sequences—for example, MIDI files of a piece interpreted by a musician—feature fluctuations of note onsets, durations, and velocities,

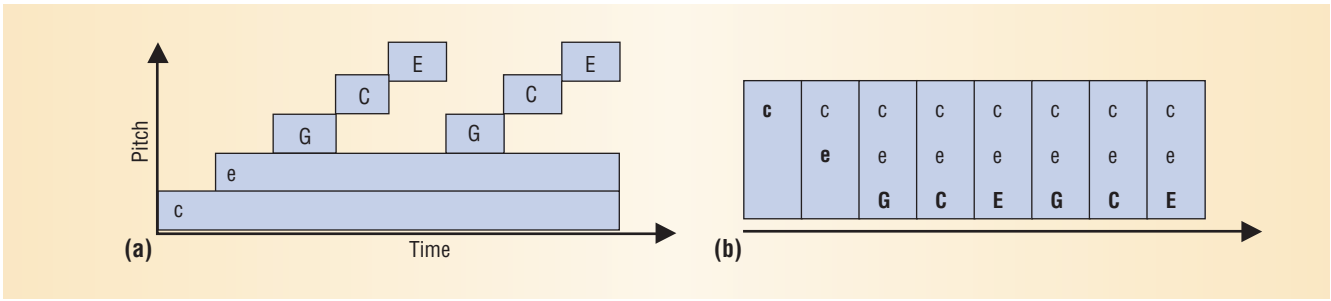

Figure 3. Quantization example. (a) Piano roll notation for the beginning of Bach's* Prelude in C*, and (b) the same sequence sliced into symbols.

inducing a complexity that fools the analysis. The alphabet size tends to increase in an intractable way, leading to unexpected failures and poor generalization power.

We have thus developed a toolkit containing five simplification filters:

- the *arpeggio filter* vertically aligns notes that are attacked nearly at the same time;
- the *legato filter* removes overlap between successive notes;
- the *staccato filter* ignores silence between successive notes;
- the *release filter* vertically aligns note releases; and
- the *duration filter* statistically quantizes the durations to reduce the duration alphabet.

Users can apply thresholds to change these features. For example, a threshold of 50 ms separates a struck chord on the piano from an intentionally arpeggiated chord. Using the simplification toolkit on MIDI files containing live performances provides musical data that is full of idiosyncrasies and is thus of great value as a model for synthetic improvisation.

Generative constraints

It is possible to specify constraints during the generation phase. At each cycle, if the new symbol does not respect the constraint, the generation is canceled and a new symbol is considered. If no symbol is found to be satisfactory, the algorithm backtracks to the previous cycle, or more, if necessary. For example, it is possible to specify a continuity constraint that imposes a minimal context size anytime during the generation phase to avoid musical discontinuity in the case of an empty context.

Loop escape

If no prediction smoothing is applied, the generation phase can easily enter into an infinite loop. A context-generated subtree consists of the exhaustive set of all the possible contexts that can be met after the present one. Practically, we need to know only the size of this subtree, which is computed once before the generation phase. This computation consists of marking, directly on the original tree, all the possible future contexts and a count of these marks.

When the size of the context-generated subtree is below a user-specified threshold N, an N-order-loop is detected. The loop phenomenon occurs principally because the generation phase searches for the unique maximal context and proposes few alternative continuation symbols. In the case of a loop, we loosen this constraint and examine not only the maximal context but also shorter ones, which escape the loop in most cases.

Analysis-synthesis parameter reconstruction

In our formulation, a symbol is a Cartesian product of several musical parameters. To increase abstraction power in the analysis, we allow the system to discard some parameters—for example, the velocity. The retained parameters are called *analysis information*. However, in the generation phase, the system cannot retrieve the discarded information. For example, if the system discards note durations and dynamics, the result is isorhythmic and dynamically flat musical sequences that sound like the work of a musical box. Our solution is to store the excluded information, such as note dynamics that are important for expressive performance, in the model. This synthetic information can be reconstructed during the generation phase.

This solution has significant advantages. First, the generated music regains much of the diversity, spirit, and human appeal of the original MIDI recording. Moreover, since it is possible to restrict the analytic information and therefore find more redundancy in the original sequence, the generation phase is less constrained. At each generation cycle, every context features many more continuations than is possible without restriction.

An OpenMusic example

These musical machine-learning methods were realized in software that was implemented as a library in OpenMusic,[13] an open source, Lisp-based visual programming environment developed at Ircam (www.ircam.fr/OpenMusic) for music composition and analysis.

Figure 4 shows the learning and generating stages in an OpenMusic-commented visual program. Input parameters in the boxes let the user choose between the IP and PST models and tune its parameters: loop escape, analysis-synthesis, and so on. The boxes on the right side use the raw MIDI-like

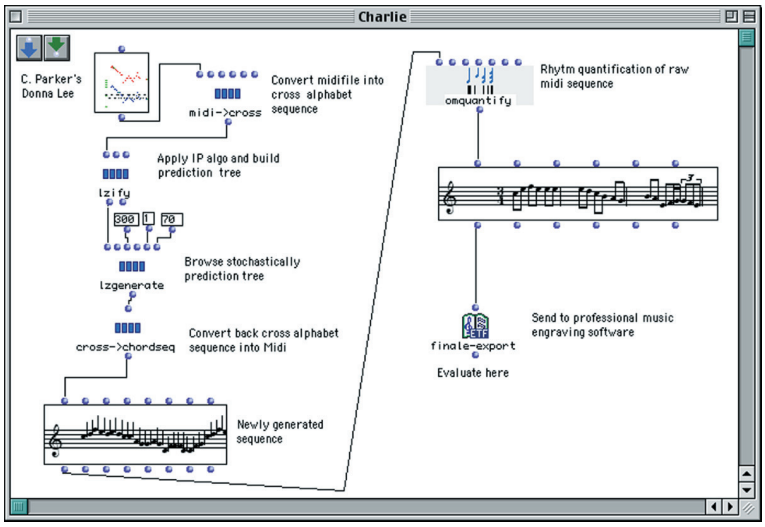

Figure 4. OpenMusic programming environment. The user chooses the IP or PST model, then tunes it using parameters such as loop escape and analysis-synthesis. The boxes on the right side use the raw output to compute a metric-rhythmic segmentation that is compatible with common music notation.

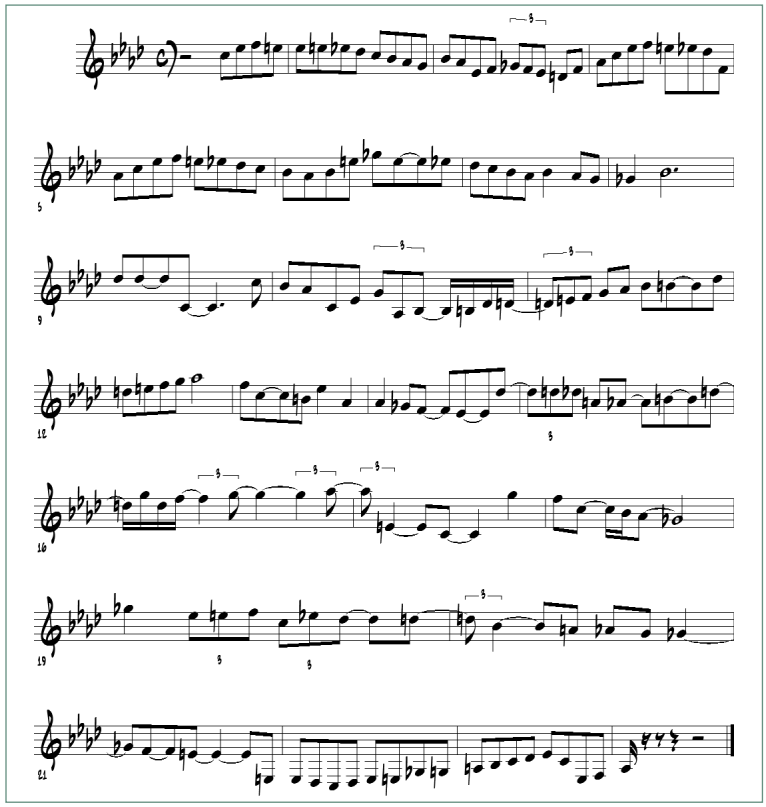

Figure 5. Learned MIDI file. This possible machine improvisation on Charlie Parker's famous standard,* Donna Lee*, was created using an IP model.

output to compute a metric-rhythmic segmentation that is compatible with common music notation. The program sends the result to professional music notation software.

The learned MIDI file in Figure 5 is an interpretation of Charlie Parker's famous *Donna Lee* in standard music notation. An IP model provided this machine improvisation, in which only the first voice of the MIDI file, the saxophone, has been learned. The IP improvisation starts on a phrase close to the original theme and seems to turn around a repetitive pattern of C, E♭, F, E, E♭, D♭, C, B♭, and A♭, with different ornamentations, then escapes—thanks to the loop escape mechanism—toward a typical Bop style improvisation. The MIDI rendering can be heard at www.ircam.fr/equipes/repmus/MachineImpro.

MUSICAL EXPERIMENTS

We have carried out extensive musical experiments to test our system and compare the two generative algorithms. First, we gathered MIDI files from several sources, including polyphonic instrumental and piano music, with styles ranging from early renaissance and baroque music to hard-bop jazz. In cases in which the MIDI file was derived from a live recording, the combination of the simplification toolbox and the learning-generation gives excellent results. This simplification is also important in cases in which the overall complexity leads to a huge alphabet.

Style interpolation

When the system analyzes the set of different musical pieces simultaneously into a dictionary of common motifs, the generation can *interpolate* over the space that the separate models span. When the analysis finds a common pattern of any length between the separate dictionaries during training, the generation phase can create smooth transitions from one style to another by choosing at random a continuation path that belongs to the other model. Testing this idea over a set of two-voice Bach compositions reveals interesting subsequences that show original and convincing counterpoint and harmonic patterns, consistent with Bach's style.

IP versus PST

Table 1 provides a comparison of IP and PST properties. Although they are statistically related, at times, these two methods produce significantly different musical results.

Generally, in our experience, IP tends to capture a piece's musically meaningful motifs. It has a tendency to copy complete blocks from the original music, "mimicking" it with unexpected juxtapositions, and at times ridiculing the concepts of rhythm, meter, and form.

PST, on the other hand, shows more inventiveness by introducing transitions in the middle of a bar or a phrase, thus creating not just juxtapositions but more interesting transitions. The tradeoff

Table 1. Comparison of IP and PST properties.

Incremental parsing	Prediction suffix trees
Guiding principle: Lossless coding	Lossy compression
Everything parsed: Completeness assures inclusion of all observed rare transitions	Selective, partial parsing rules out rare events and events that do not improve prediction
Online estimation through instantaneous coding	Batch analysis through file compression
Poor conditional empirical probability estimation for short inputs given IP parsing's sensitivity to even a single symbol change	Robust conditional empirical probability estimation for significant events

is that PST might cause false notes to appear more often.

Available results

We used OpenMusic programs to produce a variety of musical examples, including piano improvisations in the style of Chick Corea, Chopin's etudes, polyphonic Bach counterpoint, and modern jazz improvisations derived from a training set fed by several performers asked to play for the learning system (www.ircam.fr/equipes/repmus/MachineImpro).

With the assistance of Frederic Voisin, Ircam and the Cite de la Musique Jazz Festival have used this system as a tool for computer-assisted composition, creating two original pieces by Jean-Remy Guedon that were performed in a concert by the French Orchestre National de Jazz. Several solo parts in these pieces were IP-generated, transcribed to a score, and performed during the concert on saxophone and electric guitar.

Interactive and real-time issues

Because the IP algorithm is incremental in nature, it is suitable for real-time interactive performance. We have developed a prototype[1] that browses a prediction tree with added constraints derived from a performer who plays one voice of the polyphony and generates the other parts, which may be contrapuntal, in real time, consistent with the learned style. This differs dramatically from usual accompaniment systems, which generate chords or arpeggios based on simple diatonic triad rules. Other researchers have built Markov-related real-time interactive systems.[14,15]

We are currently working on a more general, OpenMusic-based real-time performance system in which the machine can interact with one or several performers, catching the style and responding on the fly. In addition to the learning and generation process, users will be able to insert compositional transformations expressed in the powerful OpenMusic language between the learning, generation, and rendering steps.

To better understand and control the system's behavior, we are collaborating with Emmanuel Bigand, a psychoacoustician from Université de Dijon, to measure to what extent and for how long our generation system can "fool" both musically educated and uneducated listeners. In a blind-test setup, participants are asked to distinguish between excerpts from original performances and from generated ones, using different parameter configurations. ■

Acknowledgment

Gill Bejerano is supported by a grant from Israel's Ministry of Science.

References

1. G. Assayag, S. Dubnov, and O. Delerue, "Guessing the Composer's Mind: Applying Universal Prediction to Musical Style," *Proc. Int'l Computer Music Conf.*, Int'l Computer Music Assoc., 1999, pp. 496-499.
2. S. Dubnov, G. Assayag, and R. El-Yaniv, "Universal Classification Applied to Musical Sequences," *Proc. Int'l Computer Music Conf.*, Int'l Computer Music Assoc., 1998, pp. 332-340.
3. L. Meyer, *Emotion and Meaning in Music*, University of Chicago Press, 1961.
4. K. Ebcioglu, *An Expert System for Harmonization of Chorals in the Style of J.S. Bach*, doctoral dissertation, Dept. Computer Science, SUNY at Buffalo, 1986.
5. D. Lidov and J. Gambura, "A Melody Writing Algorithm Using a Formal Language Model," *Computer Studies in Humanities*, 1973, pp. 134-148.
6. F. Brooks Jr. et al., "An Experiment in Musical Composition," *Machine Models of Music*, S. Schwanauer and D. Levitt, eds., MIT Press, 1993, pp. 23-40.
7. D. Conklin and I. Witten, "Multiple Viewpoint Systems for Music Prediction," *Interface*, vol. 24, 1995, pp. 51-73.
8. D. Cope, *Computers and Musical Style*, Oxford University Press, 1991.
9. J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable Rate Coding," *IEEE Trans. Information Theory*, vol. 24, no. 5, 1978, pp. 530-536.
10. M. Feder, N. Merhav, and M. Gutman, "Universal Prediction of Individual Sequences," *IEEE Trans. Information Theory*, vol. 38, 1992, pp. 1258-1270.

11. D. Ron, Y. Singer, and N. Tishby, "The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length," *Machine Learning*, vol. 25, 1996, pp. 117-149.
12. G. Bejerano and G. Yona, "Variations on Probabilistic Suffix Trees: Statistical Modeling and Prediction of Protein Families," *Bioinformatics*, vol. 17, 2001, pp. 23-43.
13. G. Assayag et al., "Computer Assisted Composition at Ircam: PatchWork & OpenMusic," *The Computer Music J.*, vol. 23, no. 3, 1999, pp. 59-72.
14. D. Zicarelli, "M and Jam Factory," *The Computer Music J.*, vol. 11, no. 4, 1987, pp. 13-29.
15. F. Pachet, "Interacting with a Musical Learning System: The Continuator," *Proc. ICMAI 2002*, Springer-Verlag, 2002, pp. 119-132.

Shlomo Dubnov *is a lecturer in the Department of Communication Systems Engineering, Ben-Gurion University, Israel. His research interests include multimedia communication, computer music, and signal processing. He received a PhD in computer science and musicology from the Hebrew University, Jerusalem. Contact him at dubnov@bgumail.bgu.ac.il.*

Gerard Assayag *is head of the research group on music representations at Ircam, the Institut de Récherché et Coordination Acoustique/Musique, Paris. His research interests include programming languages, computer music, and music modeling. He received a Diplôme d'Etudes Approfondies in computer science from Université Paris VI. Contact him at assayag@ircam.fr.*

Olivier Lartillot *is a PhD student at Ircam. His research interests include computer music and cognitive sciences. He received a Diplôme d'Etudes Approfondies in an interdisciplinary program (acoustics, signal processing, and computer music) from Université Paris VI. Contact him at lartillo@ircam.fr.*

Gill Bejerano *is a postdoctoral researcher at the Center for Biomolecular Science and Engineering, University of California, Santa Cruz. His research interests include computational molecular biology, machine learning, and information theory. He is a PhD candidate in computer science at the Hebrew University, Jerusalem. Contact him at jill@soe.ucsc.edu.*