Introducing CatOracle: Corpus-based Concatenative Improvisation with the Audio Oracle Algorithm

Aaron Einbond, Diemo Schwarz, Riccardo Borghesi, Norbert Schnell

Introducing CatOracle: Corpus-based concatenative improvisation with the Audio Oracle algorithm

Aaron Einbond
City University London
Aaron.Einbond@city.ac.uk

Diemo Schwarz
IRCAM/CNRS/UPMC
Diemo.Schwarz@ircam.fr

Riccardo Borghesi
IRCAM/CNRS/UPMC
Riccardo.Borghesi@ircam.fr

Norbert Schnell
IRCAM/CNRS/UPMC
Norbert.Schnell@ircam.fr

ABSTRACT

CATORACLE *responds to the need to join high-level control of audio timbre with the organization of musical form in time. It is inspired by two powerful existing tools:* CataRT *for corpus-based concatenative synthesis based on the* MUBU *for* MAX *library, and* PYORACLE *for computer improvisation, combining for the first time audio descriptor analysis and learning and generation of musical structures. Harnessing a user-defined list of audio features, live or prerecorded audio is analyzed to construct an "Audio Oracle" as a basis for improvisation.* CATORACLE *also extends features of classic concatenative synthesis to include live interactive audio mosaicking and score-based transcription using the* BACH *library for* MAX. *The project suggests applications not only to live performance of written and improvised electroacoustic music, but also computer-assisted composition and musical analysis.*

1. INTRODUCTION

One of the most influential paradigms in recent digital music making has been the notion of "reproduction" [1]. This includes processes of transcription, such as audio mosaicking. However, it could also be extended to reproduction of musical behavior: not only imitating sound in-the-moment, but as it unfolds in time.

A notable recent technique that lends itself to audio reproduction and transcription is corpus-based concatenative synthesis (CBCS); however, still missing is a better temporal logic for organizing synthesis based on musical structure. Individual samples are selected by targeting a list of associated features, however there is no inherent connection between the descriptors of one sample and a successive sample to be concatenated. [1]

At the same time, the *Factor Oracle* (FO) algorithm [2] has proven a successful approach to realtime pattern-recognition, most notably applied musically in OMAX [3]. Could a factor-oracle-based system be used to augment realtime CBCS, permitting a predictive logic for synthesis? Our goal is to build on the wealth of timbral detail available through CBCS along with the pattern-generating capabilities of the FO to create a flexible tool for realtime synthesis, improvisation, computer-assisted composition, and musical analysis.

[1] David Wessel, personal communication, 23 March 2012.

2. PREVIOUS WORK

The approach presented here draws on some of the most versatile existing tools for realtime interaction: CATART for CBCS and the OMAX/PYORACLE for computer-assisted improvisation.

2.1 Corpus-Based Concatenative Synthesis

CBCS systems such as CATART [4] build up a database of prerecorded or live-recorded sound by segmenting it into *units*, usually of the size of a note, grain, phoneme, or beat, and analysing them for a number of sound *descriptors*, which delineate their sonic characteristics. These descriptors are typically pitch, loudness, brilliance, noisiness, roughness, spectral shape, or meta-data, like instrument class, phoneme label, that are attributed to the units, and also include segmentation information. These sound units are then stored in a database (the *corpus*). For synthesis, units are selected from the database that are closest to given *target* values for some of the descriptors, usually in the sense of a weighted Euclidean distance. The selected units are then concatenated (overlapped) and played, possibly after some transformations. CBCS has the advantage of combining the richness and nuances of recorded sound with a direct and meaningful access to specific sound characteristics via high-level perceptual or musical descriptors.

2.2 Factor Oracle

OMAX has proven a dynamic tool for combining realtime computer-performer interaction with high-level musical representation. It first requires a "learning" phase during which audio input (for example from a live performer) is recorded, segmented, and the FO structure is calculated. The "improvisation" phase follows, in which the FO recombines the recorded segments of audio to produce new permutations of material. "Learning" and "improvisation" can overlap, so that as further audio input is added, the FO is extended as a basis for later improvisation. Multiple im-

provisations can be generated simultaneously, polyphonically, from the same underlying FO.

Improvisation with the FO algorithm has been described in detail elsewhere [3, 5]: the central idea is that at each segment or *state* of the improvisation, the oracle can jump along forward *transitions* to states with shared context, along *suffix* links back to states with the longest shared past, or continue to the next adjacent state. The choice among these available states is determined by user-defined probabilities and thresholds.

OMAX can take as an input symbolic MIDI pitches, or live audio signal analyzed with the YIN algorithm and *Mel Frequency Cepstral Coefficients* (MFCCs), complementing the pitch estimate with a spectral description [5]. Building on this work, we introduce a more extensive and customizable list of descriptors, especially for timbral features, to facilitate computer improvisation in contexts where pitch descriptions are inadequate: "computer noise improvisation." In the tradition of CATART, we propose that user-defined and weighted descriptor choices offer powerful creative advantages over features that describe the timbre as a whole such as MFCCs, as each of them can describe a specific aspect of the sound.

2.3 Audio Oracle

The *Audio Oracle* (AO) algorithm is an extension of FO optimized for processing of audio signals [6]. FO and AO rely on parsing the incoming signal into an *alphabet* of states; however, when instead of MIDI values, continuous ranges of descriptors are used, this becomes a non-trivial task. One of the most powerful features of AO is that it uses concepts from music information geometry to calculate an ideal distance threshold based on *information rate* (IR), a measure of "the reduction in uncertainty about a signal when past information is taken into account" [7]. Units with descriptor values within this threshold are grouped into the same state, or *letter* of the oracle alphabet.

The AO algorithm has been implemented in the freely distributed PYTHON library PYORACLE [2] [7]. In addition to providing a flexible collection of code for audio processing, PYORACLE also includes the MAX patch *pyoracle_improviser*, which allows PYTHON scripts to be called using the *py/pyext* externals. [3] The resulting improvisation tool shares many features with OMAX, but now using the AO algorithm with features calculated with the *Zsa.descriptors* library including pitch, amplitude, MFCC, spectral centroid, zero-crossing, and chroma. However these features can only be selected one-at-a time and not combined. Once the desired feature has been chosen, the AO requires an initial "training" phase: an example of the audio input is analyzed for roughly one minute, in order to calculate the IR-based distance threshold that will be used to analyze audio afterwards. Afterwards learning and improvising proceed as with OMAX.

Another innovative feature of PYORACLE, shared by the SOMAX project [8], is context sensitivity: improvisation is informed both by past events in the oracle, and simultaneously by the current audio input. For example in pitch-focused music this could encourage improvisation that blends with the immediate harmonic context of a live improvisation partner.

2.4 How they work together in CatOracle

The key to combining CBCS with the FO or AO algorithm is to associate *units* in CATART with states of the oracle. As mentioned above, for real-valued descriptors (as opposed to MIDI) multiple states are grouped together into letters to form an alphabet. Units may also correspond to multiple states: while this would not occur with a live input, where each new unit is unique, it could occur when the input is based upon a pre-recorded corpus, in which the same unit could be repeated multiple times (see Figure 7(b) below). These correspondences between units, states, and letters are stored in PYTHON arrays and MAX *coll* objects. Once an AO has been learned, these data can be saved for later use so subsequent improvisations can be performed without repeating training or learning phases.

As in *pyoracle_improviser*, CATORACLE incorporates a PYTHON script with the *py* MAX object. In order to support user-defined and weighted descriptors, the PYORACLE code has been adjusted to accept an incoming list of descriptors of arbitrary length and units. Each descriptor may be weighted by the user with a *multislider* object (see Figure 2 below). During the training phase, the incoming descriptors are normalized (either based on minimum and maximum values or mean $\pm$ standard deviation) and scaled by descriptor weights before the AO distance threshold is calculated. While this straightforward approach might produce statistical infelicities if descriptors are not fully independent, it is nevertheless advantageous for the subjective control it permits. As with other features of CATORACLE, the user's creative aural judgements are favored over theoretical criteria.

CATORACLE adopts the approach to context-sensitivity implemented in PYORACLE, but taking advantage of CATORACLE's extended descriptors for timbrally rich music. During improvisation, the list of next available oracle states is filtered based on a comparison with the descriptors of the incoming audio signal. Only states falling within a chosen descriptor distance, the "query threshold," are permitted for the oracle's next jump.

3. IMPLEMENTATION

After evaluating several potential architectures, it was decided that that CATORACLE would be implemented with the MUBU library for MAX and PYORACLE. This offers the efficiency and modularity of MUBU with the easy legibility and customizability of PYTHON code.

3.1 MuBu and PiPo

Multi-Buffer [9] is a multi-track container library, representing multiple synchronised data streams. A particular track might represent audio samples, a single audio descriptor or a vector of descriptors, markers or any other stream of numerical data associating each element of the stream to a precise instant in time.

The freely available binding of MUBU for MAX comes with a number of graphical visualisers/editors and externals that allow granular, concatenative, and corpus-based

[2] https://pypi.python.org/pypi/PyOracle/5.5

[3] http://grrrr.org/research/software/py/

synthesis. Paired with the PiPo (*Plugin Interface for Processing Objects*) framework, analysis of audio descriptors and segmentation can be performed in realtime or in batch on a whole collection of sound files.

We implemented CBCS in realtime in our CataRT system, [4] now rebased on MuBu and PiPo (Figure 1). [5]

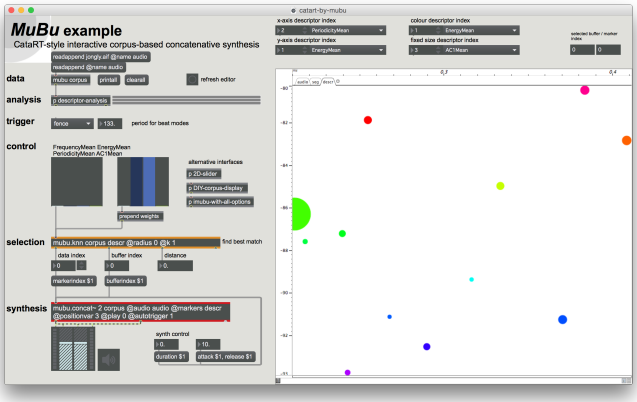

Figure 1: Screenshot of *catart-by-mubu*.

3.2 CatOracle Patch Structure

CataRT is distributed with MuBu in the examples folder. [6] It takes advantage of MuBu's modular structure, with multiple objects accessing the same multi-buffer data structure through a shared argument (Figure 2).

3.2.1 *Live Input*

An extension of classic CBCS is realtime control using live audio to search the corpus. When descriptors are compared for closest matches between units, this could be termed realtime "audio mosaicking." Already implemented in CataRT for *FTM&Co* with the module *catart.analysis* [10], this process can now take advantage of the symmetrical architecture of MuBu and PiPo for even more transparent control of identical parameters for deferred- and realtime analysis and segmentation.

In CataRT two segmentation methods are provided for both: "chop," which segments periodically by a specified duration; and "onseg," a simple attack detector on a specified descriptor threshold (by default based on amplitude in decibels, but reconfigurable by the user to any descriptor). The descriptors values are compared in the *mubu.knn* external, which constructs a kD-tree on the prerecorded corpus for efficient comparison with the live input to find the k-nearest-neighbors for each incoming unit. Following previous work with CataRT [11], analysis can be carried out in "targeted-transposition" mode, where differences in frequency and energy between corpus and target descriptors are taken into account before re-synthesis. These data can be stored in bach slots (see Section 3.2.5) and later edited to affect playback.

[4] http://ismm.ircam.fr/catart/
[5] http://ismm.ircam.fr/mubu, http://ismm.ircam.fr/pipo/
[6] http://forumnet.ircam.fr/product/mubu-en/

3.2.2 *Audio Descriptors*

By default the analysis subpatches are set to use *pipo.basic*, providing as descriptors: frequency, energy, periodicity, autocorrelation, loudness, centroid, spread, skewness, and kurtosis. This allows CataRT to run entirely within the free MuBu distribution. Other descriptor calculations may be customized by replacing the PiPo module with *pipo.yin*, *pipo.moments*, or *pipo.mfcc*. Or, with a software license, the full range of the IrcamDescriptors library [12] is available with *pipo.ircamdescriptors*. [7]

A subpatcher with convenient checkboxes for descriptor selection may be substituted for the existing analysis modules in CataRT allowing access to spectral, temporal, or many other features in any combination (Figure 3).

3.2.3 *Key Values*

For large corpora, tagging individual sound files with metadata can be an invaluable tool for navigation: for example, to organize an orchestral sample library by instrument name. For this purpose CataRT includes the subpatch *select-by-key* to enable and disable parts of the corpus. This takes advantage of the key-value data structure of MuBu. When loading a new sound file (or folder containing sound files) to the corpus, an arbitrary key-value pair may be entered through a *textedit* object. Or the key "SoundSet" may be assigned automatically with its value set to the directory of the file, thereby allowing to group sounds beforehand. These values are saved and reloaded with the corpus. Then the sounds matching a given key-value pair can be enabled or disabled by a checkbox.

3.2.4 *iMuBu View*

Multi-buffers can be viewed using the graphical interface object *imubu*. Within CataRT, this object is accompanied by useful presets to view the waveforms of individual sound files in the corpus (wave view) with their segmentation *markers* (equivalent to units in classic CataRT). Or, inspired by the CataRT *lcd* view, markers may be viewed in a scatterplot with user-chosen descriptors as $x-$ and $y-$positions, $x-$ and $y-$widths, or color. Transparency is used to indicate sound files (and their markers) that have been disabled by key-value (Figure 4). From both wave view and scatterplot view, a mouse or other controller can be used to select markers for playback through *mubu.concat~*.

3.2.5 bach *Transcription*

In previous work, CataRT was connected to the bach library [8] to build a DAW-like interface for concatenative synthesis based on musical score notation [13]. A similar procedure was implemented in CataRT: in summary, units or markers are represented as note heads on a musical staff, using either *bach.roll* or *bach.score*. Along with frequency mean (pitch), energy mean (dynamic), and duration, any other descriptor data and metadata can be saved with each note in its *slots*. In particular, the indices

[7] http://forumnet.ircam.fr/product/max-sound-box-en/
[8] http://www.bachproject.net

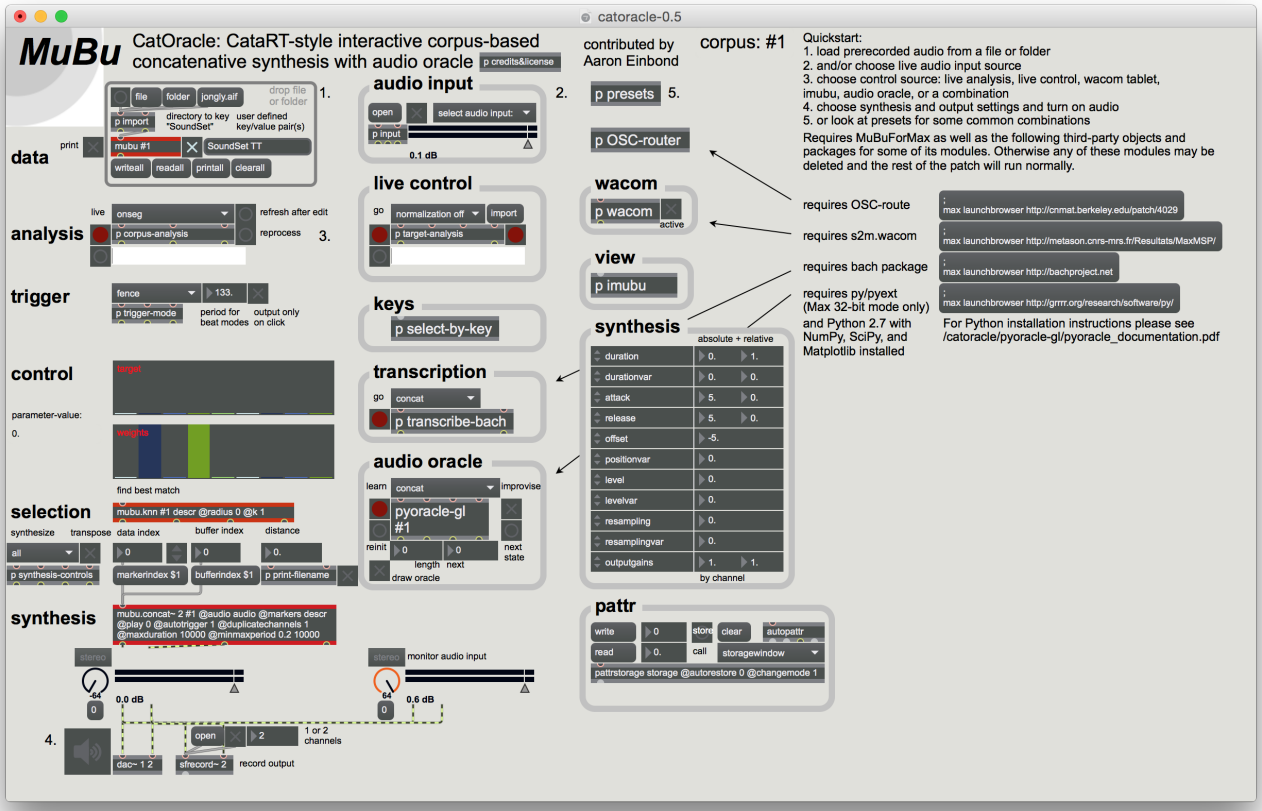

Figure 2: Screenshot of CATORACLE main patch.

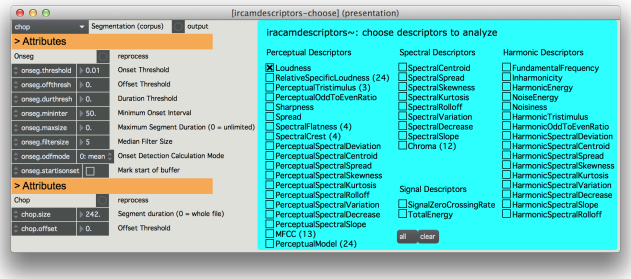

Figure 3: *pipo.ircamdescriptors* analysis subpatch.

of the marker and buffer are saved with each note, permitting playback from BACH through *mubu.concat*~. This information and other slot contents, for example source filename, can be displayed directly in the roll or score. Checkboxes permit quick selection of permitted rhythmic values with *bach.quantize*. Taking advantage of *bach.score's* proportional spacing attribute (@spacingtype 2), the roll and score are aligned rhythmically by default (Figure 5). From *bach.score* a MusicXML file can be exported, including slot metadata like dynamics and textual annotations, for further editing (see corresponding passage in Figure 9)

Combined with the audio oracle, this interface now offers new potential improvisation scenarios. For example, a computer improvisation can be transcribed in music notation for later use in computer-assisted composition (see Section 4.2 below). Or a transcription, as it is generated in real time, could be read by a human instrumentalist for acoustic playback (see Section 5 below).

3.2.6 Audio Oracle

The agent for computer-assisted improvisation is contained in the abstraction *pyoracle-gl*. As described above, descriptor values are received from other modules in the patch (*pipo*, *mubu.knn*, or *imubu*) depending on the scenario. They are normalized and weighted before being sent to the AO. The "queryae-gl" script loaded in the *py* external calls functions from the PYORACLE library to calculate the ideal distance threshold, learn the oracle, and generate the next state for improvisation. The module features a number of control parameters common to OMAX and PYORACLE: the probability of linear continuity versus jumping along the oracle, restriction to a region of the oracle, and forbidding repetition of the n most recent states with the "taboo" parameter (Figure 6). Due to the hybrid nature of CATORACLE the timing of improvisation can be controlled in several ways: durations can be reproduced from the durations of the learned oracle, durations can be taken from the pre-recorded corpus (possibly affected by *mubu.concat* synthesis attributes), or the oracle can wait to be triggered externally to advance to the next state.

The oracle can be visualized graphically using *Jitter* OpenGL objects for computational efficiency. These images, inspired by OMAX and PYORACLE show incisive views of musical structure, with forward transitions above

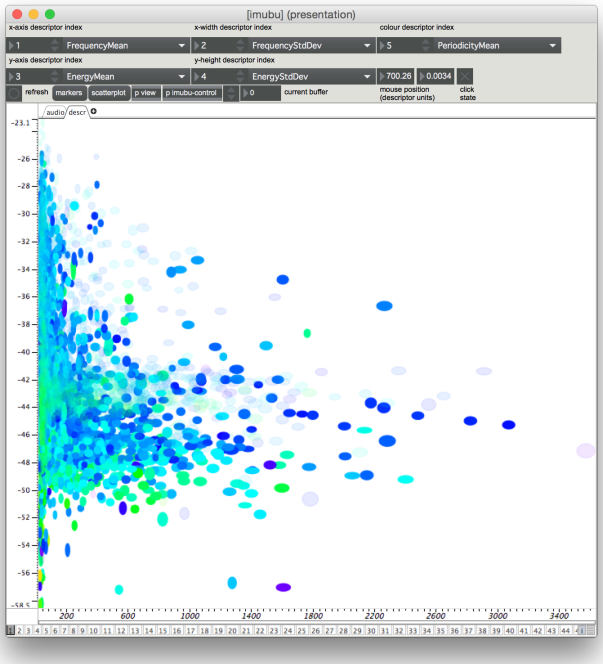

Figure 4: *iMuBu* scatterplot showing a corpus with some sound files (and their markers) disabled (transparent).

and suffix links below. The shaded ball represents the current state of an improvisation, and the shaded rectangle corresponds to a region to which improvisation is restricted (Figure 6).

3.2.7 Additional Features

Further features improve the user interface and performance of CATORACLE: communication through OSC messages using *OSC-route*, [9] control of *imubu* with a WACOM tablet using the *s2m.wacom* external, [10] *attrui* objects to control the granular-synthesis-style parameters of *mubu.concat*, and *pattr* objects with bindings to these attributes as well as other important parameters in the patch for convenient saving and reloading of preset scenes.

3.3 CataRT-MuBu-Live

An additional "light" version of the patch, entitled *catart-mubu-live*, is made available without the Audio Oracle algorithm and with no dependencies on any third-party libraries and externals. The remaining patch, requiring only MUBU and the standard MAX distribution, still retains the other features of CATORACLE, notably live analysis of an incoming audio signal for audio mosaicking, live recording the corpus, and an expanded list of triggering methods, as well as the tagging system provided by key-value pairs in MUBU. Furthermore, it avoids the limitation of the *py* external to 32-bit mode, and so can be used with MAX in 64 bits. It complements the even more streamlined *catart-by-mubu* and more elaborate CATORACLE, and all three are distributed in the MUBU examples folder.

[9] http://cnmat.berkeley.edu/downloads
[10] http://metason.cnrs-mrs.fr/Resultats/MaxMSP/

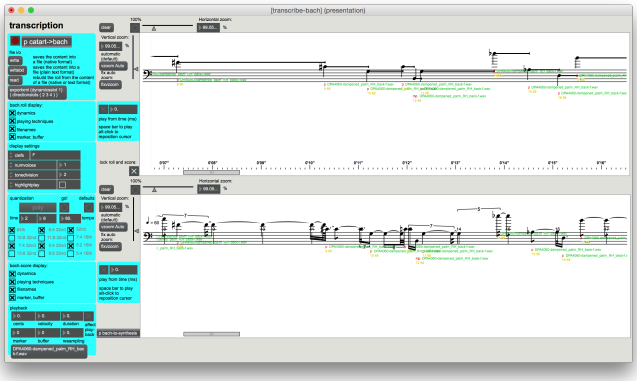

Figure 5: Transcription subpatch showing markers displayed in *bach.roll* and *bach.score* with slots for metadata.

4. MUSICAL APPLICATIONS

A range of applications extend the existing capabilities of CATART, OMAX, and PYORACLE as outlined in Figure 7.

(a) Depicts a process similar to OMAX: the performance begins with an empty corpus and the oracle is learned from a live audio input, stocking both the audio corpus and the oracle structure upon which improvisation is to be based.

(b) Represents a variation taking advantage of CBCS: a pre-recorded corpus is used in place of a live input. The oracle is learned from a musical sequence generated from the corpus, activated by a gestural controller such as a mouse or WACOM tablet. No new audio is recorded, but the oracle is recorded and used to generate an improvisation based on the same corpus.

(c) Combines (a) and (b): again the process begins with a pre-recorded corpus. But instead of a gestural controller, live audio input is used to control the initial musical sequence: for example through a live audio mosaic, comparing the live input to the closest matches in the corpus. No new audio is recorded, but the recorded oracle captures the structure of the input in terms of its descriptors. This could be advantageous in a performance situation where realtime control is desired, but without the risk of recording audio in non-ideal conditions (see *Xylography* below). Or it could be used for a more radical interpretation of computer improvisation: to imitate the behavior of one musical sequence using completely different sound material, raising intriguing æsthetic as well as technical questions.

(d) Begins with an audio oracle generated through any of the previous methods. But when improvisation begins, a live audio input is taken as a guide for navigation using PYORACLE's "query mode," so that the improvisation is informed by the current audio context. For noise-improvisation, this could be used to guide the computer improvisation toward timbral fusion with the live input, especially effective with the expanded list of timbral descriptors available from *pipo.ircamdescriptors*. [11]

4.1 Comprovisation

The combination of pre-composed music with computer-assisted improvisation, or "comprovisation" [5], is well-

[11] See a video of context-sensitive noise improvisation with CATORACLE by violist Nils Bultmann at https://vimeo.com/157177493.

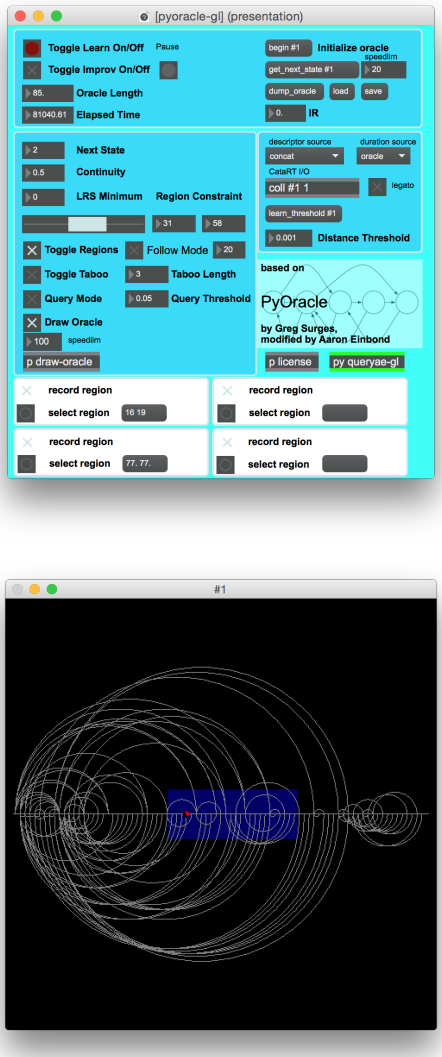

Figure 6: Audio Oracle abstraction showing (above) improvisation controls and (below) oracle visualization.

suited for CATORACLE. The first work to use the system for composition and performance is *Xylography* for violoncello and electronics by Aaron Einbond. [12] In this rigorously-composed work, no audio is recorded live: all of the electronics are generated from samples pre-recorded in the studio. However there is still a high degree of interactivity: audio oracles are learned in realtime, responding to the performer's fleeting variations in timbre and timing, especially relevant in a score with extended instrumental techniques. When the computer takes this as a basis for improvisation, it is informed by the performer's unique interpretation of the score. At times query mode is used to bring these improvisations into closer proximity with the performer as she continues playing from the notated score.

4.2 Computer-Assisted Composition

Xylography also makes use of computer-assisted composition: applying the notational capabilities of BACH,

[12] Written for Pierre Morlet and Séverine Ballon; videos available at http://medias.ircam.fr/xfb3c40 and https://vimeo.com/137971814.

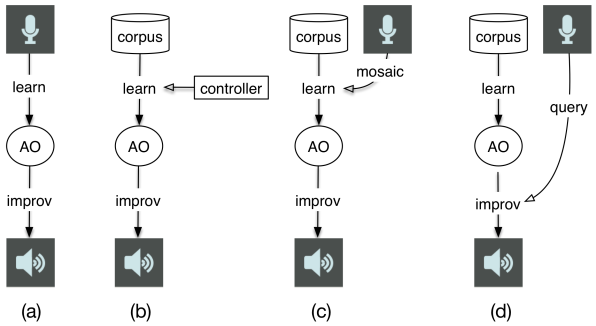

Figure 7: Paradigms of improvisation with CATORACLE.

computer-improvised sequences were transcribed as the basis for parts of the score to be performed acoustically. In this way, computer-improvisation becomes a technique for elaborating and developing acoustic material. In Figure 9, the first passage from the opening of the work is transcribed precisely from a recorded improvisation by the human performer. The second is transcribed from a computer-improvisation based on this recording, to be reinterpreted live by the performer near the end of the work. The intended effect is of a recapitulation, recognizable timbrally, but as if mis-remembered in its temporal sequence. The repetition and permutation of similar elements can be observed in spectrograms of the learned 'cello passage and the computer-improvised response (Figure 8), as well in the score, which has been edited in FINALE to render the graphical symbols (Figure 9).

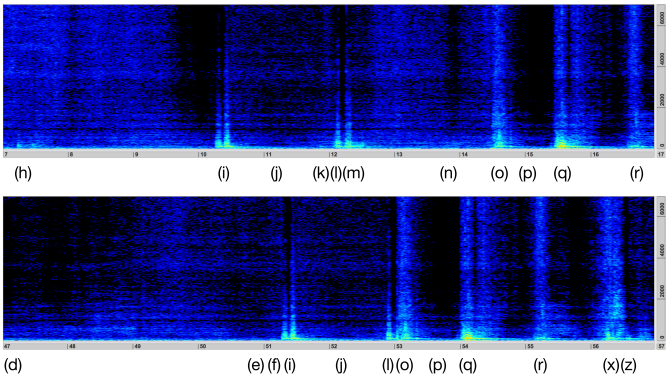

Figure 8: Spectrograms of learned and computer-improvised passages labeled with oracle states/markers.

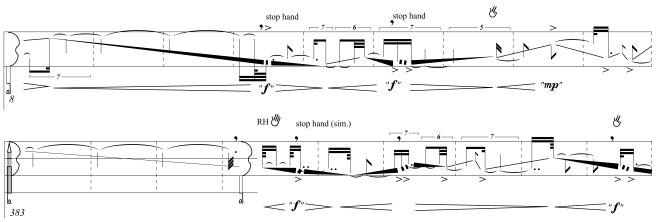

Figure 9: *Xylography* for 'cello and electronics, excerpts corresponding to those in Figures 5 and 8.

5. DISCUSSION AND FURTHER DIRECTIONS

A number of directions for further research could extend the implications of this project further.

The possibility of transcribing improvised sequences in music notation to be re-interpreted by a human performer in realtime has not yet been implemented in existing computer-assisted improvisation platforms. While the BACH package offers promising possibilities, further development will be necessary to refine the notation of dynamics, playing techniques, and realtime rhythmic quantization before it is useable in performance.

CATART'S potential for soundscape texture synthesis has already been proposed [14]. Could an AO algorithm offer a more "natural" reproduction of a soundscape, in effect imitating its behavior by permitting limitless renewal of non-repetitive textures? While no additional technical apparatus is necessary, listening tests must be employed to evaluate the effectiveness of potential results.

So far FO and AO algorithms have been used predominantly for musical creation. However their capacity for musical pattern identification and data reduction could also have uses for analysis of existing music, especially electroacoustic or timbrally rich music that still offers a challenge for existing techniques. In particular, the graphical representation of the oracle could be used to visualize large-scale formal and sonic connections. CATORACLE could be integrated with existing tools for digital analysis such as INDESCRIP or EANALYSIS [15] to provide another complementary view of musical structure.

Finally, FO and AO are only two of several oracle algorithms that could be evaluated. Another recent example is the Variable Markov Oracle (VMO) [16]. Or a related project is ImproTek [17], exploring the possibility of using pre-defined structures as templates for context-sensitive improvisation. While it could rely on a tonal structure, like a jazz progression, it could also follow an arbitrary trajectory of descriptors in time. Presently implemented in OPENMUSIC, it could exchange data with CATORACLE in the form of OSC messages. These alternative algorithms should be explored to determine how their results differ from CATORACLE and how they could be musically enriching.

Acknowledgments

We gratefully thank Séverine Ballon, Pierre Morlet, Arshia Cont, Benjamin Lévy, Gérard Assayag, Jean Bresson, Mikhail Malt, Emmanuel Jourdan, Paola Palumbo, Stephanie Leroy, Pascale Bondu, Aurèlia Ongena, Jérémie Bourgogne, Julien Aleonard, Sylvain Cadars, and Eric de Gélis. This paper is dedicated to the memory of David Wessel, mentor and inspiration for this work.

6. REFERENCES

[1] N. Donin, "Sonic Imprints: Instrumental Resynthesis in Contemporary Composition," in *Musical Listening in the Age of Technological Reproduction*, G. Borio, Ed. Farnham/Aldershot: Ashgate, 2015, pp. 323–341.

[2] C. Allauzen, M. Crochemore, and M. Raffinot, "Factor Oracle: A New Structure for Pattern Matching," in *Proceedings of SOFSEM99*. Springer-Verlag, 1999, pp. 291–306.

[3] G. Assayag, G. Bloch, M. Chemillier, B. M. Juin, A. Cont, and S. Dubnov, "Omax brothers: a dynamic topology of agents for improvization learning," in *ACM Multimedia Conference*, Santa Barbara, 2006.

[4] D. Schwarz, "Corpus-Based Concatenative Synthesis," *IEEE Signal Processing Magazine*, vol. 24, no. 2, pp. 92–104, 2007.

[5] B. Lévy, "Principles and Architectures for an Interactive and Agnostic Music Improvisation System," Ph.D. dissertation, Université Pierre et Marie Curie, Paris, 2013.

[6] S. Dubnov, G. Assayag, and A. Cont, "Audio Oracle: A New Algorithm for Fast Learning of Audio Structures," in *Proc. ICMC*, Copenhagen, 2007.

[7] G. Surges and S. Dubnov, "Feature Selection and Composition Using PyOracle," in *AIIDE Conference*, Boston, 2013.

[8] G. Assayag, "Keynote Talk: Creative Symbolic Interaction," in *Proc. ICMC*, Athens, 2014.

[9] N. Schnell, A. Röbel, D. Schwarz, G. Peeters, and R. Borghesi, "MuBu & Friends – Assembling Tools for Content Based Real-Time Interactive Audio Processing in Max/MSP," in *Proc. ICMC*, Montreal, 2009.

[10] A. Einbond, D. Schwarz, and J. Bresson, "Corpus-Based Transcription as an Approach to the Compositional Control of Timbre," in *Proc. ICMC*, Montreal, 2009, pp. 223–226.

[11] A. Einbond, C. Trapani, and D. Schwarz, "Precise Pitch Control in Real Time Corpus-Based Concatenative Synthesis," in *Proc. ICMC*, Ljubljana, 2012, pp. 584–588.

[12] G. Peeters, "A large set of audio features for sound description (similarity and classification)," IRCAM, Tech. Rep., 2004, unpublished.

[13] A. Einbond, C. Trapani, A. Agostini, D. Ghisi, and D. Schwarz, "Fine-tuned Control of Concatenative Synthesis with CataRT Using the bach Library for Max," in *Proc. ICMC*, Athens, 2014, pp. 1037–1042.

[14] D. Schwarz and N. Schnell, "Descriptor-based Sound Texture Sampling," in *Sound and Music Computing*, Barcelona, 2010, pp. 510–515.

[15] P. Couprie and M. Malt, "Representation: From Acoustics to Musical Analysis," in *EMS Network Conference*, Berlin, 2014.

[16] C. Wang and S. Dubnov, "Guided Music Synthesis with Variable Markov Oracle," in *AIIDE Conference*, Raleigh, 2013, pp. 56–62.

[17] J. Nika and M. Chemillier, "ImproteK, integrating harmonic controls into improvisation in the filiation of OMax," in *Proc. ICMC*, Ljubljana, 2012, pp. 180–187.