Musical Metacreation: Papers from the 2013 AIIDE Workshop (WS-13-22)

Feature Selection and Composition Using PyOracle

Greg Surges, Shlomo Dubnov
{gsurges, sdubnov}@ucsd.edu
Music Department
University of California, San Diego
La Jolla, CA

Abstract

A system is described which uses the Audio Oracle algorithm for music analysis and machine improvisation. Some improvements on previous Factor Oracle-based systems are presented, including automatic model calibration based on measures from Music Information Dynamics, facilities for compositional structuring and automation, and an audio-based query mode which uses the input signal to influence the output of the generative system.

Introduction

PyOracle is a new machine improvisation and analysis system in the family of software built around the Factor Oracle and Audio Oracle algorithms (Assayag et al. 2006; François, Chew, and Thurmond 2007). PyOracle is the first software to use Audio Oracle (AO), a graph structure built using features derived from input audio, in the context of machine improvisation. AO is useful for both analysis and generative purposes, and will be described below.

The PyOracle project can be divided into two parts: PyOracle Analyzer and PyOracle Improviser. PyOracle Analyzer is a Python library for AO analysis of music, which also has functions related to Music Information Dynamics. While PyOracle Analyzer contains some generative music functions, it is primarily intended for offline work. PyOracle Improviser embeds PyOracle Analyzer into the Max/MSP programming environment, enabling real-time analysis of input and generation of new musical content based on that input. PyOracle Improviser is conceived of as a software improvising partner which learns elements of a performer's style and generates a real-time audio accompaniment. In this context, the AO algorithm is used to determine the repetition structure of an improvisation in an online fashion. This repetition structure is then used to recombine the original musical material into new, yet related material.

The PyOracle system makes use of ideas from Music Information Dynamics, in order to determine the best model of an arbitrary input signal. A measure called Music Information Rate (IR) is used to measure the amounts of complexity and repetition in the signal over time, and can be used to find the ideal AO model. IR can also be used to determine the most relevant or informative audio feature at a given time.

PyOracle Improviser provides some unique features for enabling composition and structured improvisations. Constraints, probabilities, and other parameters can be modified in real-time or according to a predefined script. We will discuss the use of PyOracle Improviser in a compositional context, and suggest some directions for further work in this direction.

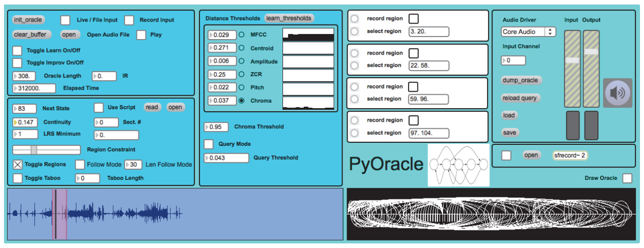
init_oracle
Live / File Input
Record Input
clear_buffer
open
Open Audio File
Play
Toggle Learn On/Off
Toggle Improv On/Off
308. Oracle Length
0. IR
312000. Elapsed Time
83 Next State
Use Script
read
open
0.147 Continuity
0 Sect. #
1 LRS Minimum
0.
Region Constraint
☒ Toggle Regions
Follow Mode
30 Len Follow Mode
Toggle Taboo
0 Taboo Length
Distance Thresholds
learn_thresholds
0.029 MFCC
0.271 Centroid
0.006 Amplitude
0.25 ZCR
0.022 Pitch
0.037 ● Chroma
0.95 Chroma Threshold
Query Mode
0.043 Query Threshold
record region
select region 3. 20.
record region
select region 22. 58.
record region
select region 59. 96.
record region
select region 97. 104.
PyOracle
Audio Driver
Core Audio
Input Channel
0
Input
Output
dump_oracle
reload query
load
save
open
sfrecord~ 2
Draw Oracle

Figure 1: The PyOracle Improviser interface.

Machine Improvisation

Machine improvisation is a field of research wherein computer programs are designed to function as improvisational soloists or partners. In a situation where the software functions as part of a duo with a human performer, the software receives musical input from the performer in the form of MIDI or audio, and responds appropriately according to some stylistic model or other algorithm. An important early work in this style is George Lewis's *Voyager* (1988), described as a "virtual improvising orchestra." The *Voyager* software receives MIDI input derived from an instrumental audio signal, analyzes it, and produces output with one of a set of many timbres (Lewis 2000). *Voyager* is capable of both producing variations on input material and generating completely new material.

The Continuator project uses variable-length Markov chains to generate new continuations from a MIDI input stream (Pachet 2003). The MIDI stream is parsed and a tree structure is constructed. As futher MIDI input arrives, the tree is traversed to find continuations of the input. If no continuation is found, the next event is randomly chosen. The use of a weighted fitness function enables control over the "sensitivity" of the machine improvisation to the current input musical context. The authors also present some interesting ideas for structured improvisations using the Continuator system.

In (Gifford and Brown 2011), a system called the Jambot is introduced. The Jambot attempts to combine "imitative" and "intelligent" behaviors, using a confidence measure to trigger switching between the two behaviors. The authors demonstrate this confidence-based switching method with the example of a beat-tracking behavior. If the system has a high confidence in its beat-tracking ability, given the current musical context, it will function more "intelligently" - producing more novel material. On the other hand, if the system is unconfident in its current model of the beat, it will switch to an "imitative" behavior until it regains confidence.

Many previous machine improvisation systems use symbolic music representations, such as MIDI, which can have some advantages, but can also fail to capture significant musical information, such as timbre. Additionally, by their very nature, these symbolic representations imply quantization of musical features. For many features, such as those related to the harmonic structure of a sound, it is often unclear how best to quantize the features - a "one size fits all" approach will often fail to accurately represent many sounds. The Audio Oracle algorithm, described in the following section, attempts to address these problems.

Audio Oracle

Audio Oracle (AO) is an audio analysis method and a representation of musical structure, and has applications to machine improvisation. AO extends previous methods which used Lempel-Ziv and Probabilistic Suffix Trees, described in (Dubnov, Assayag, and Cont 2007). The AO algorithm is based on a string matching algorithm called Factor Oracle (FO), but extends it to audio signals (Allauzen, Crochemore, and Raffinot 1999). AO accepts an audio signal stream as input, transforms it into a time-indexed sequence of feature vectors (calculated from a moving window on the time-domain signal), and submits these vectors to pattern analysis that attempts to detect repeating sub-sequences or factors in the audio stream. Mathematically speaking, the AO generalizes the FO to partial or imperfect matching by operating over metric spaces instead of a symbolic domain. In other words, AO does not require the same kind of quantization of input that FO does. One of the main challenges in producing an AO analysis is determining the level of similarity needed for detecting approximate repetition. This can be done using information theoretic considerations about the structure of the resulting AO. The "listening mechanism" of the AO tunes itself to the differences in the acoustic signal so as to produce a optimal representation that is the most informative in terms of its prediction properties. This "tuning" process will be described in greater detail below. Like the FO-based method, AO generates an automaton containing pointers to different locations in the audio data that satisfy certain similarity criteria, as found by the algorithm. The resulting automaton is passed later to an oracle compression module that is used for estimation of Information Dynamics, as will be described in the following section.

An example Audio Oracle segment is shown in Figure 2. Similar to the Factor Oracle algorithm, forward transitions (upper arcs) correspond to states that can produce *similar patterns* with alternative continuations by continuing forward, and suffix links (lower arcs) correspond to states that share the *largest similar sub-clip* in their past when going backward.

During construction, the oracle is updated in an incremental manner, allowing for real-time construction. The algorithm iteratively traverses the previously learned oracle structure, jumping along suffix links, in order to add new links. For a more detailed treatment of the AO construction algorithm, see (Dubnov, Assayag, and Cont 2007).

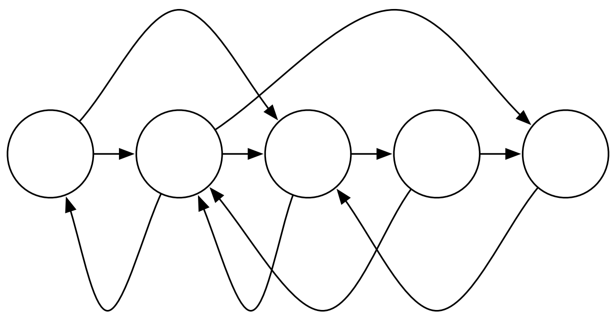

Figure 2: An example of an Audio Oracle structure.

Computational Aesthetic Theories

One of the attractive properties of PyOracle Analyzer is that it performs analysis of music in terms of signal complexity and repetition structure (related to familiarity) during the learning stage undertaken in preparation for improvisation. The link between statistical properties of the signal and human perception, initially established in (Dubnov, McAdams, and Reynolds 2006), motivates our current approach to using AO for composition, as explained below.

Recent research has emphasized the relationship between familiarity and complexity in aesthetic appreciation. Much of this work is influenced by Birkhoff's notion of *aesthetic measure* (Rigau, Feixas, and Sbert 2008). This measure, defined by Birkhoff as the ratio between order and complexity, has been elaborated by researchers using information theoretic measures such as entropy to quantify the complexity of a stimuli. In (Rigau, Feixas, and Sbert 2008), Rigau et al. reformalized the aesthetic measure using images. The complexity of an image was defined according to the palette of colors used, while the familiarity measure corresponded to the difference between the maximum size of the image and the output of a compression algorithm run on the image. In other words, the compression algorithm detected redundancies or repetitions in the image, and exploited them in order to reduce the size of the compressed image. The difference between the maximum size and the compressed size is equivalent to the amount of the image which can be explained as a repetition of something which occurs elsewhere.

In (Silvia 2005), the authors describe a series of experiments investigating the effects of novelty and coping-potential on viewer interest in art. The studies found that viewers with a (self-described) high ability to understand complex art spent more time viewing complex art, compared to viewers with a low ability. These results suggest that there is an optimal balance between the amounts of novelty and repetition in a work of art. If coping-potential - the ability to "explain" a work - increases with repetition, and novelty is indicated by the appearance of unique elements or events, we can relate this model of aesthetic appreciation to the model built by the AO algorithm.

Potter, Wiggins, and Pearce (Potter et al. 2007) present an approach toward modeling listener expectation in minimalist music. The authors use a symbolic representation of the musical surface, and two learning models. The first model, based on short-term memory, learns from the piece being analyzed. The second, a long-term memory model, is trained on a large corpus of melodic material. The two models are then combined into a single statistical model of the musical surface. The "information content" and entropy of the model are then measured, where the information content is related to surprise or unexpectedness and the entropy is related to the strength of the listener's expectations.

Music Information Dynamics and Information Rate

Music Information Dynamics is a field of study that considers the evolution in information contents of music (Abdallah and Plumbley 2009; Potter et al. 2007). Music Information Dynamics is often assumed to be related to structures captured by cognitive processes such as the forming, validation, and violation of musical expectations (Meyer 1956). In particular, a measure called Information Rate (IR) has been studied in relation to human judgements of emotional force and familiarity (Dubnov, McAdams, and Reynolds 2006).

Information Rate (IR) measures the reduction in uncertainty about a signal when past information is taken into account. It is formally defined as mutual information between the past $x_{past} = \{x_1, x_2, ..., x_{n-1}\}$ and the present x_n of a signal x

$$IR(x_{past}, x_n) = H(x_n) - H(x_n|x_{past}) \quad (1)$$

with $H(x) = -\Sigma P(x) log_2 P(x)$ and
$H(x|y) = -\Sigma P(x, y) log_2 P(x|y)$ being the Shannon entropy and conditional entropy respectively, of the variable x which is distributed according to probability $P(x)$. In (Dubnov, Assayag, and Cont 2011), an alternative formulation of IR using AO was developed as shown in (2).

$$IR_{AO}(x_{past}, x_n) = C(x_n) - C(x_n|x_{past}) \quad (2)$$

where $C(\cdot)$ is the coding length obtained by a compression algorithm, and is measured in terms of the number of bits required to represent each element x_n. It was shown how Audio Oracle structure can be used to estimate the IR of an audio recording (Dubnov, Assayag, and Cont 2011). For more details on the compression method itself the reader is referred to (Lefebvre and Lecroq 2001).

Recalling the AO algorithm presented above, we can use the notion of balancing complexity and familiarity to solve the problem of how to determine the best model of the signal. The ideal model should represent the complexity of the signal, while also capturing repetitions which occur. If we consider IR to be related to the amount of information shared between the past and the present, i.e. how much of the present is a repetition of something which has come before, we can relate IR to the balance between complexity and familiarity. We can also draw a connection between IR and surprise. A low IR value corresponds to a high amount of novelty, and therefore surprise, while a high IR value corresponds to a repetition of something previously heard. As found in the previous studies on computational aesthetics, the most pleasing aesthetic experiences rely on a balance between order and complexity. This has a practical application in determining the best distance threshold to use during AO construction.

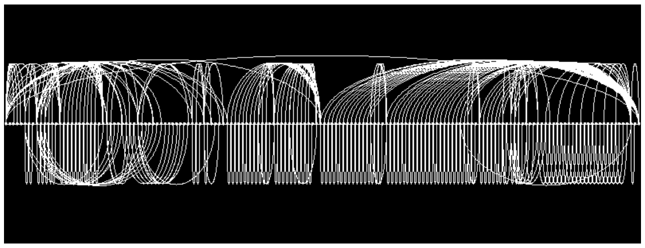

Figure 3: The ideal AO model. Note the balance between occurences of transitions (upper arcs) and suffixes (low arcs).

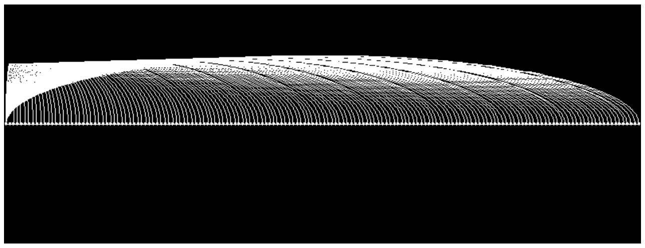

Figure 4: An AO with the distance threshold set too low. Each state is considered unique.

Tuning the AO Algorithm

For a more intuitive understanding of IR and how it relates to the AO structure, consider Figures 3 - 5. These figures demonstrate the difference between an oracle constructed using a well-chosen distance threshold and one constructed where the threshold is too low or too high. The distance threshold is used during the construction of the AO, to detect similar states. If the distance between two states is found to be lower than the threshold, the states are determined to be similar. The work analyzed here is the complete fourth

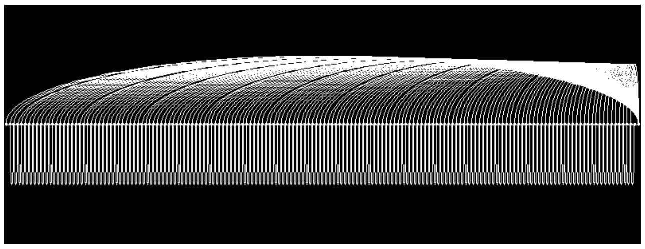

Figure 5: An AO with the distance threshold set too high. All states are considered repetitions.

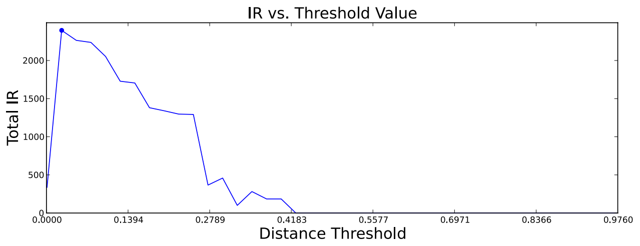

Figure 7: Total IR vs. Distance Threshold for Shakuhachi recording.

movement of Prokofiev's *Visions Fugitives*, a short composition for solo piano. The first oracle, shown in Figure 3, is considered the optimal oracle. We assume that a good AO analysis will capture as much as possible of the mutual information between the past and present of the signal, and therefore we favor analyses with high total IR. Through an iterative process, we compare the total IR of each of a range of possible distance thresholds, and select the particular threshold which maximizes total IR. Figure 6 shows the total IR as a function of the distance threshold. The peak of this function yields the optimal AO, though it is often worth studying oracles formed using secondary peaks as well. In the case of the Prokofiev work, the optimal distance threshold was found to be 0.21, which yielded a total IR of 1592.7. Figure 4 shows an oracle where the distance threshold was too low. In this case, no similarity between any frames was found, and all frames were considered to be new. Each frame has only a link from state 0, indicating that this is its first appearance in the sequence. The algorithm has determined the input to be a "random" sequence, where all frames are unrelated and no repetitions or patterns occur. Conversely, Figure 5 shows an oracle where the distance threshold was set too high, lumping dissimilar frames together, and producing an oracle where all frames were considered to be repetitions of the first. Each frame has a suffix link to the previous frame, indicating this repetition. Note that in these diagrams, suffix links to the 0th frame, present when a new frame is detected, are omitted for visual clarity.

To see how this threshold selection process "tunes" the oracle to its input, consider the difference between Figures 6 and 7. As described above, Figure 6 shows the total IR as a function of the distance threshold for the Prokofiev piano work. Figure 7 shows a similar plot, but for a recording of a solo shakuhachi. The underlying feature for both plots was Mel-Frequency Cepstral Coeffecients (MFCCs). The optimal threshold for the Prokofiev was 0.21, while that of the shakuhachi performance was 0.02. This corresponds to the oracle adjusting to the more subtle timbral variation in the shakuhachi recording. A lower distance threshold was required to detect meaningful distinctions between frames with smaller variations.

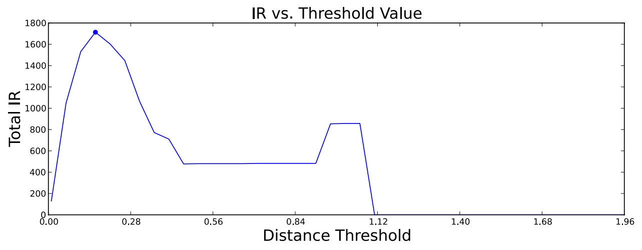

Figure 6: Total IR as a function of Distance Threshold for Prokofiev piano work.

Music Generation with PyOracle Improviser

Audio Oracle can be used in a generative context, to produce real-time variations on the learned musical structure. PyOracle Improviser aims to facilitate experimentation and composition with this algorithm. In order to understand how one might interact with PyOracle Improviser, it is helpful to consider the basic parameters of the system. A pointer is used to navigate the oracle structure, navigating linearly forward with probability p and jumping forward or backward along a suffix link with probability $1 - p$. This parameter is referred to as the *continuity* control. As the pointer moves, audio corresponding to the current oracle state is played. Each new frame is cross-faded with the previous, in order to avoid discontinuities in the output signal. Forward linear movement corresponds to playback of the original audio, while jumps forward or backward produce new variations on the original material. Since suffix links connect states which share similar context, the transitions between these jump points will be musically smooth. The length of this shared context, the *longest repeated suffix* (*LRS*), can also be used to constrain navigation. Setting a minimum *LRS* allows jumps to occur only when a certain length of context is shared between states. This parameter allows the user to control the smoothness of the jumps, with shorter contexts producing less smooth jumps. If a sufficiently long shared context is not found, the system defaults to a linear continuation to the next frame. The oracle navigation can also be limited to only certain areas of the data structure, in effect restricting the musical generation to specific materials. This is accomplished through specifying a *region* of contiguous states. Transitions or jumps which would cause the navigation to fall outside this region are ignored. Finally, a *taboo list* can be used. This functions as a circular buffer of flexible length, into which the index of each played frame is placed. During navigation, if the taboo list is active, jumps or transitions which would cause playback of a frame already present in the taboo list are ignored. This feature helps to eliminate excessive repetition and loop-like behavior during oracle navigation.

PyOracle Improviser uses the IR tuning process described in the previous section to independently determine the best distance threshold for each of 6 signal features. PyOracle Improviser currently uses MFCCs, spectral centroid, RMS amplitude, zero-crossing rate, pitch, and chroma estimates. These features were chosen because they relate to important musical parameters. MFCCs and the spectral centroid provide information about musical timbre and brightness, RMS amplitude corresponds to signal energy and musical dynamics, zero-crossing rate is related to both the fundamental frequency and the noisiness of the signal, and pitch and chroma estimates relate to melodic or harmonic materials used. Oracles built on different features will often have different structures, and so it is important to determine each threshold independently. The training process is simple: the human improviser plays a short (approximately 30 seconds to 1 minute) improvisation, aiming to create an excerpted version of their full improvisation. During this short improvisation, feature vectors are stored. After the improvisation, oracles are built on each feature, iterating over a range of possible distance thresholds. Finally, as described above, the threshold which maximizes the total IR of the sequence for each feature is retained, and used to build the oracle for that feature during the full improvisation.

Composition and Structured Improvisation using PyOracle Improviser

PyOracle Improviser also provides several new features to facilitate composing and designing structured improvisations using AO. The primary means for creating a reproduceable structure is through a scripting mechanism. Any of the parameters described in the previous section - such as *continuity*, *LRS*, and *regions* - can be scripted using a simple timeline-based system. Sections based on different musical materials are easily defined using *regions*, and areas of greater or lesser musical “fragmentation” can be defined using the *continuity* parameter.

Several different modes of interaction are implemented in PyOracle Improviser. In the **standard** mode, oracle navigation and interaction are defined according to the above parameters, and modified according to a script or in real-time by a human machine operator. The **query** mode is a novel introduction, and deserves special mention here. A similar idea has been explored in the SoMax system, developed by Bonasse-Gahot, in which input MIDI data is used to query a corpus of symbolic musical material. A new accompaniment is generated driven by the input but using materials found in the corpus. In PyOracle Improviser, this notion is extended to operate on the audio signal itself, and without relying on a preanalyzed corpus. In query mode, the pitch-content (chroma) of the input signal is captured and used as a guide for oracle navigation. When query mode is active, each Oracle navigation jump triggers a three-part search process:

1. The first aspect of the search compares past Oracle feature frames to the most recently received input (from the human musician). The indices of those frames which are determined to be similar enough, according to a query distance threshold, are stored.
2. Next, the algorithm iteratively backtracks along suffix links starting from the current state k, and collects states which are connected via suffix, reverse suffix, and transition links.
3. Finally, the next state is determined with a random choice made from the intersection of the set of states with similar features and the set of connected states. If there are no states in common, the oracle simply advances to the next state $k + 1$. The function described returns the index of the next state for navigation.

It was found that the query distance threshold should be set slightly higher than the ideal distance threshold for the chroma feature. When set in this manner, less precise matches will be found, which may have produced a less ideal oracle structure but are still similar enough to be musically meaningful. The final mode is **follow** mode. Like the mode of the same name found in OMax, this mode creates a sliding window which advances automatically and constrains oracle navigation to only the material most recently played by the human improviser.

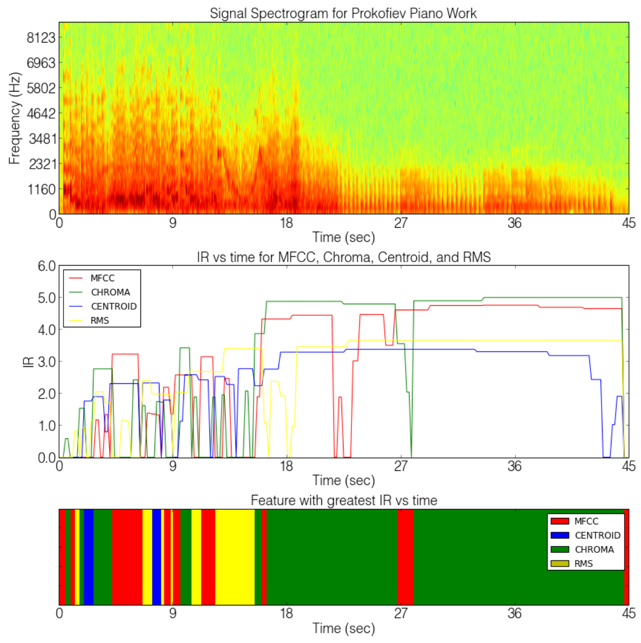

Figure 8: Regions set according to feature IR. The feature with the highest IR value is considered to be the most informative of a given segment.

Feature Selection Using IR

Since PyOracle Improviser learns across multiple features simultaneously, but only generates music using a single oracle built on a single feature, it is problematic to determine which feature should be used at a given time. Over the course of a composition or improvisation, particular features may change in relative importance. For example, an improviser might begin a performance by focusing on manipulations and variations on a large set of pitches, before transitioning to a more timbrally-focused playing style with less pitch diversity. When navigating the learned oracles for generative purposes, the feature chosen while navigating an oracle segment should correspond to the musical material performed during the learning of that segment. In this section, we propose a method for the automatic detection of the most "important" feature during a given segment.

Recalling the above discussion of IR, we consider that the oracle which maximizes total IR is in some way the best model of the musical signal. We extrapolate from here, and assume that the feature which has the highest IR at a given moment is the feature which gives us the most information about the music at that moment. Consider Figure 8: the uppermost plot shows a spectrogram of the Prokofiev work used above. The second plot shows the evolution of IR over time for four signal features: MFCCs, chroma, spectral centroid, and RMS amplitude. The third plot segments the composition into blocks, with each block colored according to the feature with the highest IR.

Both plots contain potentially useful information. In a situation where regions are manually defined, by a human machine operator taking part in a musical performance, the second plot provides a visual reference for which regions of the audio signal are best modeled by which features. For example, consider the segment running from the sharp drop followed by an abrupt increase in chroma IR (green) at approximately 28 seconds. From here to the end of the piece, chroma is shown to provide the most information about the signal, followed by MFCCs (red), RSM amplitude (yellow), and finally spectral centroid (blue). This information can be used creatively and in multiple ways. If the machine operator has constrained oracle navigation to a region encompassing the segment just defined, the operator knows that switching the oracle to the one built on chroma will create a situation where the oracle navigation is able to make many continuations and recombinations of the original material. On the other hand, since the MFCC IR is very close to that of chroma, the operator may switch between the two oracles, depending upon the musical context he or she is working within. If the human improviser is playing pitch-based material, perhaps it makes most sense to navigate according to chroma, while if the material is more spectrally oriented, the MFCC oracle could be used to mirror the organization of the current input material. The third plot, containing blocks or segments, can be useful in driving a semi-autonomous machine improviser. Beginning from an initially selected feature, these segments can be used to provide transitions from one feature oracle to another. Upon a jump from a segment found to emphasize one feature to a segment which emphasizes another, the oracle can switch to the new feature, and will therefore have a new oracle with new links and transitions to navigate. Alternately, the segments themselves could be used to automatically determine region constraints for oracle navigation. Since the segments correspond to contiguous areas in which one feature was more prominent or informative than others, they could be seen as corresponding to specific musical materials. Through some automatic scheme, this knowledge could be used to provide a generative structure to a semi-autonomous machine improviser. It remains to be seen how well these segments would correspond to the results of a traditional musical analysis, but it seems likely that they reveal interesting larger structures of the signal.

Case Study: *Nomos ex Machina*

Nomos ex Machina is a structured improvisation which was composed for the Musical Metacreation Weekend, held in June, 2013. A graphical score indicating the structure of the improvisation is shown in Figure 10. This score is intended for a human performer to read, and is accompanied by a script file which automates PyOracle Improviser. The total duration is approximately eight minutes, and is broken into eight sections, two of which are repeated. The human improviser's part is indicated on the upper line, while the oracle behavior is indicated on the lower line. For the human, the sections in the score indicate time-brackets within which he or she is free to play. Within each section, the performer should aim to produce distinct, focused musical material. This emphasis on focused material is important when considering the way the script automates the PyOracle Improviser. In terms of the software, each section is primarily demarcated by the use of the *region* parameter described above. The region for each section corresponds to the material played during a previous section or sections. The piece unfolds as a series of introductions and interactions of new material which is layered with previously heard material recombined via the oracle. As mentioned previously, PyOracle Improviser builds oracles on multiple audio features simultaneously. In the case of the two repeated sections, sections five and six, the same structure is performed twice, but with a change in oracle feature to shift the emphasis from pitch-based material to timbral material. The instrumental-

Figure 9: Peter Farrar and Ollie Bown soundchecking "Nomos ex Machina." Photo courtesy Ben Carey.

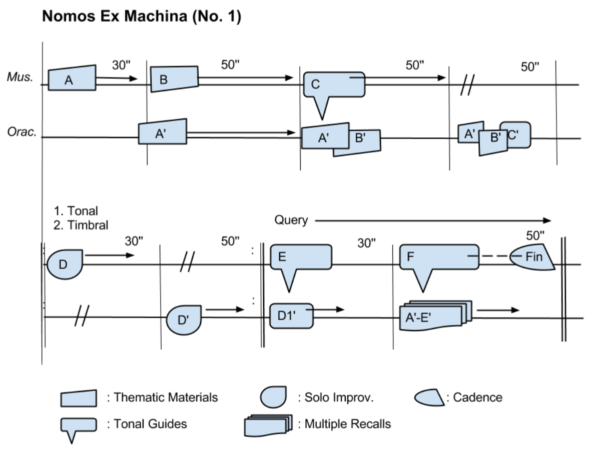

Figure 10: Score for *Nomos ex Machina*.

ist is asked to mirror the shift in emphasis. Finally, the last sections of the piece use the query mode. At this point in the performance, there is a variety of material for the query algorithm to choose from, which helps to ensure better matches.

This method of composition / improvisation structuring presents some unique challenges to the human performer. Due to the strictly-timed progression of sections in the piece, the performer must develop materials which can be established or developed within a specific duration. Additionally, the material associated with each section will need to function when layered with previous and/or future materials.

During rehearsals, a few performers expressed a desire for a more flexible timing system. The automatic nature of the timing mechanism may feel inhibiting or limiting to a certain playing style. Additionally, it was found that if timing shifts during performance (i.e. if the performer begins counting a section before the software), it is possible to define erroneous regions containing silence or unintended sounds. One of our major future projects is to explore methods for adding flexibility or variability to the PyOracle Improviser score system. A few options are being considered:

- Enabling performers to advance through sections manually, via a MIDI footpedal or other controller. This would increase flexibility, at the cost of additional burden on the performer.

- Using some form of score-following mechanism to advance sections according to the musical performance. This would add significant design overhead during the composition of a new piece, as the score-following mechanism and its mapping to parameters would need to be specified.

- Automatic section changes, according to some compositional algorithm. This would not necessarily increase flexibility, but would provide variety in performance. This could be combined with the IR-based segmentation scheme outlined above.

Conclusion

PyOracle Analyzer and PyOracle Improviser use the Audio Oracle algorithm to enable music analysis and machine improvisation. Unlike many other machine improvisation programs, PyOracle Improviser does not use a symbolic or quantized representation of the musical events, but instead learns directly from musical signal features. The quality of an AO representation depends on the distance threshold used, and Music Information Rate can be used to determine the best threshold. Since IR is easy to derive from the AO structure, it is straight-forward to employ Music Information Dynamics considerations in music composition and improvisation. We have described a procedure for predicting the best model distance threshold of an improvisation, by learning an excerpt of similar material. IR can also be used to determine the most informative feature during a given segment of the audio. Through this knowledge, a human operator or algorithm can switch features during a performance to represent the changing musical materials. Finally, PyOracle Improviser presents a group of features enabling reproduceable compositions or structured improvisations. In addition to a set of modes defining specific modes of interaction, a scriping function has been implemented and used in a composition entitled *Nomos ex Machina*. The script enables the creation of compositional structures by automating parameters and constraints during performance. Though some performers have found found the timing mechanism to be too rigid, we have defined some possible methods and future directions for introducing flexibility into these structures.

Acknowledgements

Thanks to Ollie Bown and Peter Farrar for their rehearsals and performance during the MuME Weekend, 2013.

References

Abdallah, S., and Plumbley, M. 2009. Information dynamics: Patterns of expectation and surprise in the perception of music. *Connection Science* 21(2-3):89–117.

Allauzen, C.; Crochemore, M.; and Raffinot, M. 1999. Factor oracle: A new structure for pattern matching. In *SOFSEM99: Theory and Practice of Informatics*, 295–310. Springer.

Assayag, G.; Bloch, G.; Chemillier, M.; Cont, A.; and Dubnov, S. 2006. Omax brothers: a dynamic topology of agents for improvization learning. In *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, 125–132. ACM.

Dubnov, S.; Assayag, G.; and Cont, A. 2007. Audio oracle: A new algorithm for fast learning of audio structures. In *Proceedings of International Computer Music Conference (ICMC)*, 224–228.

Dubnov, S.; Assayag, G.; and Cont, A. 2011. Audio oracle analysis of musical information rate. In *Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on*, 567–571. IEEE.

Dubnov, S.; McAdams, S.; and Reynolds, R. 2006. Structural and affective aspects of music from statistical audio signal analysis. *Journal of the American Society for Information Science and Technology* 57(11):1526–1536.

François, A. R.; Chew, E.; and Thurmond, D. 2007. Visual feedback in performer-machine interaction for musical improvisation. In *Proceedings of the 7th international conference on New interfaces for musical expression*, 277–280. ACM.

Gifford, T., and Brown, A. R. 2011. Beyond reflexivity: Mediating between imitative and intelligent action in an interactive music system. In *25th BCS Conference on Human-Computer Interaction*.

Lefebvre, A., and Lecroq, T. 2001. Compror: compression with a factor oracle. In *Proceedings of the Data Compression Conference*, 502. IEEE Computer Society.

Lewis, G. E. 2000. Too many notes: Computers, complexity and culture in voyager. *Leonardo Music Journal* 10:33–39.

Meyer, L. B. 1956. *Emotion and meaning in music*. University of Chicago Press.

Pachet, F. 2003. The continuator: Musical interaction with style. *Journal of New Music Research* 32(3):333–341.

Potter, K. S.; Wiggins, G.; Pearce, M. T.; et al. 2007. Towards greater objectivity in music theory: Information-dynamic analysis of minimalist music. *Musicae Scientiae* 11(2):295–324.

Rigau, J.; Feixas, M.; and Sbert, M. 2008. Informational aesthetics measures. *Computer Graphics and Applications, IEEE* 28(2):24–34.

Silvia, P. J. 2005. Emotional responses to art: From collation and arousal to cognition and emotion. *Review of general psychology* 9(4):342.